GPU Acceleration of Clustering Meta-feature Extraction using RAPIDS

Lucas L. Silva[1], **Ricardo Franco**[1], **André Carvalho**[2], **Wellington Martins**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Goiânia – GO – Brazil

[2]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo
São Carlos – SP – Brazil

Abstract. *Although machine learning algorithms have been successful when applied to several tasks, the selection of the most suitable for a given dataset is not straightforward. The recommendation of machine learning algorithms can be automated through the use of meta-learning, but this requires efficient methods for the characterizations of datasets, i.e. meta-features extraction. In this work we propose to accelerate the extraction of clustering-based meta-features on GPUs, taking advantage of the optimized libraries and API from the RAPIDS framework. We parallelized a well-known meta-feature extraction tool (MFE) via RAPIDS to accelerate the clustering meta-features extraction process. Our experiment shows that significantly less time is required to complete the extraction, up to 10x faster than the MFE implementation. These results are promising and suggest greater feasibility for large-scale experiments involving meta-learning.*

1. Introduction

With the increasing use of machine learning (ML), the selection of the most suitable algorithm for a given dataset is a challenging problem [Rice 1976]. This problem has been approached as a learning task, commonly referred to as Meta-Learning (MtL). MtL has been used to recommend the most suitable ML algorithm for a specific dataset [Lemke et al. 2015]. For such, a ML algorithm is applied to a meta-dataset, where each example has as predictive attributes characteristics extracted from a dataset, named meta-features, and, as the target attribute the algorithm that had the best predictive performance when applied to this dataset.

Meta-features can be categorized into six groups [Rivolli et al. 2022], which are simple, statistical, information-theoretic, model-based, landmarking, and others. All measures are calculated using the entire dataset, and their computational complexity increases polynomially with the number of instances in the dataset [Rivolli et al. 2022, Paiva et al. 2022]. Simple, statistical, and information-theoretic meta-features are typically easy (linear) to extract. Model-based and landmarking are more computationally demanding, usually requiring linearithmic time. However, meta-features from the others group can be quite time-consuming (quadratic), particularly those that involve distance calculations between all pairs of instances. This is the case for the extraction of clustering meta-features cause they require both local (compactness) and global (separation) calculations of all clusters [Zerabi et al. 2020].

The extraction of clustering-based meta-features can be very time consuming, especially when the size of the dataset is large [Ncir et al. 2021]. To address this issue, we propose utilizing the RAPIDS framework [Team 2018], which includes optimized libraries and API, to speed up the process. The main contribution is the implementation that consists of integrating the MFE with RAPIDS. Specifically, we parallelized the extraction tool MFE [Alcobaça et al. 2020] through RAPIDS, resulting in significantly faster extraction times. Our experiments, which were conducted using standard ML datasets, demonstrate that our implementation achieved up to a 10x speedup compared to the original MFE implementation. These results are promising and suggest greater feasibility for large-scale experiments involving MtL.

This article is organized as follows. Section 2 describes existing work in the domain of parallel extraction of meta-features. Section 3 gives an overview of both the MFE package and the RAPIDS framework. The proposal for parallelizing meta-feature extraction for clustering using the RAPIDS framework for MFE is outlined in Section 4. The experimental analysis and the obtained results are presented in Section 5. The last Section provides conclusions and directions for future work.

2. Related Work

The difficulty in clustering-based meta-features, especially for large datasets, has led the proposal of alternative ways of performing this task. Two approaches, not necessarily independent, have been investigated: use of parallelism to accelerate the extraction of these meta-features, and the extraction of approximate values of these measures. The work by [Zerabi et al. 2020] proposes two parallel, distributed models for clustering internal validation indices (Silhouette and Dunn), which are designed to handle the growing volume of the datasets. The proposed models were tested considering the Hadoop MapReduce framework. The work by [Luna-Romera et al. 2016] takes advantage of Spark's agility, which, unlike Haddop, works with data in memory. They present a new Spark implementation of the Silhouette and Dunn indices and highlight the potential of these indices to handle big data. The work by [Ncir et al. 2021] also uses Spark, but focuses on calculating the Dunn index. Their solution is improved by a sampling technique that proved to be scalable through an approximate calculation of the Dunn's index.

The works described make use of horizontal scalability, that is, they use several machines interconnected through high-speed networks. The proposal of the present work, however, focuses on increasing performance through vertical scalability, that is, adding computational power (GPUs) to a single existing machine. Furthermore, we compute more meta-features (eight) and do so exactly without any approximation.

