Credit Card Batch Processing in Banking System

Khem Poudel*, Movinuddin, Sanjana Gutta, Revanth Kumar Kommu, and Samir Poudel

Department of Computer Science, Middle Tennessee State University, Murfreesboro, TN, 37132, USA
{knp4k, m2b, sg6t, rk4y, sp2ai}@mtmail.mtsu.edu

Abstract. A batch payment is when you make one payment to many recipients at once using one bank account, which is more effective than making many separate payments to many recipients.
On your bank statement, the batch list transaction appears as a single debit. Batch payment processing speeds up payments and makes businesses happier. Sending a batch payment with a bank wire transfer is the most typical method. How to Process Batch Payments In contrast to real-time processing, which often takes more time and effort, batch processing is when a merchant sends all of their authorized credit card transactions via their credit card processor at the end of the business day or at a set time [1]. The authorization codes from all of the credit cards used by the merchant's clients may be submitted in one batch per day to the banks of each customer in order to receive permission. After approval, the money is sent to the business's bank to be paid. Our goal is to recognize fraudulent credit card transactions so that the customers of credit card companies should not be charged for the items they didn't purchase. The model used must be simple and fast enough to detect the anomaly and classify it as a fraudulent transaction as quickly as possible. . . .

Keywords: Batch Payment, Payment Processing, Bank Wire Transfer, Batch Processing, Credit Card Transactions, Authorization Codes, Fraudulent Transactions, Anomaly Detection

I INTRODUCTION

This Method Used to Process Credit Card Payments You must first comprehend the processing of credit card payments in order to comprehend batch payments. The procedure can be broken down into three parts: authorization, processing, and settlement, even if the steps may vary based on the payment processing provider chosen.

- 1. Authorization The initial step in using a credit card to make a purchase is authorization. The customer will pay using their credit card in exchange for the merchant services if batch payment processing is being used. The credit

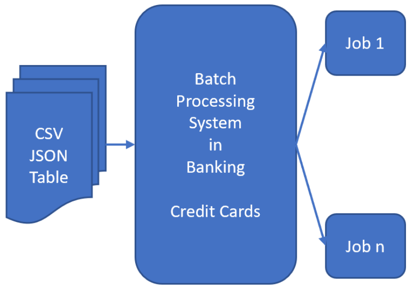


Fig. 1. Batch Processing

card details and transaction amount will then be sent by the POS to the credit card issuer (Mastercard, Visa, etc.).
In essence, the POS asks if the issuer believes the card is stolen, whether the card is real, and whether there is money available for this transaction. The card will then have a hold placed on it for the transaction amount, provided that it doesn't appear to be stolen, has enough money on it to cover the purchase, and isn't maxed out. The money has already been spoken for even if the transaction has not yet been completed.

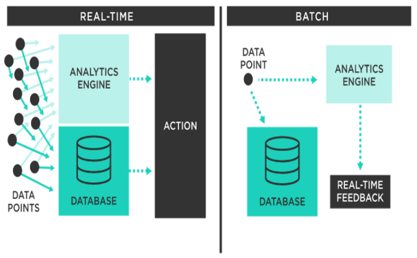


Fig. 2. Real Time vs Batch

– 2. Processing Processing credit cards comes next. The authorization hold is performed at this point in order to transfer the customer's funds to the merchant account.
– 3. Settlement Settlement occurs when the payment processor instructs the bank to transfer funds from one account to another. When the money reaches the seller's account, the transaction is settled, also known as closing the transaction.

Batch processing is when a computer processes a number of tasks that it has collected in a group. It is designed to be a completely automated process, without human intervention. It can also be called workload automation (WLA) and job scheduling [2].

Batch processing has a range of benefits, but it is ideal in businesses where:

– There's a process that doesn't need to be addressed immediately and real-time information isn't needed
– Large volumes of data need to be processed
– There's a span of time when a computer or system is idle
– A process doesn't need the input of humans and is repetitive
– A good example of batch processing is how credit card companies do their billing. When customers get their credit card bills, it isn't a separate bill for each transaction; rather, there is one bill for the entire month. That bill is created using batch processing [3]. All the information is collected during the month, but it is processed on a certain date, all at once.

