

**Professor Handan Liu**

**Credit Card Fraud Detection using Parallel and Distributed Computing techniques**

**High Performance Parallel Machine Learning and AI**



**Team 05**

**Ronak Patil - 001007035**

**Nikita Gaurihar - 002980962**

**Credit Card Fraud Detection using Parallel and Distributed Computing techniques**

## **Introduction**

## **Background**:

With the advent of technology, Machine Learning and Artificial Intelligence applications are expanding their reach in most of the scientific research study fields. The implementation of Machine Learning and Deep Learning applications can be encountered with not only Healthcare, Human Resources, Marketing, Gaming, but the Finance industry is also maintaining its pace in this competition. From blockchain to real-time stock exchange data, relies mainly on technology. Data fidelity, consistency and Data lineage forms an important aspect when it comes to any machine learning problem focusing on forecasting.

Although, machine learning resolves the forecasting and future prediction from business perspectives, but the cost of implementing the model and computing on such a real-time live data is high. Apart from this, even processing a few GiBs of data on our local machine is an uphill task unless the laptop has a lot of computational power and substantial RAM. A few professionals like Data Scientists end up buying GPU machines and computation power on cloud for their research.

In today’s era, where digitalized banking is promoted, Banks need to keep track of legitimate cashflows for each account. The most fascinating use of Machine learning along with Parallel computing in Finance Industry was when an algorithm was developed to predict the future stock price. Dealing with such a million amounts of real-time data needs high computational powers which our local machines with single core unit cannot handle resulting in the system meltdown due to heat generated due to processing. This is where the use of Parallel and Distributed computing techniques comes into picture.

This project focusses on identifying fraudulent transactions of Credit cards used by customers. With the help of the Dask concept which is a Distributed Data Parallel library in Python for multi-threaded and multi-machine model training, machine learning algorithms are developed on multiple CPUs and compared with traditional Machine Learning models for an Empirical study purpose. The models will then be evaluated based on high performance evaluation metrics.

1. **Motivation:**

With digitalization, an exponential increase in the use of credit and debit cards can be observed in the past decade. Nowadays, the use of mobile apps for day-to-day transactions is preferred over the traditional methods. Even though electronic transactions have several security flaws, many customers choose to use them because of how convenient they are and do not hesitate to make even high-value transactions. This project will give an opportunity to Banks to identify frauds and fraudsters in a cost-effective way, thus maintaining their customer relations. However, since the customer data is real-time, handling such a bigdata on basic local machines is a challenging task. This study focuses on implementing machine learning through parallel computing techniques using an open-source scalable library named dask to solve this problem for the Banking Industry.

1. **Goal:**

The goal of this project is to identify the anomalies in the Credit card transactional data and justify the fraudulent transactions using Machine learning algorithm by spending less time and money. The cost-effective way to solve this issue is through distributed parallel computing which enables us to use our local machines over those expensive cloud storages and GPUs. This will not only help banks to process real-time data in just nanoseconds, thereby highlighting the red flags, but will also help in taking actions for the frauds that were identified before it is too late for the customers.

## **Methodology**

This project focuses on predicting fraudulent transactions with the help of parallel computing techniques. We are using Dask Dataframe (using scikit learn and pandas packages) and Dask Machine learning methods to train the model. The methodology to achieve this objective is mentioned in depth as follows:

1. **Data Preprocessing:**
2. **Exploratory Data Analysis:**

As a part of data preprocessing, Exploratory data analysis and data cleaning were incorporated to find out if there are any null values. The detailed information is as follows –

* **Dataframe Dimensions:**

We observed that our dataset has 23 attributes with a total of 1852394 records. To find the dimensions of our dataset, following commands were used –

Graphical user interface, text, application

Description automatically generated

* **Checking if there are any null values –**

A picture containing chart

Description automatically generated

There were no null values in our available dataset. To find if there were any nulls in our dataset, following commands were used –

* **Treating Categorical Variables:**

#### Since columns -'category' and 'gender' are categorical in nature, we need to encode the data in the numerical form. This can be achieved by creating ‘Label Encoders’.

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

* **Feature Engineering**

A few columns such as Age, Hour, day, month, year from Date of Birth and Transaction\_date respectively was derived and replaced with their original columns. Following commands explains the derivation –

A picture containing text

Description automatically generated

* **Normalization**

Once the data was cleaned, normalization was performed on the data to maintain uniformity across the dataset.

