Financial Fraud Detection Using Batch Processing

Contents

1 Introduction

1.1 Overview

In today's digital financial landscape, detecting and preventing fraudulent activities is crucial for maintaining the integrity and security of financial transactions. Financial institutions and organizations process vast amounts of transaction data daily, making it challenging to identify fraudulent patterns manually. This project, **"Financial Fraud Detection Using Batch Processing"**, aims to leverage advanced data processing and machine learning techniques to identify suspicious activities and generate alerts for potential fraud.

1.2 Project Objectives

The primary objective of this project is to develop a robust batch processing system that can:

1. **Ingest Financial Data**: Efficiently import large volumes of financial transaction data from a relational database into Hadoop HDFS using Sqoop.
2. **Process and Transform Data**: Utilize Apache Spark to perform comprehensive data processing and transformation tasks.
3. **Query and Store Results**: Employ Apache Hive to query the processed data and store the results in a structured format.
4. **Apply Machine Learning Models**: Implement machine learning algorithms, such as Decision Trees or Support Vector Machines (SVM), to detect fraudulent patterns and predict potential fraud.
5. **Visualize Results**: Use a Business Intelligence (BI) tool to visualize the detection results and generate insightful reports for stakeholders.

1.3 Technologies Used

- **Sqoop**: Used for efficient data ingestion from relational databases into Hadoop HDFS.
- **Hadoop**: Provides scalable storage through Hadoop Distributed File System (HDFS) to handle large datasets.
- **Apache Spark**: A powerful data processing engine used to handle large-scale data transformations and processing tasks.
- **Apache Hive**: A data warehousing solution that provides SQL-like querying capabilities over data stored in Hadoop.
- **Machine Learning Algorithms**: Algorithms like Decision Trees and SVM will be used for fraud detection, analyzing patterns and anomalies in transaction data.
- **Business Intelligence (BI) Tools**: Tools to visualize detection results and generate reports for analysis and decision-making.

1.4 Importance of the Project

The ability to detect fraudulent activities is essential for financial institutions to mitigate risks and prevent financial losses. Traditional methods of fraud detection are often insufficient due to the sheer volume and complexity of transaction data. By utilizing modern big data technologies and machine learning, this project aims to build a scalable and efficient system capable of handling large datasets and providing actionable insights into fraudulent activities.

1.5 Expected Outcomes

By the end of this project, we expect to achieve:

- A fully operational batch processing system capable of ingesting, processing, and analyzing financial transaction data.
- Effective implementation of fraud detection algorithms that can identify and classify suspicious activities with high accuracy.
- Comprehensive reports and visualizations that offer valuable insights into transaction patterns and potential fraud.

This project will not only enhance the ability to detect and prevent fraudulent transactions but also contribute to a better understanding of how advanced data processing and machine learning techniques can be applied in financial contexts.

2 Ingest Financial Data

2.1 Database and Table Creation

2.1.1 Start and Enable MariaDB Service

The MariaDB service was started and configured to start on boot using the following commands:

```
sudo systemctl start mariadb sudo systemctl enable mariadb
```

2.1.2 Access MariaDB and Create Database

Logged into the MariaDB shell:

```
sudo mysql -u root -p
```

Created a new database named financial_fraud:

```
CREATE DATABASE financial_fraud; USE financial_fraud;
```

2.1.3 Create Table Schema

Defined the schema for the transactions table to match the data structure:

```
CREATE TABLE transactions ( step INT,
                                                    -- Step number or transaction
        ,→ step
    type VARCHAR(20),                               -- Type of transaction
    amount DECIMAL(15, 2),                          -- Amount of the transaction
    nameOrig VARCHAR(50),                           -- Originating account name/
        ,→ ID oldbalanceOrg DECIMAL(15, 2),
                                                    -- Balance before the
        ,→ transaction (origin) newbalanceOrig DECIMAL(15, 2),
                                                    -- Balance after the
        ,→ transaction (origin) nameDest
    VARCHAR(50),                                    -- Destination account name/
        ,→ ID oldbalanceDest DECIMAL(15, 2),
                                                    -- Balance before the
        ,→ transaction (destination) newbalanceDest
    DECIMAL(15, 2),                                 -- Balance after the
        ,→ transaction (destination) isFraud TINYINT(1),
                                                    -- Boolean (0 or 1) for fraud
        ,→ detection isFlaggedFraud TINYINT(1)      -- Boolean (0 or 1) for
        ,→ flagged fraud
);
```