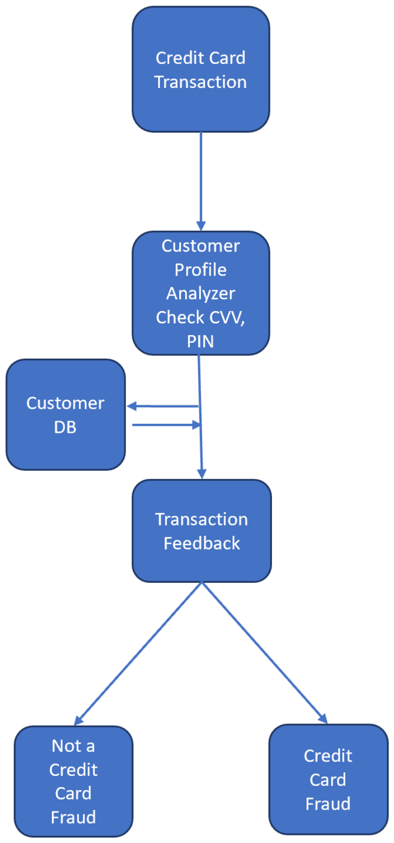


Fig. 3. Credit Card Fraud Detection Block diagram

Credit card transaction:

- This is the very first step where a customer uses his/her credit card for the transaction.
- This transaction made will be stacked on the merchant database.
- EOD, they will be sent for authorization.
- The merchant then submits the information of the credit card transaction to the gateway.
- The credit card gateway sends the transaction information to the merchant's bank processor

Customer Profile Analyzer/database: The merchant's bank submits this transaction to the network that is connected to the customer's bank. The customer's bank will accept or decline the payment by analyzing the customer's database. The network then analyzes the payment with different scenarios and checks the database of the customer.

II LITERATURE REVIEW

Susan Herbst-Murphy and the Payment Card Center held a meeting where senior MasterCard officials shared information with Federal Reserve payment analysts about clearing and settlement functions that MasterCard made for the customers. These functions involve the transfer of card transaction information (clearing) and the exchange of monetary value (payment) taking place between the bank, whose customer is the cardholder, and the bank whose customer accepts the card and receives the card. This document summarizes some of the key points of this meeting [1].

Niels Martin, Marijke Swennen, Benoît Depaire, Mieke Jans, An Caris, Koen Vanhoof discussed about a resource typically performs a specific operation on a series of instances. When a resource performs an operation on several instances concurrently, sequentially or concurrently, we talk about batch processing [2]. Because of its effect on process performance, batch processing should be considered when modeling business processes for performance evaluation purposes. This document recommends the event log as a source of information to better understand batch processing behavior. It marks the first step towards more in-depth support for retrieving batch processing knowledge from event logs by (i) defining different types of batch processing and (ii) describing Brief method of generating event log information.

Luise Pufahl, Mathias Weske describes an approach to defining requirements in batches not in a single process model, but to centrally define batch operations. Therefore, the lifecycle of the data object is used. Data-driven business processes; processing operations that read data and update it. Data operations allowed by process operations for data object class are defined centrally in data object's lifecycle. In this article, data object lifecycle is extended by batch conversion [3].

III KEY FINDINGS AND CONTEXT

A Credit Card Fraud and Recent Trends

Credit card fraud is a type of identity theft that occurs when someone other than you uses your credit card or bank account information to make unauthorized charges. Fraud can result from stolen, lost, or counterfeit credit cards. Additionally, with the rise of online retail, card fraud and the use of credit card numbers in e-commerce have become more common.

In the United States, credit card fraud has been the most common form of identity theft in four of the last five years. The United States is the country with the most fraud incidents, accounting for more than one-third of global card fraud losses. It is important to be knowledgeable about credit card fraud and identity theft so that you can practice good money habits and awareness in your daily life.

