Записки научных
семинаров ПОМИ
Том 540, 2024 г.

S. Muravyov, V. Kazakovtsev, I. Usov, P. Shpineva, O. Muravyova, A. Shalyto

AN OPENSOURCE LIBRARY FOR AUTOML MULTIMODAL CLUSTERING ON APACHE SPARK

Abstract. We present a library that allows to choose and configure the clustering algorithm for multimodal datasets, i.e., for data where every object is stored not as a single vector but can be presented as a vector, text, and an image at the same time, and every modality is significant. Our library automatically finds a tradeoff between exploration and exploitation for the input data among a set of implemented clustering algorithms according to the selected internal clustering validation index. The library also implements a recommender system for the internal validation index and can predict the best fitting measure for the input data. We used Apache Spark to implement clustering algorithms, thus, it can be used on distributed computing system to clusterize big multimodal data.

§1. Introduction

Clustering is a classical and popular problem in unsupervised machine learning. It does not have a precise mathematical definition, but can be informally described as the task of dividing a set of objects into subsets, where objects within each subset are highly similar to one another and distinctly different from those in other subsets. Similarity is usually defined formally by a specified measure [1].

Solving clustering problems often entails tackling research dilemmas. Even with sufficient computational resources to explore all possible configurations of clustering algorithms, there remains ambiguity regarding how to evaluate the outcomes. Selecting an appropriate clustering algorithm primarily requires expertise and is influenced by numerous factors. As this process is predominantly carried out manually, oversights are common, rendering decision-making laborious and time-consuming [2].

Key words and phrases: automatic machine learning, multimodal models, clustering, Apache Spark.

This work was carried out as part of ITMO University project No. 623097 "Development of libraries of promising machine learning methods".

There exist a lot of different clustering algorithms, but the topic is far from exhausted: new research emerges every year. In addition to improving the accuracy and performance of algorithms, researchers are focused on self-configuring algorithms [3], automatic machine learning technologies [4], and big data clustering. Many problems require to choose a specific approach, a specific selection of the clustering algorithm and its hyperparameter tuning, which is extremely time-consuming without an automatic machine learning approach. Also, the development of information repositories and Internet technologies poses new challenges, including processing large multimodal datasets, where each object is described by more than one vector (for example, each object is simultaneously described by a row in a table, an image, and a text).

In this work, we propose a special distance measure that aggregate distances between objects calculated for individual modalities. We modify several clustering algorithms and implement them on Apache Spark to work in a distributed computing systems (clusters of computers). AutoML methods were implemented for selecting and configuring the most suitable clustering algorithm and selecting the clustering quality measure to optimize.

§2. Related Work

2.1. AutoML methods for unsupervised machine learning models. The work [1] presents an algorithm that serves as a method for selecting and configuring unsupervised learning algorithms. The MASSCAN algorithm works as follows.

A set of machine learning algorithms is given, each associated with its corresponding hyperparameter space. The goal is to find an algorithm optimal in terms of a specified clustering quality measure within fixed time. If it is reasonable to divide the tuning time budget among different clustering algorithms, a significant portion of time may be spent on tuning potentially ineffective algorithms. On the other hand, prioritizing the tuning of one algorithm without considering others may lead to a loss of information about the performance of other clustering algorithms, which could potentially improve the quality of the resulting partition. Thus, it is necessary to develop an algorithm that takes into account the compromise between the two extremes described above. This compromise is called the "exploration–exploitation tradeoff". To achieve the exploration-exploitation trade-off, an algorithm based on reinforcement learning, specifically solving the multi-armed bandit problem, was proposed in [1]. In this problem, an agent

iteratively pulls various so-called "arms" and receives a reward after each iteration. The goal of the agent is to develop a strategy for the sequence of arm activations to maximize the reward, and in general, to optimize the target function. The arms in this context represent optimization algorithms that tune the hyperparameters of each clustering algorithm. Importantly, the algorithm should be iterative so that its execution can be "paused", its current state saved and continued later in case the arm is selected again. The reward is represented as a chosen cluster quality measure improvement.

