UPLIFT: Parallelization Strategies for Feature Transformations in Machine Learning Workloads

Liia Sharipova
University of Bamberg
Bamberg, Germany
liia.sharipova@stud.uni-bamberg.de

ABSTRACT

Data Preprocessing is an important step in the Machine Learning life-cycle, that improves the performance of the designed model. Feature Transformation, which transforms raw data into numerical matrices or tensors for training and learning, is a crucial part of Data Preprocessing. Existing ML systems provide optimal performance solutions for simple transformations but not complex, multi-pass transformation workflows with challenging data characteristics. In this paper, is introduces the UPLIFT framework, which focuses the choice of parallelization strategies depending on data characteristics. UPLIFT builds fine-grained task graph for transformation tasks, optimizes the execution plan and executes it in a cache-conscious manner. As a result of conducted experiments with the created FTBench benchmark with transformations and datasets, UPLIFT speedups the Feature Transformation process 9.27x on average compared to modern ML systems.

1 INTRODUCTION

Machine learning (ML) has found applications across various fields and industries, revolutionizing the way we approach problem-solving, data analysis, and decision-making including healthcare (Siddique and Chow, 2021), energy and economics (Ghoddusi et al., 2019), transportation and logistics (Tsolaki et al., 2022), education and other important domains. ML algorithms can handle various types of data, depending on the specific task and algorithm being used comprised of structured attributes, images, speech, video, graphs, and text(Molino et al., 2019). In this paper the overview is given for the article about parallelization Strategies for Feature Transformations in ML Workloads (Phani et al., 2022a) and created the UPLIFT framework. The theory and challenges of Feature Transformation in Section 1.1 are covered, provided the summary of the suggested approach in Section 2 with UPLIFT architecture introduction, interpreted the experiment results in Section 3, discussed the advantages and disadvantages of the approach, suggested future works in Section 4.

1.1 Common Feature Transformations

Feature transformations in ML encompass various techniques tailored for different types of data. Numerical transformations include normalization, binning (Bin), aggregation, and scaling. Categorical transformations involve recoding (RC), dummy-coding (DC) and feature hashing (FH). Modality-specific transformations are applied to text data using techniques like bag of words and word embeddings, while image data includes transformations such as cropping, rotating, and contrast adjustment. Table 1 illustrates main Feature Transformation types with build input and output types.

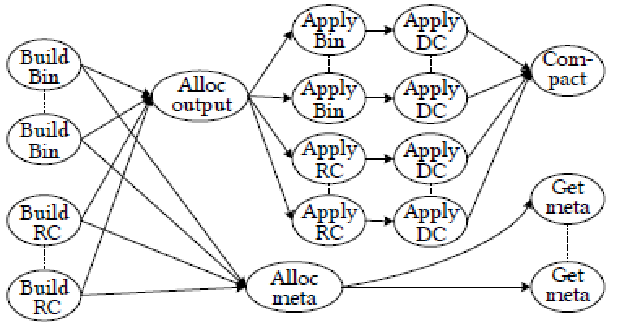

Figure 1: Task Graph for Adult Dataset (Phani et al., 2022a)

Binning (Bin) converts continuous or numerical data into categorical, e.g. age range 18-30 transforming into the category *Young*. Equi-width binning (BinW) - equal range. Equi-height binning (BinH) - equal amount.
Recoding (RC) modifying the values of a variable to create a new representation that better aligns with the requirements, e.g. transforming age *Young* into 1, age *Middle-aged* into 2.
Dummy-coding (DC) represent categorical variables as binary or "dummy" variables in ML, e.g. transformation of Color into several binary fields: *IsRed, IsBlue, IsGreen*.
Feature Hashing (FH) applies a hash function to each feature, which maps the original feature values to a fixed number of hash buckets or indices.

1.2 Challenges of Feature Transformations

Feature transformation in ML presents several challenges, including a large number of output columns, high cardinality, sparsity and cardinality skew, expensive string processing, ultra-sparse outputs, larger-than-memory data, and the need to explore a wide variety of transformations. Overcoming these challenges requires efficient resource management, advanced algorithms, and careful feature engineering to find the optimal combination of transformations for improved model performance.