Key Credit Card Theft Findings: According to compiled key findings from the Federal Trade Commission's (FTC) Annual Data Book of 2020 to keep you informed about the frequency and severity of credit card fraud, as well as identified statistics about the populations who are most vulnerable to fraud:

- The most frequent payment method identified out of all fraud reports was credit cards.
- Credit card fraud made up a total of 459,297 reported instances of fraud and identity theft combined in 2020.
- 66,090 cases of reported fraud
- 393,207 cases of reported identity theft
- In identity theft cases, people ages 30-39 reported the most instances of credit card fraud while those age 80 and older reported the least.
- Instances of identity theft by credit card fraud increased by 44.6 percent from 271,927 in 2019 to 393,207 in 2020.
- Identity theft by new credit card accounts increased by 48 percent in 2020.

B PYSPARK AND APACHE AIRFLOW

A Python API for Apache Spark. Apache Spark is an analytical processing engine for large-scale powerful distributed data processing and machine learning applications.

- PySpark is a general-purpose, in-memory, distributed processing engine that allows you to process data efficiently in a distributed fashion.
- Applications running on PySpark are 100x faster than traditional systems.
- You will get great benefits using PySpark for data ingestion pipelines. Using PySpark we can process data from Hadoop HDFS, AWS S3, and many file systems.
- PySpark also is used to process real-time data using Streaming and Kafka. Using PySpark streaming you can also stream files from the file system and also stream from the socket.

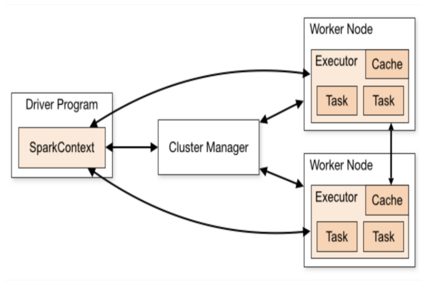


Fig. 4. PySpark Architecture

- PySpark natively has machine learning and graph libraries.

- Apache Spark works in a master-slave architecture where the master is called “Driver” and slaves are called “Workers”.
- When you run a Spark application, Spark Driver creates a context that is an entry point to your application, and all operations (transformations and actions) are executed on worker nodes, and the resources are managed by Cluster Manager.

Apache Airflow: Apache Airflow is already a commonly used tool for scheduling data pipelines. Airflow is using the Python programming language to define the pipelines. Users can take full advantage of that by using for loop to define pipelines, executing bash commands, using any external modules like pandas, sklearn, GCP, or AWS libraries to manage cloud services and much, much more.

Workflows can be created, scheduled, and monitored programmatically using Apache Airflow. Data engineers utilize it as one of the most reliable platforms for orchestrating workflows or pipelines. You can quickly see the dependencies, progress, logs, code, trigger tasks, and success status of your data pipelines. Users of Airflow can create workflows as task-based Directed Acyclic Graphs (DAGs). With Airflow’s sophisticated user interface, it’s simple to visualize pipelines that are currently in use, keep track of their progress, and address problems as they arise. When a task succeeds or fails, it can send an alert via email or Slack and links to different data sources. Given its distributed architecture, scalability, and flexibility, Airflow is a good choice for orchestrating complicated business logic.

Why Airflow? Airflow is a batch workflow orchestration platform. The Airflow framework includes operators that connect to many technologies and can be easily extended to connect to new technologies. If your workflow has a definite start and end and runs regularly, you can program it as an Airflow DAG.

If you’d rather code than click, Airflow is the tool for you. Workflows are defined as Python code. This means:

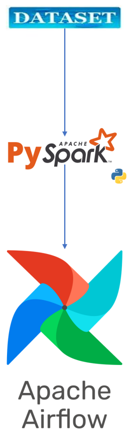


Fig. 5. Technology Flow: Dataset -> PySpark -> Airflow

- Workflows can be saved in version control so you can revert to previous versions
- Workflows can be developed by multiple people simultaneously
- You can write tests to verify functionality
- Components are extensible and can be built on a wide collection of existing components

Extensive scheduling and execution semantics make it easy to define complex pipelines that run at regular intervals. Backfilling allows you to (re)run pipelines on historical data after changing logic. Also, sub-pipelines can be rerun after bug fixes, maximizing efficiency.