In [5], the authors proposed a meta-learning based method for choosing cluster validity index (CVI). In a broad sense, meta-learning refers to a set of approaches, methods, and techniques aimed at transferring knowledge about solving one set of tasks to expedite the search for solutions to other tasks. In a narrow sense, meta-learning involves the use of meta-models (meta-classifiers), an approach to solving the algorithm selection problem where machine learning algorithms, specifically classification algorithms, are applied to meta-data about previous machine learning experiments.

In the context of the problem formulation of meta-learning in the narrow sense, characteristics of algorithms' performance are predicted based on data characteristics (meta-features). In such a task, datasets serve as objects, so in [5] the meta-classifier is trained on the dataset of datasets.

2.2. Existing machine learning methods for multimodal datasets. One of the basic methods is learning a joint representation. This method involves independently transforming data from different modalities into vector representations and subsequently merging the vectors into a common semantic subspace using a merging model (such as EmbraceNet, EmbraceNet+, simple vector concatenation, or BiT [6]). However, this approach has drawbacks as well. The resulting representation tends to preserve the general semantics between modalities while ignoring the specific information inherent to each modality. Moreover, after applying this method it is impossible to extract the representation of each modality separately.

The coordinated representation model uses separate representations for each modality and then "merges" them into a unified coordinated space. Such representation allows to consider modality-specific characteristics [7].

Encoder-decoder architectures translate the representation of one modality into another. The encoder translates the representation of the first

modality into a hidden vector, while the decoder constructs the representation of the second modality from this vector. The model may consist of multiple encoders and decoders to separate multiple independent characteristics of one modality (for example, multiple recorded musical instruments for audio). The hidden vector is created not only considering the input modality, as it might seem, but also the output (target) modality, as the translation error propagates from the decoder to the encoder, taking into account both modalities [8].

Another important approach for multimodality are deep multimodal Boltzmann machines (DBM). DBMs are probabilistic graphical models consisting of two restricted Boltzmann machines with a shared representation layer. Modalities are connected using a correlation-based loss function. Additionally, the networks are bidirectional, allowing for data translation between modalities. A seriouse disadvantage of such models is the amount of needed computational resources [9].

Autoencoder architectures allow for the reconstruction of any modality even if one modality is missing. The training process for autoencoders involves minimizing the loss after reconstructing modalities. In [10], regularization of the loss function weight for different modalities is proposed to reduce redundancy in the resulting representation. The model is also capable of identifying modality-specific features.

Despite existing methods operating on multimodal data, none of the considered approaches is suitable for clustering problems. We cannot train our models on existing datasets since in the general case the clustering problem does not require any labels.

§3. Proposed methodology

Let $X = (X_1, X_2, \ldots, X_n)$ be a multimodal dataset, where we denote by $X_i = (X_i^1, X_i^2, ..., X_i^m)$ a single multimodal object, X_i^k is the representation of the k^{th} modality of object X_i. We denote the partition of dataset X into disjoint set of objects as $G(X)$ and target measure as $Q : G(X) \to R$.

We aim to find the partition $\hat{G}(X)$, that maximises the target measure: $\hat{G}(X) = \text{argmax}_{G(X)} Q(G(X))$

3.1. Target measure recommendation. We based our recommender system on the hypothesis [5] that allow us to assume that we can determine the best fitting CVI based on aggregated data from the dataset. Thus, we can calculate several statistical measures and predict the measure from

these statistics. The OpenML artificial set of datasets [11] was used to train the predictor model. For each dataset the best fitting CVI was determined by experts visually, allowing to create labeled dataset with internal measures as labels. Available measures include the Calinski-Harabasz measure, silhouette index, generalized Dunn index, and Score-Function.

To make the process of selecting a quality measure automatical, a distributed algorithm based on meta-learning was developed. In a broad sense, meta-learning encompasses a set of methods aimed at transferring knowledge about solving one task to accelerate the search for solutions to other tasks. In a narrow sense, meta-learning involves the use of meta-models (meta-classifiers) for algorithm selection, where machine learning algorithms, particularly classification algorithms, are applied to meta-data about previous machine learning experiments.