1.3 Existing Approaches

Several existing approaches have been developed to address the challenges associated with complex feature transformation workflows and challenging data characteristics. These approaches include:

- Caching and reuse of pre-processing operations: This approach involves caching intermediate results of pre-processing

Table 1: Common Multi-pass Transformations (Phani et al., 2022a)

Transformation	Build Input	Build Output	Apply Output
Recoding	Nominal	Dictionaries	Integer
Feature Hashing	Nominal	None	Integer
Binning	Numeric*	Bin boundaries	Integer
Pass-through	Numeric*	None	Numeric
Dummy-coding	Integer	Offsets	Sparse vectors

Table 2: Overview of FTBench Datasets and Use Cases (Phani et al., 2022a)

ID	Dataset	Input Shape	Transformations	Significance	Output Shape
T1	Adult	32K × 15	Bin+DC (5), DC (9), PT (1)	Popular dataset	32K × 130
T2	KDD	98 95K ×	469 Bin (334), DC (135), Scale (469)	Skewed #distinct: 50-900	95K × 6K
T3	Criteo	10M × 39	DC (26)	Skewed large #distinct: 10-1.4M	10M × 5.8M
T4	Criteo	10M × 39	Bin (13), RC+Scale(26)	Scaled binning #distinct	10M × 39
T5	Santander	200K × 200	Bin+DC (200)	Equi-height with small #bins	200K × 2K
T6	Crypto	48M × 10	Bin (10)	Large #bins (100K), equi-width	48M × 10
T7	Crypto	48M × 10	Bin (10)	Large #bins (100K), equi-height	48M × 10
T8	HomeCredit	31K × 122	DC (16)	Popular use case	31K × 245
T9	CatInDat	3M × 24	FH+DC (24)	Feature hashing for large #rows	3M × 24K
T10	Abstract	281K × 3	Count Vectorizer	Bag-of-Words w/ large #distinct	281K × 25M
T11	Abstract	100K × 1K	Embedding (dim = 300)	Embedding large #words	100K × 300K
T12	Synthetic	100K × 100	Bin (50), RC (50)	Mini-batch transformation	100K × 100
T13	Synthetic	10M × 10	RC (10)	Varying strlen: 25-500	10M × 10
T14	Synthetic	100M × 4	RC (4)	Varying #distinct: 100K-1M	100M × 4
T15	Criteo	5M × 39	Various Combinations	End-to-end feature engineering	Scalar

operations to avoid redundant computations. By reusing pre-processed data, the runtime of subsequent transformations can be improved, especially for repetitive tasks. (Derakhshan et al., 2020)

- Interleaving element-wise transformations with data loading: This approach combines element-wise transformations with data loading, enabling concurrent processing of data while it is being loaded. This helps to reduce the overall processing time and improve efficiency. (Phani et al., 2022b)
- Static parallelism (row/column-wise): Static parallelism divides the data into fixed partitions, allowing for parallel processing of rows or columns. This approach can provide good runtime performance for simple transformations but may be suboptimal for complex workflows that involve multi-pass transformations and data with many features or distinct items. (Meng et al., 2016)

Existing solutions have good runtime for simple transformations but are suboptimal for complex, multi-pass transformation workflows, and challenging data characteristics (many features/distinct items).

2 SUMMARY OF APPROACH

2.1 UPLIFT System Architecture

The UPLIFT framework serves as the foundation for a set of specialized built-in functions for transformation. These functions operate on a data frame and a transform specification provided in the form of a JSON configuration. They optimize and execute the transformations based on the specific characteristics of the input data. The design of the UPLIFT framework draws inspiration from future-based parallelization schemes and query-processing techniques employed in column stores. One of the key features of the framework is its ability to create and optimize finely-grained task graphs, as demonstrated in Figure 1.

Taks types

(1) Build - scans an assigned feature of the input data frame and creates the necessary metadata.
(2) Output Allocation - creates and allocates the output matrix.
(3) Metadata Allocation - creates and allocates a frame for materializing all encoder's metadata.
(4) Apply - reads a feature from the input frame, encodes it using the metadata, and writes the encoded values into the output matrix.
(5) Sparse Row Compaction - compacts sparse rows in place by removing the zeros (Missing values), shifting the non-zero entries, and updating offsets.
(6) Metadata Collection - serializes the metadata into a frame.