1. **Machine Learning Model Training**

* To predict fraudulent transactions, following Machine Learning algorithms were used to train the models using the available data –

1. Random Forest Classification Model

2. XGBoost Classification Model

3. Light Gradient Descent Classification Model

* The plan is to divide the dataset for training and testing in the 70:30 ratio respectively.
* To start with, we ran base models for the above 3 Machine learning classification models.

**Hyperparameter Tuning:**

The Random Search will be used to find the model's ideal parameters as part of the hyperparameter tuning process. This will help us find the right features for the best performing model. Further we compared the accuracies of these models and justified the best performing model.

1. **Random Forest Classifier:**

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

After Tuning the model, the best parameters found are –

{'n\_estimators': 100, 'min\_samples\_split': 1.0, 'min\_samples\_leaf': 8, 'max\_features': 'auto', 'max\_depth': 450, 'criterion': 'entropy'}

1. **XGBoost Model:**

**Graphical user interface, text, application

Description automatically generated**

After Tuning the model, the best parameters found are –

{'reg\_lambda': 1, 'reg\_alpha': 1, 'n\_estimators': 100, 'learning\_rate': 0.05, 'gamma': 1, 'booster': 'gblinear', 'base\_score': 0.5}

**Light Gradient Descent Boost Model :**

**Graphical user interface, text, application, email

Description automatically generated**

After Tuning the model, the best parameters found are –

{'reg\_alpha': 0.01, 'num\_leaves': 80, 'min\_child\_samples': 15, 'max\_depth': 10, 'learning\_rate': 0.2}

1. **Performance Analysis using Parallel computing:**

* First, the model will run on a single machine without using the Dask Library. We will calculate the CPU time and wall time to gauge how much time it takes to process the data in a single Dataframe. Then, using the Dask library in python, Dask cluster was implemented to enable efficient parallel computation on our single machine.
* The model was then trained across multiple CPUs using the Dask library's Dask cluster and workers to keep them in sync, thus speeding up the process. A Dask distribution function will be used to incorporate the interaction between the scheduler and the workers. This is how the scheduler interacts with the client to make sure the sync is in parallel.
* Implementation of Dask ML on the above Machine learning models for classification was carried out to process the data in parallelly.
* The Dask visualization for the models were also generated.

Diagram

Description automatically generated

* Three components make up a Dask network:
* A centralized scheduler that oversees workers and assigns tasks for them to perform
* Numerous workers do the computation, store results, and exchange results with other workers.
* One or more clients, via which users communicate with Jupyter notebooks or scripts, and provide tasks to the scheduler for execution by the workers

## **Description of dataset**

## To implement this into practice, we are using the following Kaggle Dataset –

<https://www.kaggle.com/datasets/kartik2112/fraud-detection?select=fraudTrain.csv>

It is a classic example of classification problems. The in-depth information about the data is mentioned below:

* **index** – This column acts as a unique identifier for each row
* **transaction\_time** – This column detailed information about the transaction date and time
* **cc\_num** – This column describes about the Credit Card Number of Customer
* **merchant** – This column gives the detail about the Merchant Name
* **category** – This column describes about the merchant category
* **amt** – This column gives detailed information about the amount of transaction
* **first** – This column describes the first name of the credit card holder
* **last** – This column describes the last name of the credit card holder
* **gender** – This column describes the gender of the credit card holder
* **street** - This column describes the street address of the credit card holder
* **city** - This column describes the city of the credit card holder
* **state** - This column describes the state of the credit card holder
* **zip** - This column describes the zip of the credit card holder
* **lat** - This column describes the latitude of the credit card holder
* **long** - This column describes the longitude of the credit card holder
* **city\_pop** - This column gives detailed information about the credit card holder’s city population
* **job** – This column gives detailed information about the job of the credit card holder
* **dob** - This column describes the date of birth of the credit card holder
* **trans\_num** – This column gives detail about the transaction number
* **unix\_time** - This column gives detail about the unix time of transaction
* **is\_fraud** – This column acts as a Fraud Flag

## **Results and Analysis**

For model training and predictions through multiple processors, we have used Discovery cluster with the following specifications –

1. Memory – 32GB
2. CPUs – 8

Graphical user interface, text, application

Description automatically generated

The in-detail observations of our project is divided in 4 parts, as follows –

Part 1: Traditional Machine Learning Models Analysis

Part 2: Parallel Computing using Dask

Part 3: Evaluation Comparison

Part 4 : Dask Dashboard

**Part 1: Traditional Machine Learning Models Analysis**

1. **Random Forest Classifier**

**Chart

Description automatically generated**

Graphical user interface, text, application

Description automatically generated

**Interpretation:**

* As we can observe from the confusion matrix, the True positive is 552827 and True negatives are 5.
* Since our model accuracy is 99.48%, the predictions are justifiable.