In the context of the narrow task setting of meta-learning, characteristics of algorithm performance are predicted based on data characteristics (meta-features). The objects in such a task are datasets. Thus, meta-classifiers are trained on datasets.

3.2. Intermodal distance. To handle multimodal objects and preserve semantics, we propose to compute distances between objects in the following way:

$$D(X_i, X_j) = \sqrt{\sum_{k=1}^{m} \alpha_k \hat{d_k}^2 (X_i^k, X_j^k)},$$

$$\alpha_k = \frac{\dim(k)}{\sum\limits_{t=1}^{m} \dim(t)},$$

$$\hat{d}_k(X_i^k, X_j^k) = \frac{d_k(X_i^k, X_j^k)}{\max\limits_{p,q \in 1,2,\ldots,n} d_k(X_p^k, X_q^k)}.$$

where $D(X_i, X_j)$ is the intermodal distance, α_k is the weight coefficient for the k^{th} modality, $dim(k)$ is the dimensionality of vector representation for the k^{th} modality, $\hat{d}_k$ is the normalised intramodal distance, and d_k is the intramodal distance between vector representations for the k^{th} modality.

In other words, we calculate distances between objects in the multimodal space as a radius vector, where its coordinates are weighted innermodal distances.

Obviously, the Euclidean norm is not the only one that can be used in the future. Moreover, this approach makes it possible to calculate inner-modal distances in a specific for each modality way.

Overall, such an approach have several advantages:

- computational simplicity;
- no need to train a complicated supervised model;
- huge variety for modifications and customizations.

3.3. Optimisation of clustering hyperparameters. Now, having found way to estimate distances between multimodal objects, we can utilise existing clustering algorithms to obtain partitions $G(X)$.

Let $C(p_1, p_2, \ldots, p_h) : X \to G(X)$ be a clustering algorithm with hyperparameters $p_1, p_2, \ldots, p_h$. We denote the set of all possible values for hyperparameter p_i as P_i and the search space for algorithm C as $P(C) = P_1 \times P_2 \times \cdots \times P_h$. P_i can be both discrete (integer, categorical hyperparameters) or continuous (real numeric hyperparameters).

As a subtask we aim to find the optimal configuration for hyperparameters $(\hat{p_1}, \hat{p_2}, \ldots, \hat{p_h}) = \text{argmax}_{(p_1,p_2,\ldots,p_h)} Q(C(p_1, p_2, \ldots, p_h)))$. Let $S_C : () \to P(C)$ be the configuration sampler, which on each iteration provides the algorithm with a new set of hyperparameters. There exist many configuration samplers (random search, grid search, Bayes optimisation, and so on) and we utilise the Tree Parzen Estimator implemented in the Optuna framework [12].

This approach allows to "pause" the process of hyperparameter optimization and calculate the reward for multiarmed bandits [1]. Also, the Optuna framework [12] is easy to install, easy to integrate to the project, and does not demand a lot of requirements, which is significant for an open source project.

3.4. Multi-armed bandit. Let $C^1, C^2, \ldots, C^a$ be different clustering algorithms with the corresponding configuration samplers $S_{C^1}, S_{C^2}, \ldots, S_{C^a}$. On each iteration, we pick one of the algorithms, sample its configuration, and return the dataset partition $G(X)$ is returned. We denote an iteration as C_i^j, meaning the j^{th} configuration for the clustering algorithm C_i, consumed time budget to evaluate and estimate the algorithm configuration as $\text{time}(C_i^j)$, and the number of executed configurations for algorithm C_i as $\text{runs}(C_i)$.