2.2 Rule-based Optimizer

Once the global task-graph is constructed, it undergoes optimizations. The optimization process begins by collecting a representative sample of rows to estimate the number of distinct items (Haas and

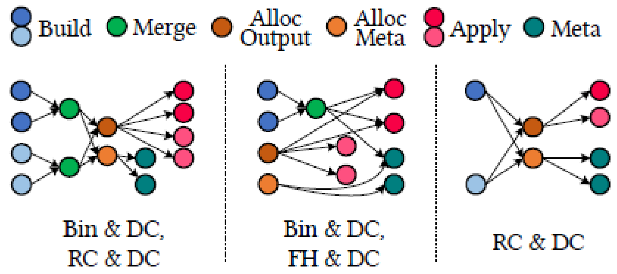

Figure 2: Three Examples of Optimized Task Graphs (Phani et al., 2022a)

Stokes, 1998) and gauge the memory requirements of the parallel tasks. Leveraging these memory estimates, along with the data characteristics and transformation specifications, the optimizer proceeds to rewrite the task-graph. This involves updating the task array and the dependency map. By consolidating dependencies within a single map, the process of introducing new rewrites is simplified. The current set of rewrites primarily focuses on:

- *Reducing Bottlenecks* - remove unnecessary synchronization barriers, concurrent build, and output dimensions are known prior to the build tasks
- *Row Partitioning* - additionally partition a column into multiple row ranges and assign a task to each block of rows
- *Choosing Number of Partitions* - increasing the number of row partitions (tasks operating on row ranges) increases memory overhead, finds a good number of partitions for each feature, and reduces the degree of parallelism if the total memory estimate exceeds the memory budget

Optimized parallelization strategies are exemplified in Figure 2, which showcases three instances, each featuring two specific features.

3 EXPERIMENT AND RESULTS

UPLIFT framework performance is compared with the existing most popular ML solutions for Feature Transformation:

- Apache SystemDS: in the figures it is called **Base** because of usage of the default configuration of Apache SystemDS and the same cache-conscious runtime implementation as UPLIFT
- Scikit-learn: currently the most popular tool for Feature Transformation
- Other ML Systems: Spark, Dask, Keras and Tensorflow - more specialized ML Systems for specific use-cases (row-based parallelization, fused transformation/training pipelines and NLP)

3.1 Datasets

To evaluate feature transformations and foster research in this area, (Phani et al., 2022a) introduce the FTBench benchmark. It combines synthetic and publicly available real datasets from **UCI** (Dua et al., 2017) **AMiner** (Dataset, 2023), and **Kaggle** (Forecasting, 2023) (Santander, 2019). Inspired by reported challenges, the benchmark includes datasets capturing choke points. These datasets cover various domains, modalities, feature transformations, data characteristics, and workload types. FTBench provides a standardized framework for evaluating feature transformations in different scenarios. Table 2 illustrates 15 use-cases with identifiers that are presented in the result figures.

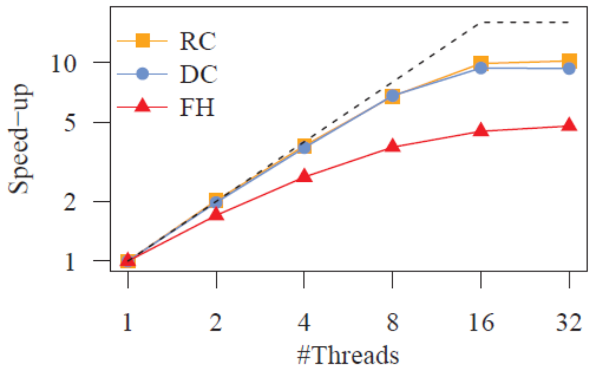

Figure 3: Speedup with Threads (Phani et al., 2022a)

3.2 Micro Benchmarks

Micro Benchmark Section focuses on measuring the performance of small, isolated code snippets or specific functions within a system, it aims to provide fine-grained insights into the performance characteristics. Here the micro benchmarks investigate speedup due to increasing threads, time breakdown of phases and impact of different numbers of row partitions.