3. MFE and RAPIDS

The literature on MtL often provides a superficial coverage of meta-features, only offering a general outline of the standard groups. In a recent work, [Rivolli et al. 2022] present a comprehensive list and detailed description of existing meta-features, and a taxonomy to effectively categorize and characterize meta-features. Additionally, they summarized a list of meta-feature extraction tools, with particular attention given to the Meta-Feature Extractor (MFE) tool [Alcobaça et al. 2020] that incorporates most of the meta-features and summarization functions described in the paper.

The MFE package computes eight meta-features for clustering using both Python and R programming languages. These meta-features are classified into two subgroups: the basic measures (sc - number of clusters, nre - normalized relative entropy, pb - point biserial coefficient, int - int index) and the validation measures (vdu - Dunn Index, vdb - Davies and Bouldin Index, sil - silhouette score, ch - Calinski and Harabasz index) - for more details we refer the reader to [Rivolli et al. 2022]. The latter are popular cluster internal validation indices, and are based on the information intrinsic to the data alone. Unfortunately, they have a high computation complexity, most with quadratic complexity, because they consider all the data points within the cluster structure [Deborah et al. 2010].

RAPIDS [Team 2018] was developed by NVIDIA as a set of open source data analysis tools, with the purpose of accelerating the process of analyzing large volumes of data using GPUs. With a wide range of features including data manipulation, visualization, machine learning, and statistical analysis, RAPIDS aims to provide a high-performance alternative to traditional data analysis packages. Built on the Python programming language and the CUDA data processing framework, the RAPIDS framework aims to enhance the processing capacity of GPUs allowing for more accurate and faster analysis of vast data sets, facilitating the extraction of valuable information.

While the MFE tool is capable of extracting a variety of meta-features, it can be time-consuming when dealing with large datasets, particularly when extracting clustering meta-features. To address this issue, we aim to leverage the processing capabilities of GPUs using RAPIDS to accelerate the computation. This is facilitated due to the common use of Python language in both the MFE package (PyMFE) and the RAPIDS framework.

4. Clustering Meta-feature Extraction using RAPIDS

In this section, our proposal for parallelizing meta-feature extraction for clustering using the RAPIDS framework for MFE is outlined. We take advantage of the fact that the MFE package is based on the Scikit-learn library [Pedregosa et al. 2011], and that RAPIDS CuML offers a similar API to Scikit-learn. In addition, we make use of RAPIDS CuPy that shares the same API set as NumPy and SciPy. CuPy [Nishino and Loomis 2017] provides support for multi-dimensional arrays, sparse matrices, and a variety of numerical algorithms implemented on top of them.

By performing several initial pre-computations of routines shared across multiple measures, the MFE tool avoids re-execution of code and accelerates the computation of meta-features. As these routines can take some time, we have accelerated them with the help of the RAPIDS CuPy library. We made use of the following pre-computation operations of CuPy: cupy.unique, cupy.sqrt, cupy.sum, cupy.equal, cupy.minimum, cupy-maximum, and cupy.asnumpy that moves data back to the host (CPU). Some of these operatioins are illustrated on the code snippet 1 of the precompute_clustering_class. In this method, cp.unique is used to compute, in parallel, the distinct classes and their frequencies from the y input array - line 4. The resulting classes and class_freqs arrays are then converted back to NumPy arrays using cp.asnumpy, and moved back to the host - lines 5 and 6.

After the pre-computation phase, the eight meta-features for clustering are extracted using both the CuPy and CuML libraries. The Silhouette score (sil) is implemented on the CuML and thus is accelerated by a direct call to the

```
precomp_vals =
if y is not None and
not "classes", "class_freqs".issubset(kwargs):
classes, class_freqs = cp.unique(y, return_counts=True)
precomp_vals["classes"] = cp.asnumpy(classes)
precomp_vals["class_freqs"] = cp.asnumpy(class_freqs)
classes = kwargs.get("classes", precomp_vals.get("classes"))
if y is not None and "cls_inds" not in kwargs:
  cls_inds = _utils.calc_cls_inds(y, classes)
  precomp_vals["cls_inds"] = cls_inds
return precomp_vals
```