To face the bias-variance tradeoff, we use a multi-armed bandit strategy to choose the next clustering algorithm. For each clustering algorithm the

following reward function is introduced:

$$
\begin{aligned}
R(C_i) &= Q_r(C_i) + T_r(C_i),\\
Q_r(C_i) &= \frac{Q_a(C_i)}{\max_{p=1,2,\ldots,a} Q_a(C_p)},\\
Q_a(C_i) &= \max_{j=1,2,\ldots,\mathrm{runs}(C_i)} (Q(C_i^j)) - \hat{Q},\\
\hat{Q} &= \mathrm{argmin}_{G(X)} Q(G(X)),\\
T_r(C_i) &= 1 - \frac{T_a(C_i)}{\sum_{p=1}^{a} T_a(C_p)},\\
T_a(C_i) &= \sum_{j=1}^{\mathrm{runs}(C_i)} \mathrm{time}(C_i^j),
\end{aligned}
$$

where $R(C_i)$ is the reward for clustering algorithm C_i on the current iteration; $Q_r(C_i)$ is the quality component of a reward; $Q_a(C_i)$ is the the best clustering algorithm's target measure estimation; $\hat{Q}$ is the estimation of the "worst" possible partition for dataset X, approximated by evaluating Q on a randomised partition $G_{\mathrm{random}}(X)$; $T_r(C_i)$ is the time component of a reward; T_a is the total time consumption for all configurations of clustering algorithm C_i.

In simple words, on every iteration the multiarmed bandit tries to predict the most promising algorithm in terms of target function improoving.

3.5. Overall pipeline. First of all, the user uploads her dataset for preprocessing. Then we run the target measure recommendation system, if needed. Finally, the user should set a time limit and start the optimization process. On each step, the optimizer chooses an algorithm to configure its hyperparameters for a limited time, then MASSCAN re-calculates the reward and starts over while the time limit is not exceeded. Figure 1 shows the pipeline of the process.

§4. Implementation notes

4.1. Distributed computing tool. As previously mentioned, distrubuted datasets are the main priority of the library, which inevitably leads to implementation constraints such as the computation paradigm, both time and memory complexity of the algorithms, and performance issues.

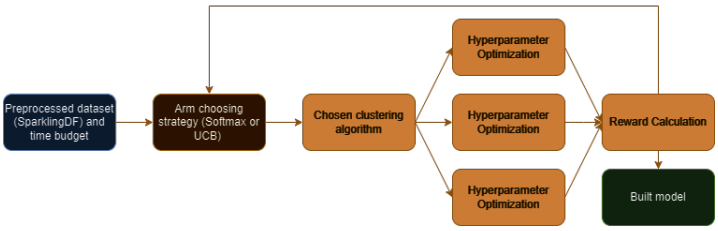

Figure 1. Overall optimization pipeline.

To operate on huge datasets, we chose to use Apache Spark. This framework is a powerful tool for clustering data on a computer cluster, thanks to its high performance due to in-memory processing, simplicity of API usage, scalability both vertically and horizontally, and in-built machine learning tools. All of the above makes Apache Spark suitable for handling large volumes of data and for solving clustering problems. It is also important to note that Apache Spark is very popular and widespread among software developers, which is important for an open source project.

4.2. Custom clustering algorithms and measures. Non-standard dataset representations (due to multimodality) and proposed intermodal distance metric do not satisfy the MLLib API out of the box. To handle multimodal datasets, custom clustering algoritms and measures implementations were introduced. Preliminary experiments show that execution of custom implementations based on PySpark API leads to excessive performance overheads and memory consumption.

To accelerate both clustering algorithms and evaluations of measures, they were reimplemented based on the Scala Spark API, because Apache Spark is natively written on Scala. Moreover, Apache Spark bootstraps JVM and holds Py4J connection between Python API and the Java process. Such PySpark architecture allows to (seamlessly from the user's perspective) inject custom Scala implementations by utilising the same JVM instance and Py4J bridge.