Airflow's user interface provides both a detailed view of pipelines and individual tasks, as well as an overview of pipelines over time. The user interface allows you to view logs and manage tasks. B. Repeat the task if an error occurs.

Airflow's open-source nature ensures that we work on components that are developed, tested, and used by many other companies around the world. You'll find many helpful resources in the form of blog posts, articles, conferences, books, and more in our active community. Connect with other colleagues through various channels like Slack and mailing lists.

DAGs In Airflow, a DAG (Directed Acyclic Graph) is a collection of all tasks to execute, organized to reflect their relationships and dependencies.

A DAG is defined in a Python script that represents the DAG structure (tasks and their dependencies) as code.

For example, a simple DAG might consist of three tasks: A, B, and C. You might say that A must run successfully before B runs, but C can always run. We can say that task A expires after 5 minutes and B can be restarted up to 5 times if it fails. It could also mean that the workflow runs every night at 10:00 PM, but is not scheduled to start until a certain date.

This is how a DAG describes how a workflow should run. But notice that I didn't say anything about what I actually want to do. A, B, and C can be anything. Perhaps A is preparing data for her B to analyze while C is sending emails. Alternatively, A can monitor your location so that B can open the garage door and C can turn on the house lights. The point is that the DAG doesn't care what its composition tasks do. Their job is to ensure that everything they do happens at the right time, in the right order, or with the right handling of unexpected problems. DAGs are defined in standard Python files placed in Airflow's DAGS FOLDER. Airflow runs the code in each file to dynamically create DAG objects. You can have any number of DAGs, each describing any number of tasks. In general, each should correspond to a single logical workflow.

Scope Airflow loads all DAG objects that can be imported from a DAG file. Importantly, the DAG must be included in globals(). Consider the following two DAGs: Only dag 1 is loaded. The other is visible only in the local area.

Sometimes it comes in handy. For example, a common pattern for SubDag-Operator is to define a subdag inside a function so that Airflow doesn't try to load it as his standalone DAG.

Dataset Credit card fraud detection data set, from Kaggle (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud). This dataset contains credit card transactions from European cardholders for September 2013. This dataset represents transactions made within two days, with 492 frauds out of 284,807 transactions. The dataset is highly imbalanced, with the positive class (fraud) making up his 0.172 percent of all transactions.

This includes only numeric input variables that are the result of PCA transformations. Unfortunately, due to confidentiality reasons, we are unable to provide detailed background information on the original features and data. Features V1, V2, ... V28 are the main components obtained by PCA and the only features that are not transformed by PCA are 'Time' and 'Magnitude'. The Time property contains the elapsed seconds between each transaction and the first transaction in the record. The property 'Amount' is the transaction amount. This property can be used, for example, for cost-aware learning. The characteristic "class" is the response variable and takes the value 1 if it is incorrect and 0 otherwise.

Given the class imbalance rate, we recommend using the area under the precision recall curve (AUPRC) to measure precision. Confusion matrix accuracy is meaningless for imbalanced classification

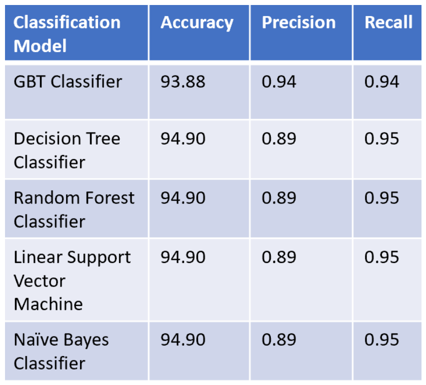
Classification Models Created Different classsification models have been used to find the best classifier for the given data.