Code Snippet 1: precompute_clustering_class

cuml.metrics.cluster.silhouette_score method. The other meta-features have to be calculated by making use of different CuPy calls so that the calculation of the desired value is accelerated by the GPU. The code Snippet 2 illustrates how the vdu - Dunn Index was accelerated. The function initializes a variable _min_intercls_dist to infinity and loops through all the values in pairwise_norm_intercls_dist to find the minimum distance between any two instances of different clusters. It then computes the Dunn Index by dividing the minimum inter-cluster distance by the maximum intra-cluster distance, and returns the result as a float - lines 7 to 10. Both cupy-maximum and cupy.minimum are performed in the GPU, and are the most expensive part of the calculation.

```
1   if pairwise_norm_intercls_dist is None:
2       pairwise_norm_intercls_dist = cls._calc_pwise_norm_intercls_dist(
3       N=N, y=y, dist_metric=dist_metric, classes=classes, cls_inds=cls_inds,)
4   if intracls_dists is None:
5       intracls_dists = cls._calc_all_intracls_dists( N=N, y=y,
6        dist_metric=dist_metric, classes=classes, cls_inds=cls_inds,)
7   _min_intercls_dist = np.inf
8   for vals in pairwise_norm_intercls_dist:
9   _min_intercls_dist = min(_min_intercls_dist, cp.minimum(vals))
10  vdu = float(_min_intercls_dist / cp.maximum(intracls_dists))
11  return vdu
```

Code Snippet 2: Dunn Intex calculation

The use of the CuPyx library was also important in parallelizing the code. With it, it was possible to speed up the calculation of entropy (cupyx.scipy.stats.entropy), the distances between pairs of two collections (cupyx.scipy.spatial.distance.cdist) and the calculation of Pearson's correlation coefficient. The complete RAPIDS modified code can be found at `github.com/poxalukas/clustering/`

5. Empirical Performance Analysis

Four extensively-used public data sets from UCI repository [Frank 2010] were employed in the experiments. The statistical summary of these data sets is as follows: Iris with 150 instances, 4 features and 3 classes; Digits with 1797 instances, 64 features and 10 classes; Breast Cancer with 35000 instances, 4 features and 3 classes; and Electric with 45000 instances, 8 features and 2 classes. The experiments were run on an Intel i7 8700k, with 32GB DDR4, SSD 1TB NVMe Gen4, GPU NVIDIA GTX 1080TI, and the Ubuntu 18.04.6 LTS operating system. To ensure that all data transfer costs are taken into account in our experiments, we report the wall times on a dedicated machine. The performance metric used is the speedup, which is calculated by dividing the time before (MFE) by the time after (RAPIDS). The reported numbers are the average of 10 independent runs.

The comparison between the runtime (in seconds) for extracting all clustering meta-features using the MFE tool and the proposed approach using the RAPIDS library is shown in Figure 1. The performance gain of the RAPIDS implementation was statistically superior with 95% confidence; we omit confidence intervals for the sake of clarity of the graph. For the Iris dataset (upper left), the MFE execution took 0.05449 seconds, while the proposed approach using RAPIDS took 0.005601 seconds, resulting in a speedup of 9.727914. That means that the RAPIDS implementation was almost ten times faster than que MFE implementation.

