Lung Cancer Classification with Dask and XGBoost

PROGRAMMING 6

Romana Mahjabin Eshita

1 Introduction

Dask is recognized for its ability to manage large datasets by leveraging parallel processing [4]. It employs lazy loading, meaning it only loads data when needed, which enhances its efficiency, especially in memory-limited environments. In our project, we built a machine learning pipeline for a lung cancer research project to classify different tumor types using gene expression and clinical data. Since the dataset is large, we used Dask to process it more efficiently. We also trained a predictive model using XGBoost with Dask. The pipeline includes a step to check how well the system performs and how scalable it is. One of our main goals was to compare how Dask performs against the more traditional Pandas and NumPy tools when working with big datasets and training machine learning models. Even though Dask is designed to handle large datasets through parallel processing, in our experiments, Pandas actually performed better. We found that the dataset wasn't big enough to benefit from Dask's strengths, and the extra processing overhead from Dask actually slowed things down. Therefore, in the end, we concluded that for this specific task and dataset size, using Pandas was more efficient than Dask.

2 Design Choice

After a thorough evaluation, we made the following design choices in our experiments.

Dataset: We used two datasets in our experiments:

- **Clinical Data:** There are 89 samples of lung cancer patients' data in CSV format, including 16 clinical variables such as tumor source location and tumor stage.
- **Gene Expression Data:** It is a matrix format data containing normalized expression levels of 66,000 genes for 89 samples.

Hardware Setup: For our experiments, we have used this hardware setup.

- **CPU Model:** Intel(R) Xeon(R) Gold 6248 @ 2.50GHz
- **CPU Count:** 80
- **RAM:** 880.35 GB

Feature Processing: Before performing feature selection, we carried out some data cleaning steps to prepare the dataset. First, we removed irrelevant columns, such as those that contained the same value in every row. Next, we replaced entries labeled "Not available" with `np.nan` to standardize missing data. Finally, we removed any columns with more than 35% `NaN` values, ensuring that only relevant and meaningful features were kept for analysis.

We used Cramér's V to help choose the most useful categorical features for our model. Cramér's V calculates how strongly two categorical columns are related, with scores between 0 (no connection) and 1 (strong connection). We calculated Cramér's V between each categorical feature from our clinical dataset and the target column, then kept the features that showed correlation higher than 0.2. We selected genes with high variance and a quantitative percentage greater than 95%, which allowed us to identify those that play a significant role in the analysis from the 66k gene set

Dask: For Dask implementation, we tried to follow the best practices; however, while working with Dask, we noticed that using the apply function was causing our code to run very slowly. This happened because applying adds extra overhead and is not as efficient for large datasets. To fix this, we switched to `map_partitions`, which applies the function directly to each chunk of the data. This made the pipeline run much faster, as `map_partitions` is better suited for Dask's way of processing data in parallel.

XGBoost: XGBoost is a gradient boosting algorithm known for its speed, accuracy, and efficiency in handling structured data for supervised learning tasks [1]. We used the XGBoost algorithm to build a predictive model for lung cancer classification[1]. The implementation was done using the XGBoost Python package, which provides efficient tools for training and evaluation. Additionally, XGBoost offers native integration with Dask, enabling scalable and parallelized training across multiple cores or distributed systems[2]. To leverage this capability, we utilized the Dask-compatible version of XGBoost to perform parallel processing and optimize computational performance during model training.

We used the following parameters for both the Pandas and Dask implementations of XGBoost: Max Depth: 5, Learning Rate: 0.1, and Objective: 'multi:softprob' (for multi-class classification).

Memory Profiling: We used the `memory_usage` function from the `memory_profiler`[3] package to monitor the memory consumption and computation time of our code. This tool allowed us to track how much memory was used during the execution of specific functions, helping us compare the efficiency of different implementations between Pandas and Dask.

3 Pandas vs Dask

In this section, we compared the computation time, memory consumption, and model performance of our pipeline between Pandas and Dask.

Computation Time: Overall, Pandas was faster than Dask in almost every step (Figure 1). For example, data processing with Dask took 100 times longer compared to Pandas. Model training times were similar, but data retrieval and model evaluation were slightly slower with Dask. This shows that for the size of our dataset, Dask's extra setup and overhead didn't help and actually made things slower, while Pandas handled everything more efficiently.

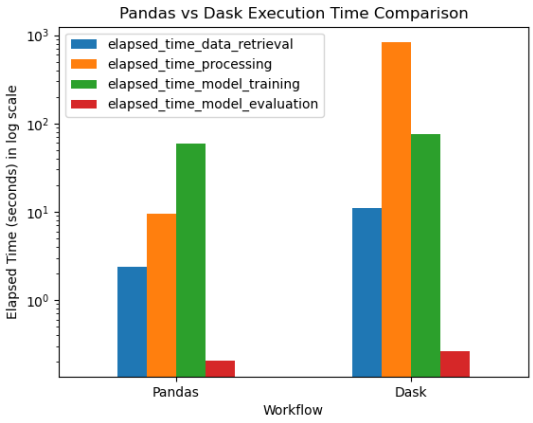

Figure 1: Computation Time comparison between Pandas and Dask workflows in log scale.