- Gradient-boosted tree classifier (GBT)

– Decision tree classifier
– Random forest classifier
– Linear support vector machine
– Naïve bayes classifier

Metrics to compare ML models

– Accuracy: Percentage of results correctly classified
– Precision: percentage of the results which are relevant
– Recall: percentage of total relevant results correctly classified



Classification Model	Accuracy	Precision	Recall
GBT Classifier	93.88	0.94	0.94
Decision Tree Classifier	94.90	0.89	0.95
Random Forest Classifier	94.90	0.89	0.95
Linear Support Vector Machine	94.90	0.89	0.95
Naïve Bayes Classifier	94.90	0.89	0.95

Fig. 6. Comparison of Different ML Models

Classification Model Classification model is created in pyspark, which will be later used to predict if a transaction is fraud or not. From the new data, we spilt the data further for training and testing. The train/test split taken is 80/20. Random forest classifier is used to create the model.

Apache WORKFLOW CHARACTERISTICS Dynamic: Airflow pipelines are configured as Python code, allowing for dynamic pipeline generation. Extensible: The Airflow framework contains operators to connect with numerous technologies. All Airflow components are extensible to easily adjust to your environment. Flexible: Workflow parameterization is built-in.

If the workflows have a clear start and end, and run at regular intervals, they can be programmed as an Airflow DAG.

Airflow DAG DAG or a Directed Acyclic Graph – is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies. DAG is a graph with nodes, directed edges and no cycles. A DAG is defined in a Python script, which represents the DAGs structure (tasks and their dependencies) as code.

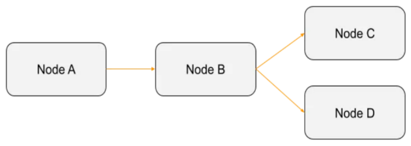


Fig. 7. Valid DAG Structure

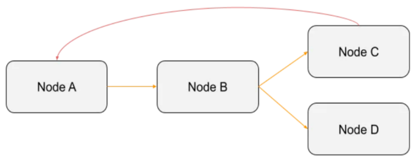


Fig. 8. Invalid DAG Structure

DAG RUN DAG run is a physical instance of a DAG, containing task instances that run for a specific execution date. A DAG run is usually created by the Airflow scheduler but can also be created by an external trigger. Multiple DAG runs may be running at once for a particular DAG, each of them having a different execution date.

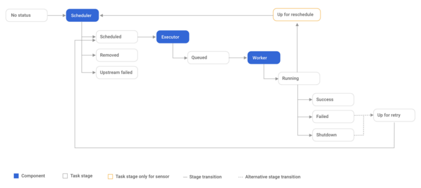


Fig. 9. Task Lifecycle

TASKS A Task defines a unit of work within a DAG. It is represented as a node in the DAG graph, and it is written in python. Each task is an implementation of an Operator, for example, a PythonOperator to execute some python code, or a BashOperator to run a bash command.

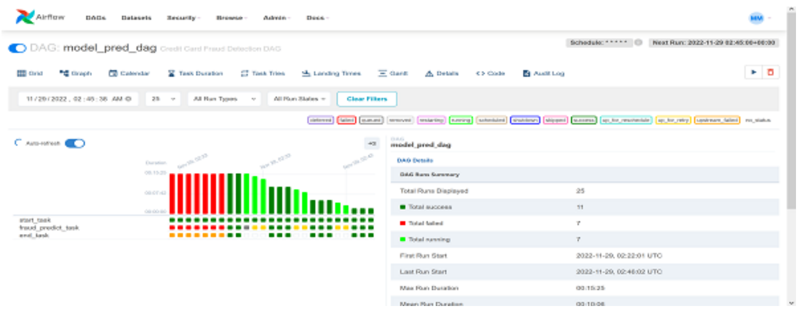
TASK LIFECYCLE The happy flow consists of the following stages:

– No status (scheduler created empty task instance)
– Scheduled (scheduler determined task instance needs to run)
– Queued (scheduler sent task to executor to run on the queue)
– Running (worker picked up a task and is now running it)
– Success (task completed)

Operators While DAGs describe how to run a workflow, Operators determine what gets done by a task. An operator describes a single task in a workflow. Operators are usually (but not always) atomic, meaning they can stand on their own and don't need to share resources with any other operators. If two operators need to share information, like a filename or a small amount of data, you should consider combining them into a single operator.