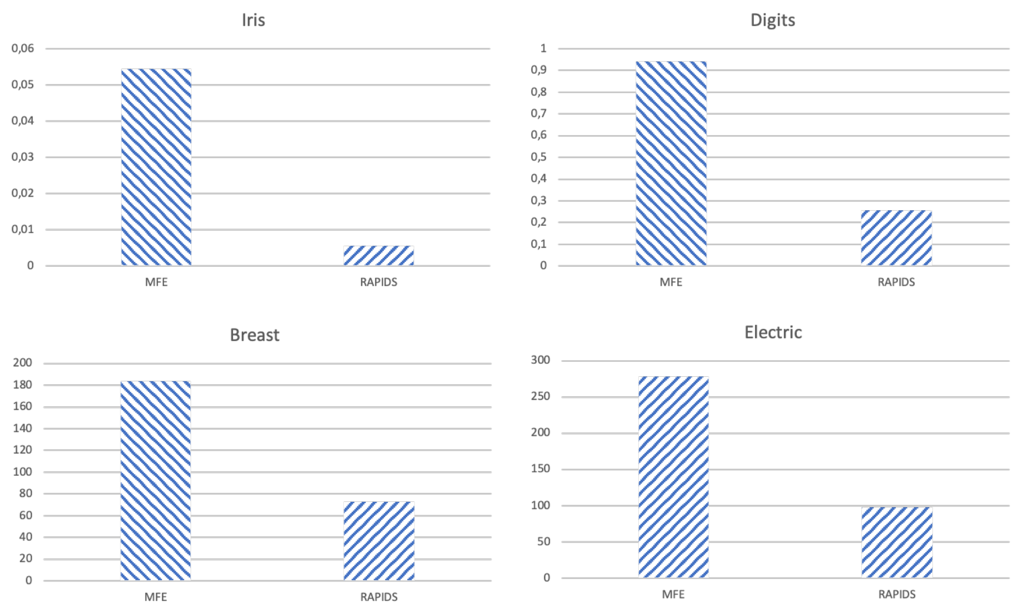

Figura 1. MFE vs RAPIDS - Runtime in seconds

As can be seen from Figure 1, the RAPIDS implementation maintains its advantage over the MFE tool for all other datasets, with the following speedus: 3.68 for Digits, 2.53 for Breast Cancer, and 2.84 for Electric. We note that the gain decreases as the size of the dataset increases. We speculate that this is due to the various data movements between the CPU and the GPU. Since we utilize independent RAPIDS calls, the system is not required to manage the memory more efficiently between the two address spaces. It is worth mentioning that the MFE tool makes heavy use of the Scikit-learn package [Pedregosa et al. 2011] and this has been optimized to utilize multiple CPU cores. In fact, we found that the MFE ran in multicore mode for some functions, one of them being the nearest neighbor search using a kd-tree.

6. Conclusions

In this work, we have leveraged the parallelism capabilities of GPUs within the RAPIDS framework to reduce the time required for extracting clustering meta-features. Experiments were performed using the original MFE package, and its modified version with RAPIDS calls. Processing standard datasets, we were able to obtain gains of up to ten times compared to the original implementation of MFE, even with modest hardware configurations. We noted that a better data management may be required for larger datasets so as to keep the data into the GPU memory between the RAPIDS calls. Another improvement would be to develop CUDA code optimized for the cluster validation indices. As future work, we plan to attack these points using larger datasets.

Referências

Alcobaça, E., Siqueira, F., Rivolli, A., Garcia, L. P., Oliva, J. T., and De Carvalho, A. C. (2020). Mfe: Towards reproducible meta-feature extraction. *The Journal of Machine Learning Research*, 21(1):4503–4507.

Deborah, L. J., Baskaran, R., and Kannan, A. (2010). A survey on internal validity measure for cluster validation. *International Journal of Computer Science & Engineering Survey*, 1(2):85–102.

Frank, A. (2010). Uci machine learning repository. *http://archive.ics.uci.edu/ml.*

Lemke, C., Budka, M., and Gabrys, B. (2015). Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44:117–130.

Luna-Romera, J. M., del Mar Martinez-Ballesteros, M., Garcia-Gutierrez, J., and Riquelme-Santos, J. C. (2016). An approach to silhouette and dunn clustering indices applied to big data in spark. In *Advances in Artificial Intelligence: 17th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2016, Salamanca, Spain, September 14-16, 2016. Proceedings 17*, pages 160–169. Springer.

Ncir, C.-E. B., Hamza, A., and Bouaguel, W. (2021). Parallel and scalable dunn index for the validation of big data clusters. *Parallel Computing*, 102:102751.

Nishino, R. and Loomis, S. H. C. (2017). Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems*, 151(7).

Paiva, P. Y. A., Moreno, C. C., Smith-Miles, K., Valeriano, M. G., and Lorena, A. C. (2022). Relating instance hardness to classification performance in a dataset: a visual approach. *Machine Learning*, 111(8):3085–3123.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier.

Rivolli, A., Garcia, L. P., Soares, C., Vanschoren, J., and de Carvalho, A. C. (2022). Meta-features for meta-learning. *Knowledge-Based Systems*, 240:108101.

Team, R. D. (2018). Rapids: collection of libraries for end to end gpu data science. *NVIDIA, Santa Clara, CA, USA*.

Zerabi, S., Meshoul, S., and Boucherkha, S. C. (2020). Models for internal clustering validation indexes based on hadoop-mapreduce. *International Journal of Distributed Systems and Technologies (IJDST)*, 11(3):42–67.