2.2 Data Import into MariaDB

2.2.1 Prepare Directory for File Imports

Created and configured a directory for file imports: sudo mkdir -p /var/lib/mysql-files/ sudo chown mysql:mysql /var/lib/mysql-files/

2.2.2 Copy CSV File

Moved the CSV file to the import directory:

```
sudo cp ~/financial_fraud.csv /var/lib/mysql-files/
```

2.2.3 Load Data into MariaDB

Loaded the data from the CSV file into the transactions table:

ExecutedvariousSQLqueriestoverifythedata:

CountofRecords:

```
SELECT COUNT(*) FROM transactions:
```

PreviewRecords:

```
SELECT * FROM transactions LIMIT 10:
```

DistributionbyType:

```
SELECT type, COUNT(*) AS count
FROM transactions
```

```
LOAD DATA LOCAL INFILE '/var/lib/mysql-files/financial_fraud.csv'
INTO TABLE transactions
FIELDS TERMINATED BY ',' LINES
TERMINATED BY '\n' IGNORE 1
LINES;
```

2.2.4 Verify Data Import

```
GROUP BY type;
```

Distribution by Fraud Flag:

```
SELECT isFraud, COUNT(*) AS count
FROM transactions
GROUP BY isFraud;
```

Missing Values:

```
SELECT
        COUNT(*) - COUNT(nameOrig) AS missing_nameOrig,
        COUNT(*) - COUNT(nameDest) AS missing_nameDest
FROM transactions;
```

Amount Summary Statistics:

```
SELECT
        MIN(amount) AS min_amount,
        MAX(amount) AS max_amount
FROM transactions;
```

3 Data Ingestion into Hadoop HDFS Using Sqoop

3.1 Run Sqoop Import Command

Imported the data from MariaDB into Hadoop HDFS:

```
sqoop import \
    --connect jdbc:mysql://localhost/financial_fraud \
    --username student \
    --password student \
    --table transactions \
    --target-dir /user/hadoop/financial_fraud_data \
    --split-by step \
    --fields-terminated-by ',' \
    --lines-terminated-by '\n'
```

3.2 Verify Data in HDFS

Listed files in the target HDFS directory:

```
hadoop fs -ls /user/hadoop/financial_fraud_data
```

Previewed a sample of the imported data:

```
hadoop fs -cat /user/hadoop/financial_fraud_data/part-m-00000 | head -n 10
```

4 Preprocessing

First, we load the dataset.

```
data = spark.read.csv("PS_20174392719_1491204439457_log.csv", header=True, inferSchema=True)
data.show(5)
```

Python

```
+----+--------+--------+-----------+-------------+--------------+-----------+-------------+-------------+
|step|    type|  amount|   nameOrig|oldbalanceOrg|newbalanceOrig|   nameDest|oldbalanceDest|newbalanceDest|
+----+--------+--------+-----------+-------------+--------------+-----------+-------------+-------------+
|   1| PAYMENT| 9839.64|C1231006815|     170136.0|     160296.36|M1979787155|          0.0|          0.0|
|   1| PAYMENT| 1864.28|C1666544295|      21249.0|      19384.72|M2044282225|          0.0|          0.0|
|   1|TRANSFER|   181.0|C1305486145|        181.0|           0.0| C553264065|          0.0|          0.0|
|   1|CASH_OUT|   181.0| C840083671|        181.0|           0.0|  C38997010|      21182.0|          0.0|
|   1| PAYMENT|11668.14|C2048537720|      41554.0|      29885.86|M1230701703|          0.0|          0.0|
+----+--------+--------+-----------+-------------+--------------+-----------+-------------+-------------+
only showing top 5 rows
```

Figure 1: Loading the dataset

4.1 Handle Missing Values

We handle missing values by filling numeric columns with their mean.

```
# Handle missing values - fill numeric columns with their mean
numeric_columns = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']

# Calculate mean for each numeric column
for col_name in numeric_columns:
    mean_value = data.select(mean(col(col_name))).collect()[0][0]
    data = data.na.fill({col_name: mean_value})
```