Airflow provides operators for many common tasks, including:

– BashOperator: executes a bash command
– PythonOperator: calls an arbitrary python function
– EmailOperator: sends an email
– SimpleHttpOperator: sends an HTTP request
– MySqlOperator: executes a SQL command
– Sensor: an Operator that waits (polls) for a certain time, file, database row, S3 key, etc..



Schedule: * * * * *
Next Run: 2022-11-29 02:45:00+00:00

model_pred_dag
DAG Details

DAG Runs Summary	
Total Runs Displayed	25
🟩 Total success	11
🟥 Total failed	7
🟩 Total running	7
First Run Start	2022-11-29, 02:22:01 UTC
Last Run Start	2022-11-29, 02:46:02 UTC
Max Run Duration	00:15:25
Mean Run Duration	00:10:06

Fig. 10. Airflow Credit Card Fraud Detection DAG

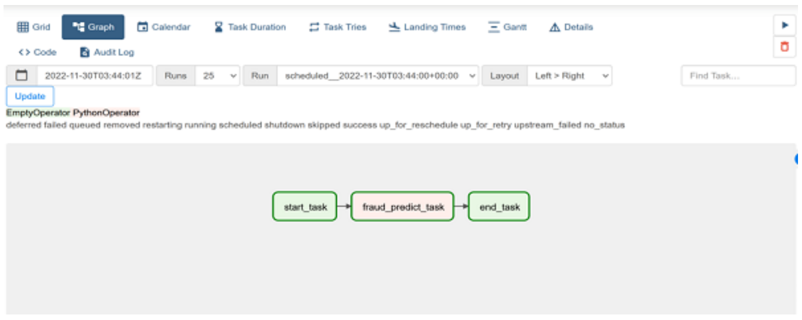
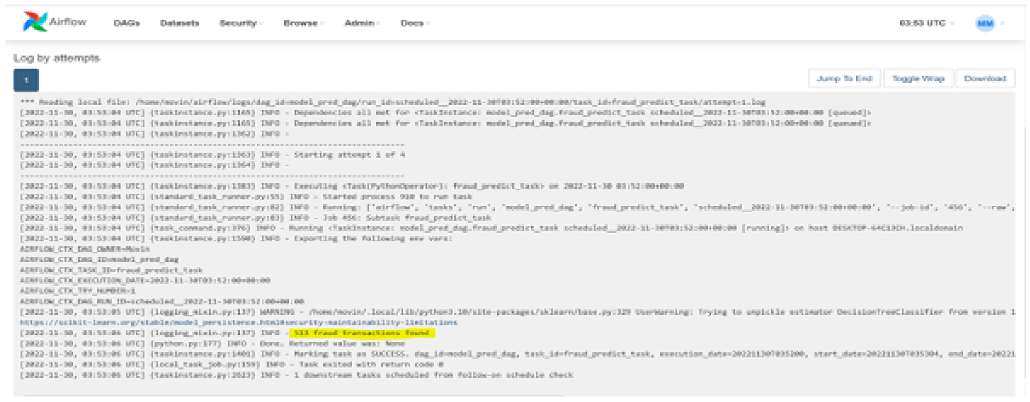

Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
2022-11-30T03:44:01Z
Runs
25
Run
scheduled__2022-11-30T03:44:00+00:00
Layout
Left > Right
Find Task...
Update
EmptyOperator PythonOperator
deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status
start_task
fraud_predict_task
end_task