4.3. Computing embeddings. To convert texts or images to vector representations, the user can access pretrained deep learning models presented in the HuggingFace repository (although Sparkling currently allows only a set of predefined models). However, computing embeddings is perhaps the narrowest bottleneck in Sparkling for several reasons:

(1) the chosen model should be distributed to each node of a cluster, which causes stress load on the cluster network, especially for models with a huge number of parameters;
(2) pretrained models require Python runtime and thus cannot be evaluated on Scala; this leads to huge overheads for PySpark UDF (user-defined function) execution;
(3) images are stored in the cluser's filesystem and not directly in the dataframe (the user should only specify path to an image for each object); this requires heavy I/O operations and extra load on the cluster network; although Sparkling attempts to optimise the transformation stage by loading a batch of images into a node's memory on demand, it only reduces memory consumption but not the number of I/O operations;
(4) Apache Spark cluster with a GPU on every node is quite an expensive hardware setup, while computing embeddings on CPU dramatically slows down dataset preprocessing.

If one intends to provide multiple runs on the same multimodal dataframe, it is strongly recommended to serialize the preprocessed dataframe into Parquet format. Once it is stored, the user can launch Sparkling optimiser an arbitrary number of times while skipping the preprocessing bottleneck.

§5. Experimental results

The library supports running Apache Spark in local mode and on a YARN cluster. Experimental results are presented for the cluster from Yandex Cloud 5 machines, each with 4 vCPUs and 16 GB RAM, running Ubuntu 18.04.

5.1. Testing clustering algorithms and quality measures. Since data preprocessing is not part of the module's tasks, 9 tabular unimodal real datasets were selected for unit testing, along with several multimodal datasets generated with varying numbers of objects (ranging from 100 thousand to 2.5 million). For testing clustering algorithms, a grid of hyperparameters was defined for each algorithm, and each configuration was evaluated on the datasets described above.

Below is the grid defined for each algorithm:

- **K-means**:
  - $k \in \{2, 3, 5, 7, 11\}$
- **Birch**:

  - maxBranches $\in \{5, 12, 25\}$
  - threshold $\in \{0.1, 0.3, 0.7\}$
  - $k \in \{2, 5, 14\}$
- **Bisecting K-means**:
  - $k \in \{2, 3, 5, 7, 11\}$
  - minClusterSize $\in \{0.2, 0.5, 1.0\}$
- **DBSCAN**:
  - eps $\in \{0.02, 0.06, 0.11, 0.17\}$
  - borderNoise $\in \{True, False\}$
- **Mean Shift**:
  - Radius $\in \{0.03, 0.07, 0.12, 0.18, 0.25\}$
- **Spectral Algorithm with Similarity Matrix**:
  - eigens $\in \{7, 13, 19\}$
  - $\gamma \in \{0.1, 0.4, 1.0\}$
  - $k \in \{2, 5, 9\}$
- **Spectral Algorithm with Adjacency Matrix**:
  - eigens $\in \{7, 13, 19\}$
  - neighbours $\in \{5, 11, 20\}$
  - $k \in \{2, 5, 9\}$

The quality measures were tested on the same datasets, with the clustering results in various hyperparameter configurations serving as labels. It is important to note that it is impossible to compare the computed results with existing implementations of quality measures. This is due to the new multimodal distance metric and the absence of alternatives to some of the developed distributed implementations of quality measures within the project scope.

5.2. Testing the data preprocessing pipeline. For each dataset, it is necessary to configure the preprocessing pipeline. One of the most time-consuming and influential parameters affecting the quality of the result is the deep learning model for computing vector representations of images and text data. Table 1 shows the time spent on data preprocessing by various models for different data modalities.

As part of the project, we aimed to choose the most optimal solution based on the following characteristics: the model structure is designed for execution on central processing units (CPUs), 12 GB of RAM will be sufficient for computing embeddings and running algorithms, embedding vector size ranges from 512 to 1024, and inference speed and accuracy are acceptable within the specified tasks.

Table 1. The amount of time spent on modal data translation to vector representation and the technologies used.

Dataset	**Model for graphical data**	**Model for text data**	**Time spent, min**
100BirdsSpecies	Swin Transformer	—	47
Flick dataset	Swin Transformer	BERT	45
DiffusionDB	Swin Transformer	ALBERT	65
Wikipedia dataset	Swin Transformer	ALBERT	10
Houses dataset	Swin Transformer, EfficientNet, ConvNeXT	—	6.50, 5.00, 8.50
Amazon	—	ALBERT	280

Table 2. Recommended quality measures for synthetic datasets.