Python

Figure 3: handle missing values

4.2 Address Data Imbalance

We noticed that our data is unbalanced because only 13% of transactions are fraudulent. We solve this problem by under-sampling the non-fraudulent data using these steps:

4.3 One-Hot Encoding

The **type** column has five unique string values: CASH-IN, CASH-OUT, DEBIT, PAYMENT, and TRANSFER. We use StringIndexer, OneHotEncoder, and VectorAssembler to transform the **type** column into a vector of columns, where each column in this vector represents a certain type. This vector of columns is suitable for machine learning models.

```
# Step 1: Index the 'type' column
indexer = StringIndexer(inputCol="type", outputCol="typeIndex")
indexed_data = indexer.fit(balanced_data).transform(balanced_data)

# Step 2: One-hot encode the indexed 'type' column
encoder = OneHotEncoder(inputCol="typeIndex", outputCol="typeVec")
encoded_data = encoder.fit(indexed_data).transform(indexed_data)

# Step 3: Assemble the features including the one-hot encoded vectors
feature_columns = ['step', 'typeVec', 'amount', 'oldbalanceOrg', 'newbalanceOrig',
                   'oldbalanceDest', 'newbalanceDest', 'balanceDiffOrig', 'balanceDiffDest']

assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Final dataset with features
final_data = assembler.transform(encoded_data)
```

Python

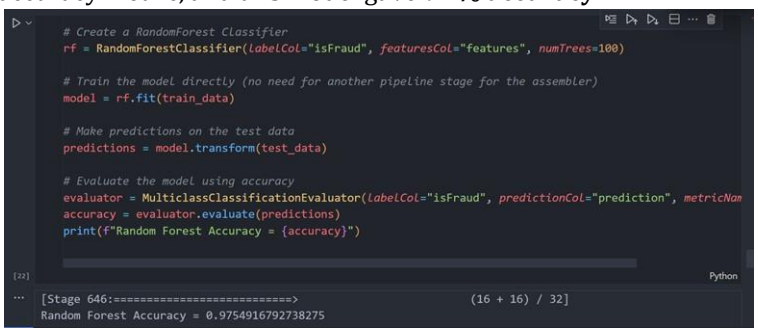
5 Machine Learning Models

First, we split the data into training and testing sets (80% for training and 20% for testing). Then, we create three classification models: Random Forest, Logistic Regression, and Gradient Boosted Tree.

5.1 Random Forest

We create an object from the RandomForestClassifier class, specifying the label column **isFraud**, the features column **features**, and the number of decision trees in the forest equal to 100. This classifier is designed to handle both binary and multiclass classification tasks and supports various hyperparameters to fine-tune the model.

We train the model on the training data and make predictions using the testing data. We evaluate these predictions using the accuracy metric, and this model gave **97% accuracy**.



5.2 Logistic Regression

We create an object from the LogisticRegression class, specifying the label column **isFraud**, the features column **features**, and the maximum number of iterations equal to 10. The Logistic Regression class supports multinomial logistic (softmax) and binomial logistic regression and includes methods for setting various hyperparameters to fine-tune the model.

We train the model on the training data and make predictions using the testing data. We evaluate these predictions using the accuracy metric, and this model gave **94% accuracy**.

5.3 Gradient Boosted Tree

We create an object from the GBTClassifier class, specifying the label column **isFraud**, the features column **features**, and the maximum number of iterations equal to 50. The GBTClassifier class sequentially builds trees, where each tree corrects the errors of the previous ones and includes methods for setting various hyperparameters to fine-tune the model.