Speedup: Figure 3 shows the result of increasing threads. Synthetic data with 5M rows, 100 categorical columns (with 100K distinct each) is used in this experiment. Recoding (RC) improves up to 10x at 16 physical cores. Dummy-Coding (DC) produces 10M columns (ultra-sparse) but equally well results. Feature Hashing (FH) shows smaller optimization because it is a memory-bandwidth bound operation.

Row Partitioning: Figure 4 illustrates that with increasing of partitions performance improves up to 8/16 partitions for Build/Apply Tasks. Feature Hashing (FH) shows to be an operation robust to partitioning (no metadata). UPLIFT optimizer also picks 8/16 partitions as an optimal number of partitions in this case.

3.3 Feature Transformation Benchmark

The Feature Transformation Benchmark (FTBench benchmark) results primarily focus on evaluating the performance of SystemDS (Base and UPLIFT) and SKlearn (Dua et al., 2017). The SKlearn pipelines, specifically the FeatureUnion, were automatically generated by parsing the same JSON transform specification used for UPLIFT and Base.

The use cases considered in the benchmark are divided into groups: small real datasets, large real datasets, scenarios involving large strings and datasets with many distinct values in synthetic data. By categorizing the use cases in this manner, the benchmark

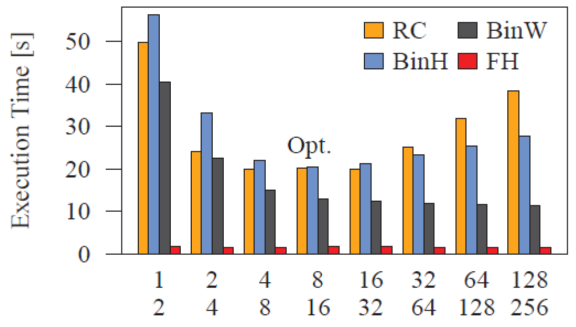

Figure 4: Build/Apply Partitions(Phani et al., 2022a)

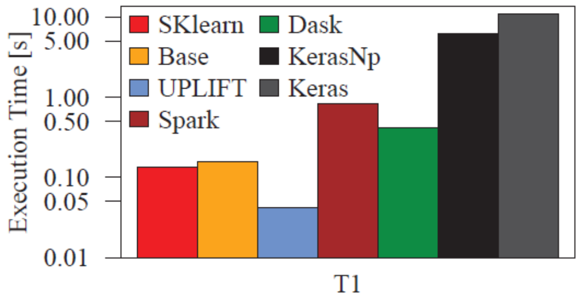

Figure 5: Small (Adult) Dataset (Phani et al., 2022a)

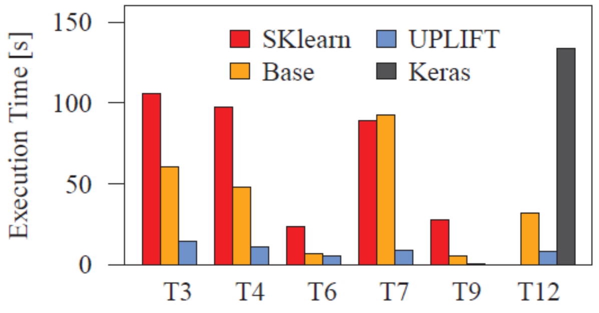

Figure 6: Large Datasets (Phani et al., 2022a)

provides a comprehensive evaluation across various data characteristics and transformation scenarios.

Small Dataset: Figure 5 shows that Base, SKlearn are 32x/52x faster than Keras. UPLIFT further improves by 6x Dask, Spark's static parallelization schemes are ineffective for smaller datasets. UPLIFT is 10x faster than Spark.