Dataset	**Recommended measure**	**Time spent, sec**
target	GD41_APPROX	18
tetra	CALINSKI_HARABASZ	9
threenorm	SCORE	24
triangle1	CALINSKI_HARABASZ	24
triangle2	SCORE	25
twenty	CALINSKI_HARABASZ	24
twodiamonds	SCORE	19
wingnut	SCORE	25
xclara	SCORE	37
xor	GD41_APPROX	24

5.3. Testing the recommendation of a quality measure based on meta-learning. This stage becomes optional if the user explicitly defines the target quality metric. Otherwise, most of the time will be allocated to computing meta-features of the dataframe, effectively equal to the overall recommendation time. Tables 2 and 3 shows the results of experiments, where the recommender system predicts the best suitable measure for various datasets.

Table 3. Recommended quality measures for real datasets.

Dataset	Recommended measure	Time spent, sec
arrhythmia	GD41_APPROX	26
balance-scale	CALINSKI_HARABASZ	21
cpu	GD41_APPROX	7
dermatology	CALINSKI_HARABASZ	17
ecoli	SCORE	10
german	GD41_APPROX	33
glass	SCORE	7
haberman	SCORE	8
heart-statlog	CALINSKI_HARABASZ	7
iono	SCORE	10

5.4. Testing the process of optimizing a quality measure. After preprocessing the data, Sparkling initiates the search for the best clustering algorithm and its optimal configuration. To determine the best algorithm and address the multi-armed bandit problem, Softmax and UCB (Upper-Confidence Bound) algorithms can be employed. For hyperparameter tuning, Optuna is recommended.

Table 4 presents information on the results obtained on the multimodal datasets: the runtime, which internal quality measure (CVI) the framework optimized, and which algorithm proved to be optimal.

In addition to real multimodal datasets, experiments were conducted on two sets of labeled tabular data (OpenML artificial and OpenML real-world). Each file from the dataset was executed with the optimization of metrics such as the silhouette index, the Calinski-Harabasz measure, the generalized Dunn index, and the Score-Function. Table 5 provides a summary of the results obtained by running Sparkling on these datasets.

The initial task in the data analysis process was to identify algorithms integrated into the default optimizer that could function effectively without the need for adjusting configuration parameters or the range of optimized hyperparameters. To accomplish this objective, a series of experiments were conducted, one of which is depicted in the Figure 2.

Figure 3 illustrates the obtained values of the internal Calinski–Harabasz measure for the standard dataset “blobs”.

Table 4. Recommended quality measures for multimodal datasets.

Dataset	**Recommended measure**	**Time spent, sec**	**Optimal Algorithm**
100BirdsSpecies	Davies-Bouldin Score	32	KMeans
100BirdsSpecies	Calinski-Harabasz Score	31	BisectingKMeans
100BirdsSpecies	Dunn Approximation	30	MeanShift
100BirdsSpecies	Silhouette Index Approx	150	KMeans
100BirdsSpecies	Score Function	77	BisectingKMeans
Flick dataset	Davies-Bouldin Score	33	Birch
Flick dataset	Calinski-Harabasz Score	35	Birch
Flick dataset	Dunn Approximation	66	MeanShift
Flick dataset	Silhouette Index Approx	34	Birch
Flick dataset	Score Function	36	BisectingKMeans
DiffusionDB	Davies-Bouldin Score	100	MeanShift
DiffusionDB	Calinski-Harabasz Score	46	Birch
DiffusionDB	Dunn Approximation	100	Birch
DiffusionDB	Silhouette Index Approx	46	KMeans
DiffusionDB	Score Function	30	BisectingKMeans
Houses dataset	Davies-Bouldin Score	40	MeanShift
Houses dataset	Calinski-Harabasz Score	32	BisectingKMeans
Houses dataset	Dunn Approximation	32	BisectingKMeans
Houses dataset	Silhouette Index Approx	27	MeanShift
Houses dataset	Score Function	32	BisectingKMeans
Hand Gestures	Davies-Bouldin Score	41	MeanShift
Hand Gestures	Calinski-Harabasz Score	53	MeanShift
Hand Gestures	Dunn Approximation	38	Birch
Hand Gestures	Silhouette Index Approx	38	KMeans
Hand Gestures	Score Function	41	Birch
Wikipedia dataset	Davies-Bouldin Score	28	MeanShift
Wikipedia dataset	Calinski-Harabasz Score	28	BisectingKMeans
Wikipedia dataset	Dunn Approximation	32	MeanShift
Wikipedia dataset	Silhouette Index Approx	29	MeanShift
Wikipedia dataset	Score Function	8	BisectingKMeans