We train the model on the training data and make predictions using the testing data. We evaluate these predictions using the accuracy metric, and this model gave **99% accuracy**.

```
# Train Gradient-Boosted Tree Classifier
gbt = GBTClassifier(labelCol="isFraud", featuresCol="features", maxIter=50)
gbt_model = gbt.fit(train_data)
gbt_predictions = gbt_model.transform(test_data)

gbt_accuracy = evaluator.evaluate(gbt_predictions)
print(f"Gradient-Boosted Tree Accuracy: {gbt_accuracy}")
```

Python

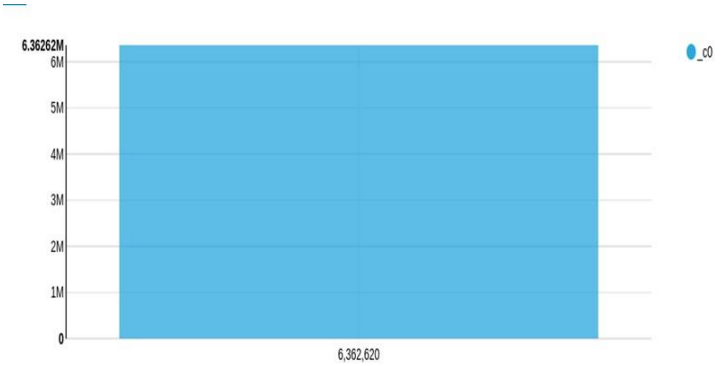
```
[Stage 597:=========================>                    (16 + 16) / 32]
Gradient-Boosted Tree Accuracy: 0.9921331316187595
```

6 SQL Queries After Importing Transactions into Hive

After importing the transactions dataset from MariaDB into Hive, the following SQL queries were executed to gain insights into the dataset.

6.1 Count Total Transactions

```
SELECT COUNT(*) FROM transactions;
```



This query counts the total number of transactions in the dataset.

6.2 Calculate Total Amount of Transactions by Type

```
SELECT type, SUM(amount) AS total_amount
FROM transactions GROUP BY type;
```

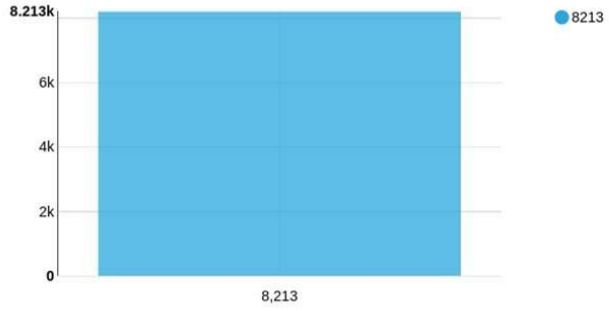
This query calculates the total transaction amount for each transaction type.

	type	total_amount
1	TRANSFER	485291987263.16833
2	CASH_OUT	394412995224.4885
3	CASH_IN	236367391912.4632
4	PAYMENT	28093371138.369926
5	DEBIT	227199221.28000027

6.3 Find Number of Fraudulent Transactions

```
SELECT COUNT(*) AS fraud_count
FROM transactions
WHERE isFraud = 1;
```

This query finds the total number of fraudulent transactions in the dataset.



6.4 Get the Top 10 Destinations with Highest Incoming Transactions

```
SELECT nameDest, SUM(amount) AS total_incoming
FROM transactions GROUP BY
nameDest
ORDER BY total_incoming DESC
LIMIT 10;
```

This query retrieves the top 10 destination accounts that have received the highest incoming transaction amounts.

6.5 Average Transaction Amount by Step

```
SELECT step, AVG(amount) AS avg_amount
FROM transactions GROUP BY
step;
```

This query calculates the average transaction amount for each transaction step.

7 Key Performance Indicators (KPIs)

- **Total Transactions:** 6.36M transactions have occurred over the time period.
- **Average Transaction Value:** The average transaction value is 179.86K in local currency.
- **Total Fraudulent Transactions:** 8213 transactions have been identified as fraudulent.
- **Average Balance Change:** The average balance change across transactions is 21.23K.
- **Total Fraudulent Transaction Value:** The total fraudulent transaction value amounts to 12.06bn.

8 Detailed Analysis

8.1 Amount by Transaction Type

- TRANSFER transactions account for the highest volume, followed by CASH-OUT, CASH-IN, and PAYMENT.
- DEBIT transactions contribute the least.

8.2 Fraud Analysis by Customer (isFraud)

- Several customers (IDs beginning with C1...) have engaged in fraudulent transactions, contributing smaller individual amounts to the overall fraud.

