National University
Of Computer and Emerging Sciences

Parallel and Distributed Computing

Project Report

Name: Umaima Hashmi, Nooran Ishtiaq, Zoha Binte Wajahat
Roll No: 22I-1894 (DS-B), 22I-2010 (DS-B), 22I-0569 (AI-C)

Submitted To:
Sir Mateen Yaqoob

Due Date:
5th May, 2025

1. Introduction:

The goal of this project was to construct an optimized machine learning pipeline for binary classification based on a provided dataset. The optimization was to minimize processing time at the expense of or while not compromising model performance. We experimented with parallel computing through multithreading (n_jobs) as the optimization method.

2. Preprocessing Pipeline:

For providing good input to the model, the following were carried out:

a. Duplicate Handling

Deleted all duplicate rows via df.drop_duplicates().

b. Missing Values

Numeric columns: Imputed missing values based on column-wise mean.

c. Categorical Encoding

feature_5 (Yes/No) as binary (1/0).
feature_3 (A, B, C) as ordinal integers (0/1/2).

d. Log Transformation

Perfected log1p() transformation for decreasing skewness in:
feature_1, feature_4, feature_7

e. Outlier Removal

Applied Interquartile Range (IQR) technique to eliminate outliers from the said transformed features.

f. Scaling

Standardized the dataset using **StandardScaler** to ensure all features have a mean of 0 and standard deviation of 1. This normalization helps improve model performance and convergence, especially for algorithms sensitive to feature magnitudes. Scaling was applied **after** log transformation and outlier removal to maintain consistency and prevent distortion.

3. Model and Training

Model 1: Logistic Regression (Machine Learning)

- Implemented using scikit-learn
- Two modes:
    - **Serial** (default)
    - **Parallel** with n_jobs=16, using 'liblinear' solver

Model 2: Deep Learning Model (Multilayer Perceptron)

- Built using **TensorFlow (Keras API)**
- Architecture:
    - Input layer
    - 2 Dense hidden layers with ReLU activation
    - Output layer with Sigmoid activation
- **Epochs**: 20
- **Batch size**: 32

Model 3 : XGBoost (Extreme Gradient Boosting)

Implemented using the `xgboost` Python library with both **CPU** and **GPU (CUDA)** acceleration modes.

- **Training Strategy**:
    - Applied log1p() transformation to reduce skewness in features.
    - Removed outliers using the IQR method.
    - Standardized numerical features with StandardScaler.
    - Handled class imbalance using the scale_pos_weight parameter.
- **Training Parameters**:
    - objective: 'binary:logistic'

    - eval_metric: 'logloss'
    - max_depth: 6
    - learning_rate: 0.1
    - subsample: 0.8
    - colsample_bytree: 0.8
    - tree_method: 'hist'
    - device: 'cpu' and 'cuda' (for GPU)

Model 4: Random Forest (Machine Learning)

Implemented using **scikit-learn's RandomForestClassifier** with parallel processing enabled via n_jobs=-1.

Training Strategy:

- Loaded data using **Dask** for scalable preprocessing and computation.
- Applied log1p() transformation to reduce skewness in select features.
- Removed outliers using the **Interquartile Range (IQR)** method.
- Standardized numerical features using **StandardScaler**.
- Encoded categorical features manually using mapping.
- Addressed class imbalance using class_weight='balanced' during training.

Training Parameters:

- n_estimators: 100 (number of trees in the forest)
- class_weight: 'balanced' (to handle class imbalance)
- n_jobs: -1 (utilizes all available CPU cores for parallel training)
- random_state: 42 (for reproducibility)

4. Results:

Model	Accuracy	F1 Score	Speedup
Logistic Regression	1.0	1.0	40.539208832298314
MultiLayer Perceptron	0.603	0.027	35.536966255848476
XGBoost (Extreme Gradient Boosting)	Cpu : 0.5184 Gpu :0.5064	Cpu: 0.5064 Gpu :0.4337	89.81
RandomForestClassifier	Cpu: 0.5854531001589826 Gpu : 0.5776570368407103	Cpu: 0.21124275270985632 Gpu : 0.2093773257256264	57.8

Insights:

- XGBoost achieved **balanced F1 scores** near 0.45 on both CPU and GPU modes, indicating moderate performance with class imbalance handled.
- GPU-accelerated training reduced processing time by 89.81**%**, offering a nearly **10× speedup** over CPU.
- While GPU didn't improve accuracy, it drastically improved training efficiency, making it highly beneficial for large datasets or parameter tuning.

Additional results:

Model Accuracy

- **Achieved Accuracy**: **60%**

Execution Time Comparison

Configuration	Execution Time
CPU	25.75 sec
GPU	10.87 sec speedup achieved 57.78%
Dask (Parallel CPU)	4.44 sec
Dask + GPU (Hybrid)	5.5 sec

Observations

- GPU alone significantly outperforms the CPU in terms of speed.
- Dask (parallel processing on CPU) provides the fastest result at **4.44 seconds**.
- Surprisingly, the hybrid **Dask + GPU** setup is slightly slower than using Dask alone, possibly due to overhead in coordination between parallel CPU and GPU processing.
- Accuracy remains constant regardless of the preprocessing method used, indicating that optimization primarily affects speed, not model output.

Random Forest Using Spark :

- Non-Spark implementation runs significantly faster (5x speedup), completing in just 5.9 seconds, compared to 28.45 seconds using PySpark.

- PySpark's balanced random forest delivers more balanced F1 scores between classes due to explicit class balancing via undersampling.

- The non-Spark model has higher overall accuracy, but it is biased toward the majority class (poor recall/F1 for class 1).

- Spark's pipeline offers scalability and distributed computation, better suited for large-scale or multi-node environments.

- For small to medium datasets, standard CPU pipelines (e.g., scikit-learn or TensorFlow) may be more efficient and easier to manage.