Fig. 11. Airflow Graph



```
*** Reading local file: /home/movin/airflow/logs/dag_id=model_pred_dag/run_id=scheduled__2022-11-30T03:52:00+00:00/task_id=fraud_predict_task/attempt=1.log
[2022-11-30, 03:53:04 UTC] {taskinstance.py:1165} INFO - Dependencies all met for <TaskInstance: model_pred_dag.fraud_predict_task scheduled__2022-11-30T03:52:00+00:00 [queued]>
[2022-11-30, 03:53:04 UTC] {taskinstance.py:1165} INFO - Dependencies all met for <TaskInstance: model_pred_dag.fraud_predict_task scheduled__2022-11-30T03:52:00+00:00 [queued]>
[2022-11-30, 03:53:04 UTC] {taskinstance.py:1362} INFO -
--------------------------------------------------------------------------------
[2022-11-30, 03:53:04 UTC] {taskinstance.py:1363} INFO - Starting attempt 1 of 4
[2022-11-30, 03:53:04 UTC] {taskinstance.py:1364} INFO -
--------------------------------------------------------------------------------
[2022-11-30, 03:53:04 UTC] {taskinstance.py:1383} INFO - Executing <Task(PythonOperator): fraud_predict_task> on 2022-11-30 03:52:00+00:00
[2022-11-30, 03:53:04 UTC] {standard_task_runner.py:55} INFO - Started process 910 to run task
[2022-11-30, 03:53:04 UTC] {standard_task_runner.py:82} INFO - Running: ['airflow', 'tasks', 'run', 'model_pred_dag', 'fraud_predict_task', 'scheduled__2022-11-30T03:52:00+00:00', '--job-id', '456', '--raw',
[2022-11-30, 03:53:04 UTC] {standard_task_runner.py:83} INFO - Job 456: Subtask fraud_predict_task
[2022-11-30, 03:53:04 UTC] {task_command.py:376} INFO - Running <TaskInstance: model_pred_dag.fraud_predict_task scheduled__2022-11-30T03:52:00+00:00 [running]> on host DESKTOP-64C13CH.localdomain
[2022-11-30, 03:53:04 UTC] {taskinstance.py:1590} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=Movin
AIRFLOW_CTX_DAG_ID=model_pred_dag
AIRFLOW_CTX_TASK_ID=fraud_predict_task
AIRFLOW_CTX_EXECUTION_DATE=2022-11-30T03:52:00+00:00
AIRFLOW_CTX_TRY_NUMBER=1
AIRFLOW_CTX_DAG_RUN_ID=scheduled__2022-11-30T03:52:00+00:00
[2022-11-30, 03:53:05 UTC] {logging_mixin.py:137} WARNING - /home/movin/.local/lib/python3.10/site-packages/sklearn/base.py:329 UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
[2022-11-30, 03:53:06 UTC] {logging_mixin.py:137} INFO - 513 fraud transactions found
[2022-11-30, 03:53:06 UTC] {python.py:177} INFO - Done. Returned value was: None
[2022-11-30, 03:53:06 UTC] {taskinstance.py:1401} INFO - Marking task as SUCCESS. dag_id=model_pred_dag, task_id=fraud_predict_task, execution_date=20221130T035200, start_date=20221130T035304, end_date=20221
[2022-11-30, 03:53:06 UTC] {local_task_job.py:159} INFO - Task exited with return code 0
[2022-11-30, 03:53:06 UTC] {taskinstance.py:2623} INFO - 1 downstream tasks scheduled from follow-on schedule check
```

Fig. 12. Airflow Output Log

IV RESULTS/DISCUSSION

The Python ML code prints out the number of credit card frauds detected in the given data set where class 0 means the transaction was determined to be valid and 1 means it was determined as a fraud transaction. This is used to calculate the accuracy score and precision of the algorithms. The results along with the classification report for the random forest algorithm are given in the output as follows. This result matched against the class values to check 284,807 transactions and 513 fraud transactions found.

V CONCLUSION

In conclusion, our analysis of various classifiers has provided valuable insights into their performance. With the exception of the Gradient Boosting Tree (GBT) classifier, all other classifiers yielded nearly identical results, which could be attributed to the relatively limited size of our dataset.