8.3 Amount by Customer Who Started the Transaction

- Customer ID C1717523... has initiated the highest number of transactions, with over 100M in total value.
- Several other customers (e.g., C2112746..., C2424764...) also have high transaction volumes.

8.4 Amount by Customer Who Received the Transaction (nameDest)

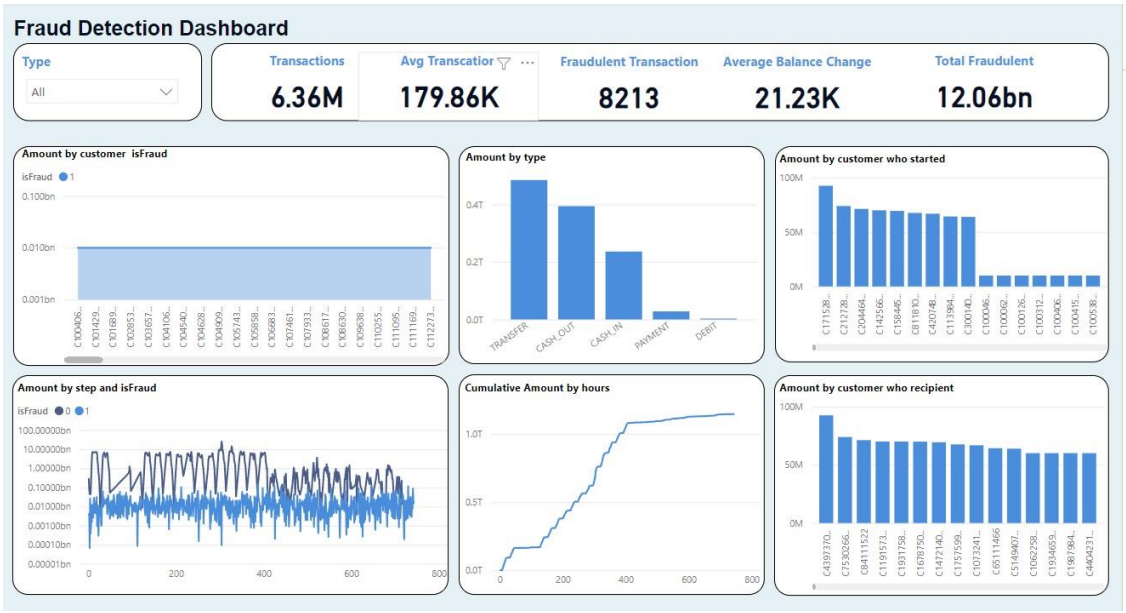
- Customer C4397706... has received the largest amount in transactions.
- Other recipients (e.g., C8711152..., C6411171...) follow closely.

8.5 Amount by Step and Fraud Status

- The plot of transaction amounts over time shows a fairly consistent volume of fraudulent (blue) vs. non-fraudulent transactions (black).
- Some spikes indicate periodic high volumes of activity, but fraud levels remain relatively steady.

8.6 Cumulative Amount by Hours

- The cumulative transaction volume shows a sharp increase at around 200 hours and continues to grow at a steady pace through 734 hours.
- This suggests a steady increase in transaction activities over time.



9 Conclusion

This project successfully developed a batch processing system that integrates various big data technologies, such as Sqoop, Hadoop, Spark, and Hive, to detect fraudulent financial activities. By implementing machine learning algorithms like Random Forest, Logistic Regression, and Gradient Boosted Trees, we were able to achieve high accuracy in fraud detection. The combination of data ingestion, processing, machine learning, and visualization tools provided a comprehensive approach to identifying and mitigating financial fraud. The results indicate that the Gradient Boosted Tree model, with 99

10 Future Work

- **Real-time Processing:** Extend the batch processing system to include real-time fraud detection using tools like Apache Kafka and Spark Streaming.
- **Additional Algorithms:** Experiment with other machine learning algorithms, such as XGBoost and Neural Networks, to further enhance detection accuracy.
- **Feature Engineering:** Investigate additional features that may provide more granular insights into fraud detection, such as transaction time and geographic location.
- **Deployment:** Deploy the system in a production environment and monitor its performance in detecting live fraudulent transactions.