The graph indicates an improvement in clustering quality over time. The resulting values of the primary external measures are provided in

Table 5. Summary of achieved external quality measures on labeled data.

Metric	Artificially Generated Data	Real Data
Rand Index, Mean	0.8	0.7
Rand Index, Median	0.8	0.7
Jaccard Index, Mean	0.7	0.7
Jaccard Index, Median	0.7	0.7
F-measure, Mean	0.7	0.8
F-measure, Median	0.7	0.8

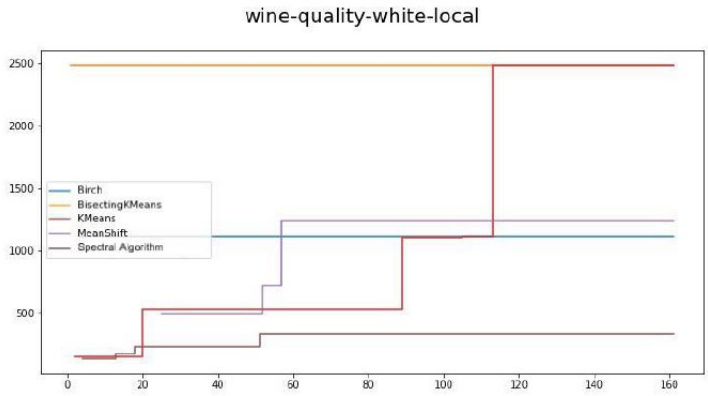

Figure 2. Algorithm performance on a dataset describing white wine; cumulative minimum of the Calinski-Harabasz measure; time resource 20 minutes; Optuna optimizer; Softmax algorithm selection strategy.

the graph's title. The presented results were obtained using the Optuna hyperparameter optimizer and the UCB bandit arm selection strategy.

Testing outcomes validate the efficiency of the developed library on small-scale datasets and underscore the importance of selecting an appropriate internal measure for optimization to attain satisfactory results.

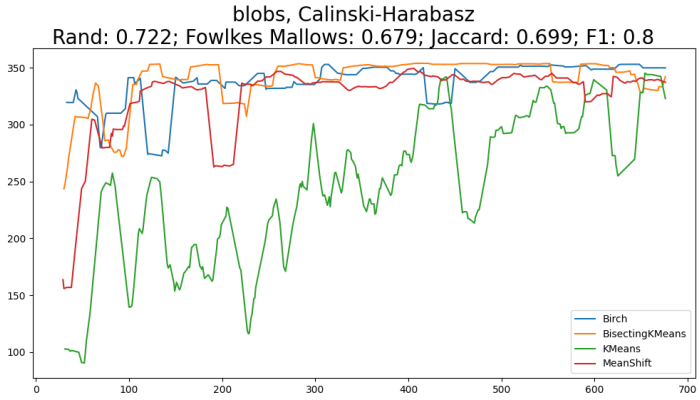

Figure 3. Quality measure values over time, shown with a moving average of 30 ticks for enhanced visual representation.

§6. Conclusion

In this work, we have developed a library for automatic multimodal clustering for distributed computing systems. An analysis of existing solutions has been conducted and various clustering algorithms supporting the use of multimodal data have been implemented.