1. **XGBoost Classifier**

**Table

Description automatically generated**

**Chart

Description automatically generated**

**Text

Description automatically generated**

**Interpretation:**

* As we can observe from the confusion matrix, the True positive is 552668 and True negatives are 1788.
* Since our model accuracy is 99.77%, the predictions are justifiable.

1. **Light Gradient Descent Classifier:**

**Chart

Description automatically generated**

**Text

Description automatically generated**

**Interpretation:**

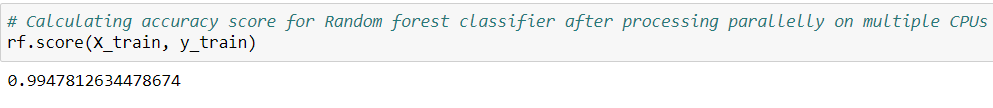
* As we can observe from the confusion matrix, the True positive is 552583 and True negatives are 2052.
* Since our model accuracy is 99.48%, the predictions are justifiable.

**Part 2: Parallel Computing using Dask**

1. **Random Forest Classifier**

**Chart

Description automatically generated with medium confidence**

****

**Interpretation:**

* As we can observe from the confusion matrix, the True positive is 552850 and True negatives are 3.
* Our model accuracy is 99.47%
* If we compare it with the base model, the model accuracy has reduced by 0.01%.
* Hence, it doesn’t guarantee that faster performance will give you more accurate results.

1. **XGBoost Classifier**

**Chart

Description automatically generated**

**Graphical user interface, text

Description automatically generated with medium confidence**

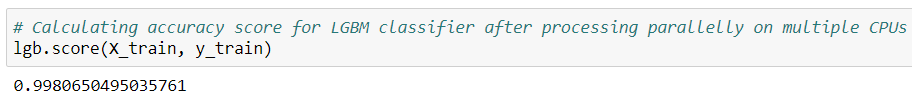
**Interpretation:**

* As we can observe from the confusion matrix, the True positive is 552703 and True negatives are 1850.
* Our model accuracy is 99.78%
* If we compare it with the base model, the model accuracy has increased by 0.01%.
* Hence, faster performance are giving us more accurate results.

1. **Light Gradient Descent Classifier**

**Chart

Description automatically generated**

****

**Interpretation:**

* As we can observe from the confusion matrix, the True positive is 552613 and True negatives are 2023.
* Our model accuracy is 99.80%
* If we compare it with the base model, the model accuracy is almost same
* Hence, data parallelism is giving same accuracy with more time.

**Part 3: Evaluation Comparison**

Following is the comparison of the performance of the traditional Machine learning algorithms processed on 1 CPU over Machine learning models computed through parallel processing techniques over multiple CPUs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Random Forest Classifier** | | | | |
| **Serial Execution Time T(1) with one processing unit = 92.53 sec** | | | | |
|  | **1 CPU** | **2CPU** | **4CPU** | **8CPU** |
| **Execution Time T(p) with 'p' no.of processing units** | 44.92 | 37.07 | 35.98 | 36.57 |
| **Speedup S(p) for 'p' no.of processing units** | 2.059884239 | 2.496088481 | 2.571706504 | 2.530216024 |
| **Efficiency** | 2.059884239 | 1.248044241 | 0.642926626 | 0.316277003 |

Graphical user interface, text, application

Description automatically generated

**Interpretation:**

* Time Taken by traditional machine learning Random Forest Classifier Model to train the data is 92.53 seconds
* However, when distributed the data and ran the same model on multiple processors using Dask, we can see the training time reduced until CPU = 4 but it increased as we increased the number of CPUs further.
* Hence, total CPUs are needed to improve processing time and improve computational accuracy by reduction in costs is CPU = 4