Remarkably, the GBT classifier demonstrated a significantly higher precision compared to the other classifiers, suggesting its ability to accurately identify true positive cases. However, when examining accuracy and recall, the GBT classifier performed similarly to the other models.

Considering these findings, one might argue that, based on the data generated, the GBT classifier could be considered the optimal choice, particularly if precision is of utmost importance. However, it's essential to note that we ultimately opted for the Random Forest classifier for our specific dataset. This choice may have been influenced by several factors, including computational efficiency, model interpretability, or project-specific requirements.

In summary, while the GBT classifier excels in precision, the Random Forest classifier was selected for our dataset due to various practical considerations. The choice of classifier should be made with a holistic view of project objectives and constraints, ensuring that it aligns with the broader context and goals of the analysis.

A REFERENCES

1. Herbst-Murphy, Susan. (October 2013). Clearing and Settlement of Interbank Card Transactions: A MasterCard Tutorial for Federal Reserve Payments Analysts. Federal Reserve Bank of Philadelphia. Retrieved from www.philadelphiafed.org/payment-cards-center/
2. Martin, Niels, Swennen, Marijke, Depaire, Benoît, Jans, Mieke, Caris, An, & Vanhoof, Koen. (n.d.). Batch Processing: Definition and Event Log Identification. Hasselt University, Agoralaan Building D, 3590 Diepenbeek, Belgium.
3. Pufahl, Luise, & Weske, Mathias. (2015). Batch Processing across Multiple Business Processes based on Object Life Cycle (Extended Abstract). ResearchGate. DOI: 10.13140/RG.2.1.4335.9120
4. Poudel, Samir, Paudyal, Rajendra, Cankaya, Burak, Sterlingsdottir, Naomi, Murphy, Marissa, Pandey, Shital, Vargas, Jorge, & Poudel, Khem. (2023). Cryptocurrency Price and Volatility Predictions with Machine Learning. *Journal of Marketing Analytics*. DOI: 10.1057/s41270-023-00239-1
5. Oluwasakin, Ebenezer, Thomas, T., Tingting, S., Yinusa, A., Poudel, S., Hasan, N., Vargas, J., & Poudel, Khem. (2023). Minimization of High Computational Cost in Data Preprocessing and Modeling using MPI4Py. *Machine Learning with Applications*, *13*, 100483. DOI: 10.1016/j.mlwa.2023.100483
6. Bandi, Raswitha, Jayavel, Amudhavel, & Karthik, R. (2018). Machine Learning with PySpark - Review. *Indonesian Journal of Electrical Engineering and Computer Science*, *12*(1), 102-106. DOI: 10.11591/ijeecs.v12.i1.pp102-106
7. Hasan, Md, Hamdan, Sammi, Poudel, Samir, Vargas, Jorge, & Poudel, Khem. (2023). Prediction of Length-of-Stay at Intensive Care Unit (ICU) Using Machine Learning based on MIMIC-III Database. In *Proceedings of the [Conference Name]*, pp. 321-323. DOI: 10.1109/CAI54212.2023.00142
8. Manem, Chaitanya, Arya, Prashant, Shekhar, Himanshu, & Acheadeth, Lay. (2023). Imbalance multi-label Classification in Pyspark.

9. Shukla, Sameer. (2022). Creating Data Pipelines using Apache Airflow. *Zenodo.* https://doi.org/10.5281/zenodo.6828344
10. Poudel, K., Uddin, M., Kommu, R., Muhammed, S., Hasan, N., and Hamdan, S. (2023). HealthCare Text Analytics Using Recent ML Techniques. In: Daimi, K., Al Sadoon, A. (eds) *Proceedings of the 2023 International Conference on Advances in Computing Research (ACR'23)*, ACR 2023. Lecture Notes in Networks and Systems, vol 700. Springer, Cham. DOI: https://doi.org/10.1007/978-3-031-33743-7_11