Course Name: DSC 650

Student Name: Aniruddha Joshi

Week : 11-12

Date: Nov 10, 2024

# Introduction

The project aims to identify fraudulent credit card transactions. As we know, credit card transactions are a common risk in the finance industry, resulting in financial losses and customer dissatisfaction. The credit card transaction predictions are based on historical data, the user profile, and transaction location. This project uses the machine learning predictive model capable of identifying fraudulent transactions.

The project is based on loading the dataset from the Kaggle site by using nifi pipeline. The data is stored under HDFS and then moved to HIVE for querying, Spark is being used for training the model and storage of the results of the trained model to Hbase.

# Dataset

For week 11-12, I have selected the project on credit card fraud detection.

Dataset Link: <https://www.kaggle.com/datasets/bhadramohit/credit-card-fraud-detection>

Dataset: Credit Card Approval Prediction

Site: Kaggle.com

GitHub Link: <https://raw.githubusercontent.com/joshianiruddha/dsc650/refs/heads/main/credit_card_fraud_dataset.csv>

# Pipeline Overview

The project uses Apache infrastructure to load, store and perform machine learning predictions. Below are the details:

1. Data Ingestion: the data set is downloaded from at the Kaggle website location: <https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction/data>

The dataset (csv file) has been uploaded to the personal Github website: <https://raw.githubusercontent.com/joshianiruddha/dsc650/refs/heads/main/credit_card_fraud_dataset.csv>

Apache Nifi is used to ingest the data, ensuring efficient and reliable data transfer. The nifi reads the data and further passes it to HDFS storage.

1. Data Storage (HDFS): the data is read by the nifi GetFile function and then stored in HDFS (Hadoop Distributed File System). The file is stored under /ajprj directory
2. Data Transformation (Hive): Hive is then used to structure the data in a table with HDFS. This allows data to be queried using SQL-like syntax. This enables the data exploration and transformation.
3. Pyspark for data exploration: Pyspark is then used to further explore the data. The data is loaded into dataframe by querying the HIVE table. Pyspark is further used to explore the data and apply filtering.
4. Pyspark for training and machine learning: After exploring the data, the MLlib library in Pyspark is being used to train the linear regression model. The training results, such as r2 and RMSE, are being calculated.
5. The results of the training models are stored back in Hbase table

# Issues Encountered

Below are issues encountered during the process:

1. Loading the data in HDFS: I had difficulty locating the data on HDFS; however, after clarification and going through chat questions and documentation, I was able to identify the location of the data.
2. Data Load on Github: Initially, I selected data that was around 100MB. However, Github was restricted to 25MB. Finally, I found a dataset for the project that was within the limit of 25MB with around 10000 rows.
3. Google VM loads: The Google VM was unresponsive on multiple occasions. Restarting the VM helped with loading and training the data.
4. Training the data sets: I had some columns in the datasets that were strings. As I was using linear regression, the string columns caused exceptions in the code. Finally, removing some of the string columns and converting those to categorical columns helped the regression.
5. Writing to HBase: My write operation was not functioning properly, after rerunning the code, writing to Hbase was successful.

# Screencaps / Codes

### Https Data Load:

A screenshot of a computer

Description automatically generated

### HDFS Load Directory:

A screenshot of a computer

Description automatically generated

### Successful Data Load:

A screenshot of a computer

Description automatically generated

### To create a table structure in Hive:

CREATE TABLE creditfraud(

`TransactionID` DOUBLE,

`TransactionDate` DATE,

`Amount` DOUBLE,

`MerchantID` DOUBLE,

`TransactionType` STRING,

`Location` STRING,

`IsFraud` INT)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

tblproperties("skip.header.line.count"="1");

### Table structure in Hive

A screenshot of a computer

Description automatically generated

### Loading data in table

LOAD DATA INPATH '/ajprj/b7ff9a29-b894-48f5-bb9a-2f63719b08a6' INTO TABLE creditfraud;

### After loading the table in Hive:

A screen shot of a computer

Description automatically generated

### PySpark Code to connect with Hive table and querying the results:

Now, use this PySpark code to connect to the Hive table and query it:  
  
from pyspark.sql import SparkSession  
  
# Step 2.1: Create Spark session with Hive support  
spark = SparkSession.builder .appName("PySpark Hive Example") .enableHiveSupport() .getOrCreate()  
  
# Step 2.2: Query the Hive table  
df = spark.sql("SELECT \* FROM creditfraud limit 10")  
  
# Step 2.3: Show the result  
df.show()  
  
# Step 2.4: Stop the Spark session (optional)  
spark.stop()

### Query Results

A computer screen with white text

Description automatically generated

### Reading data from the table using Pyspark

A screenshot of a computer

Description automatically generated

### Machine Learning/Training model

### Installing libraries on worker 1

A screenshot of a computer

Description automatically generated

### Installing libraries on worker 2

A screenshot of a computer

Description automatically generated

### Installing libraries on the Master

**A screenshot of a computer

Description automatically generated**

### Code:

from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder

from pyspark.ml.regression import LinearRegression

#from pyspark.ml import Pipeline

from pyspark.ml.evaluation import RegressionEvaluator

# from pyspark.sql.functions import  hour, dayofweek

spark = SparkSession.builder .appName("MLlib Creditfraud Prediction") .enableHiveSupport() .getOrCreate()