Memory Consumption: The Figure 2 compares the memory usage of Pandas and Dask during execution. It shows that Dask used more memory, around 530 MB, compared to Pandas, which used about 350 MB. Given the machine we used has almost 880.35 GB of memory, the workflow is perfect for pandas to fit in the memory without adding complexity, which Dask brings.

[1] https://xgboost.readthedocs.io/en/stable/python/index.html
[2] https://xgboost.readthedocs.io/en/stable/tutorials/dask.html
[3] https://pypi.org/project/memory-profiler/

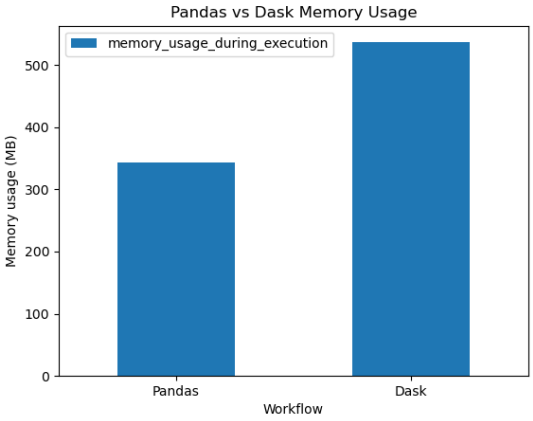

Figure 2: Memory consumption comparison between Pandas and Dask workflows.

Model Performance: This Figure 3 compares the model performance between Pandas and Dask across several metrics: accuracy, F1 score, precision, recall, and AUC. Overall, Pandas slightly outperformed Dask in most categories: Accuracy was higher for Pandas (around 0.88) compared to Dask (0.76). F1 Score, Precision, and Recall were all a bit better with Pandas, though the differences were modest. AUC (Area Under the Curve) was nearly the same for both, with Pandas having a slight edge. These results suggest that, along with being faster and more memory-efficient, Pandas also gave better model performance for this dataset in our workflow.

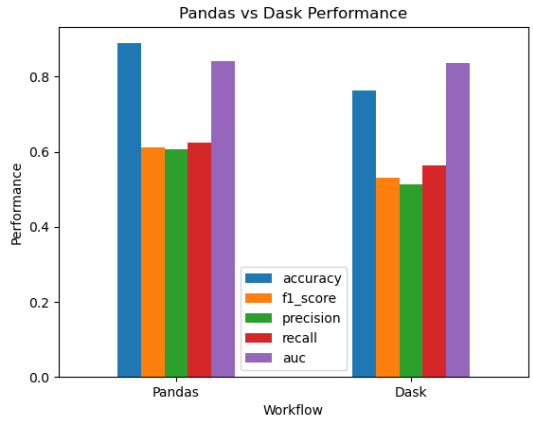

Figure 3: Model performance comparison between Pandas and Dask workflows.

4 Discussion

Pandas is one of the most widely used data processing libraries globally [2]. However, when working with large datasets and the need for parallel processing, it falls short. Unlike Dask, which is optimized for memory efficiency, Pandas loads the entire dataset into memory for processing and retains it there until the task is completed. Research shows that Pandas performs well and provides comprehensive data processing capabilities when the dataset size is small enough to fit into memory [3, 5]. In our case,

since the dataset can fit into memory, Pandas is well-suited for the task. On the other hand, Dask introduces additional overhead, which can slow down performance and does not provide significant advantages in this scenario[4].

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[2] Wes McKinney et al. Data structures for statistical computing in python. *SciPy*, 445(1):51–56, 2010.

[3] Angelo Mozzillo, Luca Zecchini, Luca Gagliardelli, Adeel Aslam, Sonia Bergamaschi, and Giovanni Simonini. Evaluation of dataframe libraries for data preparation on a single machine. *arXiv preprint arXiv:2312.11122*, 2023.

[4] Matthew Rocklin et al. Dask: Parallel computation with blocked algorithms and task scheduling. In *SciPy*, pages 126–132, 2015.

[5] Bhushan Pal Singh, Priyesh Kumar, Chiranmoy Bhattacharya, and S Sudarshan. Efficient dataframe systems: Lazy fat pandas on a diet. *arXiv preprint arXiv:2501.08207*, 2025.

[4] https://dask.pydata.org/en/stable/dataframe-best-practices.html