Large Datasets: Figure 6 depicts results of the experiment on larger datasets. UPLIFT is consistently faster than Base and Sklearn. On Criteo(T3). Spark is 2.5x faster than Sklearn .For T3, UPLIFT is 3x faster than Spark. Overall, dynamic parallelization schemes significantly improve across different data characteristics

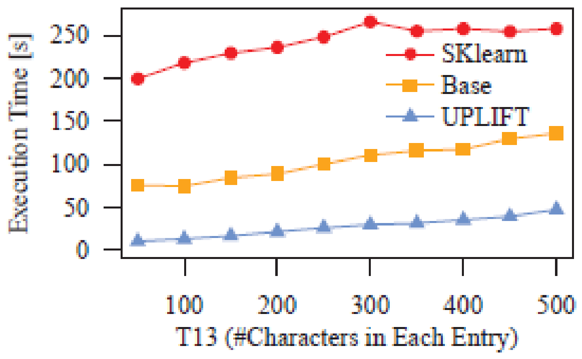

Figure 7: String Length (Phani et al., 2022a)

Data Characteristics: Figure 7 shows manipulation with the string lengths in T13 and the number of distinct values per column in T14. The results depicted in Figure 7 demonstrate that UPLIFT achieves a speedup of 7.5x compared to Base for strings with a length of 50. However, the speedup diminishes to 2.9x for strings of length 500 due to increased cache misses. Furthermore, UPLIFT improves 21x over SKlearn for smaller strings and 5x for larger strings. These findings highlight the performance characteristics of UPLIFT under varying string lengths and the advantage it holds over SKlearn for different string sizes.

4 DISCUSSION OF APPROACH

The optimization approach of the UPLIFT framework shows very promising results and even is integrated into the existing ML system Apache SystemDS. However, as the authors (Phani et al., 2022a) outline themselves the Optimizer does not show stable results in every case. Future work can focus on the specific instances that depict the instability of the UPLIFT optimizer and on the search for potential reasons for unreliable parallelization strategies. In addition to this, the framework currently only applies to local operations on CPUs. Extending UPLIFT to distributed, data-parallel operations, federated backends, and hardware accelerators (Zaharia et al., 2012) are interesting and together require necessary future work to compete with existing solutions.

5 CONCLUSION

UPLIFT is a parallel feature transformation framework with fine-grained task scheduling. Optimization based on data, workload and hardware characteristics UPLIFT showed good improvements compared to static parallelization During the development of UPLIFT, provided Feature Transformation Benchmark proved to be very useful for ML systems feature transformation performance evaluation. UPLIFT is fully integrated into Apache SystemDS. There are several future work improvements such as more explanatory optimizer work and providing reasons for unstable cases; extension to federated backends; further implementations for more baseline ML systems.

REFERENCES

C. N. Dataset. Citation network dataset, 2023. URL https://www.aminer.org/citation.

B. Derakhshan, A. Rezaei Mahdiraji, Z. Abedjan, T. Rabl, and V. Markl. Optimizing machine learning workloads in collaborative environments. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1701–1716, 2020.

D. Dua, C. Graff, et al. Uci machine learning repository. 2017.

G.-R. C. Forecasting. G-research crypto forecastingt, 2023. URL https://www.kaggle.com/c/g-research-crypto-forecasting/data.

H. Ghoddusi, G. G. Creamer, and N. Rafizadeh. Machine learning in energy economics and finance: A review. *Energy Economics*, 81:709–727, 2019.

P. J. Haas and L. Stokes. Estimating the number of classes in a finite population. *Journal of the American Statistical Association*, 93(444):1475–1487, 1998.

X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *The journal of machine learning research*, 17(1):1235–1241, 2016.

P. Molino, Y. Dudin, and S. S. Miryala. Ludwig: a type-based declarative deep learning toolbox. *arXiv preprint arXiv:1909.07930*, 2019.

A. Phani, L. Erlbacher, and M. Boehm. Uplift: parallelization strategies for feature transformations in machine learning workloads. *Proceedings of the VLDB Endowment*, 15(11):2929–2938, 2022a.

A. Phani, L. Erlbacher, and M. Boehm. Uplift: parallelization strategies for feature transformations in machine learning workloads. *Proceedings of the VLDB Endowment*, 15(11):2929–2938, 2022b.

B. Santander. Santander customer transaction prediction, 2019. URL https://www.kaggle.com/c/santander-customer-transaction-prediction/data.

S. Siddique and J. C. Chow. Machine learning in healthcare communication. *Encyclopedia*, 1(1):220–239, 2021.

K. Tsolaki, T. Vafeiadis, A. Nizamis, D. Ioannidis, and D. Tzovaras. Utilizing machine learning on freight transportation and logistics applications: A review. *ICT Express*, 2022.

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, 2012.