We developed distributed versions of internal and external clustering quality metrics, conducted experiments with various hyperparameter optimization algorithms and we implemented additional clustering algorithms. Experiments have been conducted, the results of which show the applicability of the developed multimodal data clustering algorithms and the entire library as a whole.

The developed library is ready to use and available as an open-source project on GitHub and GitLab. Users can make improovements by themselves: add algorythms, embedders, clustering measures, e.t.c. Documentation and guidelines can be found on GitHub or GitLab.

As a result of the project, a library for automatic selection of clustering algorithms and their hyperparameters optimization has been developed. The library supports large amounts of data and various data formats, including multimodal data. It also includes a recommendation system for

internal cluster validity indexes. The library allows for the selection of hyperparameters for configurable algorithms and setting the time available for optimization. Experiments have shown its effectiveness in various usage scenarios, both in terms of achieved internal quality metrics and external ones.

References

1. V. Shalamov, V. Efimova, S. Muravyov, and A. Filchenkov, *Reinforcement-based method for simultaneous clustering algorithm selection and its hyperparameters optimization.* — Procedia Comput. Sci., **136** (2018), pp. 144–153.
2. V. Kazakovtsev and S. Muravyov, *Application of the automatic selection and configuration of clustering algorithms method for the Apache Spark framework.* — ACM Int. Conf. Proc. Ser. (2021).
3. O. Taratukhin and S. Muravyov, *Meta-Learning Based Feature Selection for Clustering.* — Lecture Notes in Comput. Sci., vol. 13113. Springer (2021), pp. 548–559.
4. N. Kulin and S. Muravyov, *A meta-feature selection method based on the auto-sklearn framework.* — Sci. Tech. J. Inf. Technol. Mech. Opt., **21**(5) (2021), pp. 702–702.
5. A. Filchenkov, S. Muravyov, and V. Parfenov, *Towards cluster validity index evaluation and selection.* — Proc. 2016 IEEE Artif. Intell. Nat. Lang. Conf. (AINL). IEEE (2016), pp. 1–8.
6. M.M. Al Rahhal, Y. Bazi, T. Abdullah, M.L. Mekhalfi, and M. Zuair, *Deep unsupervised embedding for remote sensing image retrieval using textual cues.* — Appl. Sci., **10**(24) (2020), p. 8931.
7. T. Baltrušaitis, C. Ahuja, and L.-P. Morency, *Multimodal machine learning: A survey and taxonomy.* — IEEE Trans. Pattern Anal. Mach. Intell., **41**(2) (2018), pp. 423–443.
8. C. Chen, D. Han, and J. Wang, *Multimodal encoder-decoder attention networks for visual question answering.* — IEEE Access, **8** (2020), pp. 35662–35671.
9. M. Suzuki and Y. Matsuo, *A survey of multimodal deep generative models*, arXiv preprint arXiv:2207.02127 (2022).
10. W. Wang, B.C. Ooi, X. Yang, D. Zhang, and Y. Zhuang, *Effective multi-modal retrieval based on stacked auto-encoders.* — Proc. VLDB Endow., **7**(8) (2014), pp. 649–660.
11. J. Vanschoren, J.N. van Rijn, B. Bischl, and L. Torgo, *OpenML: Networked science in machine learning*, arXiv preprint arXiv:1407.7722 (2014).
12. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework.* — Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (2019).

ITMO University,
St. Petersburg, Russia

E-mail: smuravyov@gmail.com

Поступило 15 ноября 2024 г.

Siberian Federal University, Krasnoyarsk, Russia
E-mail: vokzvokz@gmail.com

ITMO University, St. Petersburg, Russia
E-mail: ivan.usov.2000@mail.ru

ITMO University, St. Petersburg, Russia
E-mail: polina.shpineva@gmail.com

ITMO University, St. Petersburg, Russia
E-mail: ilyasovaolya@gmail.com

ITMO University, St. Petersburg, Russia
E-mail: shalyto@mail.ifmo.ru