# Step 2: Load the data from the Hive table 'gradesml' into a Spark DataFrame  
fraud\_df = spark.sql("SELECT Amount, MerchantID, TransactionType, Location, IsFraud FROM creditfraud")

fraud\_df.show(10)

### Query result

A screenshot of a computer screen

Description automatically generated

# Step 3: Handle null values by either dropping or filling them  
fraud\_df = fraud\_df.na.drop() # Drop rows with null values



# Step 4: Prepare the data for MLlib by assembling features into a vector

assembler = VectorAssembler(  
 inputCols=["Amount", "MerchantID"],  
 outputCol="features",  
 handleInvalid="skip" # Skip rows with null values  
)

A black screen with white text

Description automatically generated

assembled\_df = assembler.transform(fraud\_df).select("features", "IsFraud")



# Step 5: Split the data into training and testing sets  
train\_data, test\_data = assembled\_df.randomSplit([0.7, 0.3])



# Step 6: Initialize and train a Linear Regression model  
lr = LinearRegression(labelCol="IsFraud")  
lr\_model = lr.fit(train\_data)

A screen shot of a computer screen

Description automatically generated

# Step 7: Evaluate the model on the test data  
test\_results = lr\_model.evaluate(test\_data)  
# Step 8: Print the model performance metrics  
print(f"RMSE: {test\_results.rootMeanSquaredError}")  
print(f"R^2: {test\_results.r2}")

A screenshot of a computer program

Description automatically generated

Spark.stop()

A screen shot of a computer screen

Description automatically generated

### Write to HBase with Pyspark

### Log in to Worker Nodes and Install the Library

1. Log in to Worker Node 1:  
docker-compose exec worker1 bash

2. Install the Library using pip3:  
pip3 install happybase

3. Exit Worker Node 1:  
exit  
4. Log in to Worker Node 2:  
docker-compose exec worker2 bash

5. Install the Library using pip3:  
pip3 install happybase

6. Exit Worker Node 2:  
exit

### Log in to the Master Node and Install the Library

1. Log in to the Master Node:  
docker-compose exec master bash

2. Install the Library using pip3:  
pip3 install happybase

A screenshot of a computer screen

Description automatically generated

### Create the HBase Table

1. Open the HBase shell:  
   hbase shell
2. Create a table in HBase named my\_table with a column family cf:

create 'my\_table', 'cf'

Exit the hbase shell

exit

### Start the HBase Thrift Server

This command will be run in the master docker container.  
  
hbase thrift start &  
  
The & runs the Thrift server in the background.

### Write Data to HBase Using PySpark

import happybase  
from pyspark.sql import SparkSession  
  
# Initialize SparkSession  
spark2 = SparkSession.builder \  
 .appName('WriteToHBaseWithHappybase') \  
 .getOrCreate()  
  
# Example data (row\_key, column\_family:column, value)  
data = [('row1', 'cf:col1',test\_results.r2),  
 ('row2', 'cf:col1', test\_results.rootMeanSquaredError)]  
A computer screen with white text

Description automatically generated

# Function to write data to HBase inside each partition  
def write\_to\_hbase\_partition(partition):  
 connection = happybase.Connection('master')  
 connection.open()  
 table = connection.table('my\_table') # Update table name  
 for row in partition:  
 row\_key, column, value = row  
 table.put(row\_key, {column: value})  
 connection.close()

A screen shot of a computer code

Description automatically generated

# Parallelize data and apply the function with for each Partition  
rdd = spark.sparkContext.parallelize(data)  
rdd.foreachPartition(write\_to\_hbase\_partition)

A black screen with many small squares

Description automatically generated with medium confidence  
# Stop the Spark session  
spark.stop()

### Verify Data in HBase

After the script runs, you can verify the data written to HBase by using the HBase shell:  
  
hbase shell  
  
scan 'my\_table'

A black background with white text

Description automatically generated

# Conclusion

The project demonstrates that by using big data technologies such as Apache engine and machine learning, many transactions can be stored and analyzed in real time. By building a complete pipeline using NiFi, HDFS, Hive, Spark, and HBase, we have established a robust system for ingesting, storing, processing, training, and analyzing fraudulent activities. The use of HBase allows for real-time fraud prediction, enabling immediate action to be taken on suspicious transactions.

By further enhancing the infrastructure, a real-time transaction can be passed on to the trained machine-learning model to detect any fraudulent activities.