**Line chart

Description automatically generated with medium confidence**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **XGBoost Classifier** | | | | |
| **Serial Execution Time T(1) with one processing unit = 99.72 sec** | | | | |
|  | **1 CPU** | **2CPU** | **4CPU** | **8CPU** |
| **Execution Time T(p) with 'p' no.of processing units** | 91.97 | 79.21 | 78.72 | 78.48 |
| **Speedup S(p) for 'p' no.of processing units** | 1.084266609 | 1.258931953 | 1.266768293 | 1.270642202 |
| **Efficiency** | 1.084266609 | 0.629465977 | 0.316692073 | 0.158830275 |

Graphical user interface, application

Description automatically generated

**Interpretation:**

* Time Taken by traditional machine learning XGBoost Model to train the data is 99.72 seconds
* However, when distributed the data and ran the same model on multiple processors using Dask, we can see there is a lot of difference in the training time to justify that multiple CPUs are needed to improve processing time and improve computational accuracy by reduction in costs.
* Also, we can see, as we increase the number of CPUs, the time required to process as reduced.
* Hence, the optimum number of CPUs required to process the available data parallel using Dask is 8.

Chart

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Light Gradient Boost Classfier** | | | | |
| **Serial Execution Time T(1) with one processing unit = 12.15 Sec** | | | | |
|  | **1 CPU** | **2CPU** | **4CPU** | **8CPU** |
| **Execution Time T(p) with 'p' no.of processing units** | 13.0068 | 13.58 | 13.47 | 13.63 |
| **Speedup S(p) for 'p' no.of processing units** | 0.934126764 | 0.894698085 | 0.902004454 | 0.891415994 |
| **Efficiency** | 0.934126764 | 0.447349043 | 0.225501114 | 0.111426999 |

**Base Model Time to process:**

Graphical user interface

Description automatically generated with medium confidence

**Interpretation:**

* Time Taken by traditional machine learning LGBM Model to train the data is 12.15 seconds
* However, when distributed the data and ran the same model on multiple processors using Dask, we can see there is not much difference in the training time to justify that multiple CPUs are needed to improve processing time and improve computational accuracy by reduction in costs.

**LGBM Model on multiple CPUs:**

Chart, line chart

Description automatically generated

**Part 4: Dask Dashboard**

1. **Random Forest Classifier**

**Dask Status**

**Graphical user interface, application

Description automatically generated**

**Dask Graph**

**Graphical user interface, text

Description automatically generated with medium confidence**

1. **XGBoost Classifier**

**Dask Status**

Graphical user interface

Description automatically generated

**Dask Graph**

A screenshot of a computer

Description automatically generated with medium confidence

1. **Light Gradient Descent Classifier**

**Dask Status**

**Graphical user interface, application

Description automatically generated**

**Dask Graph**

**Graphical user interface, text, email

Description automatically generated with medium confidence**

## **Conclusion**

## From the model evaluation metrics, we can conclude that the XGBoost model is the best performing model as its accuracy improved by 0.01% as we trained the model on multiple CPUs. The optimum number of CPUs required to process is 4.

## However, Random Forest Classifier showed better performance with efficiency of 2.05 on CPU =1 even when processed parallelly

## Light Gradient Descent Boost Model ran fastest among the other two. However, the efficiency and speedup are less as compared to the other two models

## Utilizing additional CPUs intermittently does not increase performance. There are instances where we receive unexpected outcomes for a parallel computation using Dask.

## **References**

[1] *Dask — Dask documentation*. (n.d.). <https://docs.dask.org/en/stable/>

[2] Somasundaram, A. (2018, July 17). Parallel and incremental credit card fraud detection model to handle concept drift and data imbalance. SpringerLink. <https://link.springer.com/article/10.1007/s00521-018-3633-8?error=cookies_not_supported&code=024a9709-9587-4678-8356-84e957f12113>

[3] Singh, A. (2022, July 21). Ultimate guide to handle Big Datasets for Machine Learning using Dask (in Python). Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2018/08/dask-big-datasets-machine_learning-python/>

[4] Optimizing Fraud Detection in Financial Services with Graph Neural Networks and NVIDIA GPUs. (2022b, October 20). NVIDIA Technical Blog. <https://developer.nvidia.com/blog/optimizing-fraud-detection-in-financial-services-with-graph-neural-networks-and-nvidia-gpus/>

[5] Sahni, Sartaj & Thanvantri, Venkat. (2002). Parallel Computing: Performance Metrics and Models.

[6] Mwiti, D. (2022, August 11). *Machine Learning in Dask - Heartbeat*. Medium