The Journal of Open Source Software

Fast k-medoids Clustering in Rust and Python

Erich Schubert[1] **and Lars Lenssen**[1]

1 TU Dortmund University, Informatik VIII, 44221 Dortmund, Germany

DOI: 10.21105/joss.04183

Software

- Review
- Repository
- Archive

Editor: Mikkel Meyer Andersen

Reviewers:

- @timClicks
- @TahiriNadia

Submitted: 24 January 2022
Published: 19 February 2022

Summary

A popular technique to cluster non-Euclidean data using arbitrary distance functions or similarities is k-medoids. The k-medoids problem is NP-hard (Kariv & Hakimi, 1979), hence we need an approximate solution. The standard algorithm for this is Partitioning Around Medoids (PAM, Kaufman & Rousseeuw, 1987, 1990), consisting of a greedy initialization (BUILD) followed by a local optimization (SWAP). Alternatively, a k-means-style alternating optimization can be employed (Maranzana, 1963; Park & Jun, 2009), but this tends to produce worse results (Reynolds et al., 2006; Rosing et al., 1979; Teitz & Bart, 1968).

FasterPAM (Schubert & Rousseeuw, 2019, 2021) recently introduced a speedup for larger k, by exploiting redundancies when computing swaps for all k existing medoids. Originally FasterPAM was implemented in Java and published within the open-source library ELKI (Schubert & Zimek, 2019).

Here, we introduce the `kmedoids` Rust crate (https://github.com/kno10/rust-kmedoids) along with a Python wrapper package `kmedoids` (https://github.com/kno10/python-kmedoids) to make this fast algorithm easier to employ by researchers in various fields. We implemented the FasterPAM approach, the original PAM, and the “Alternating” (k-means-style) approach. The implementation can be used with arbitrary dissimilarities and distances, as it requires a dissimilarity matrix as input.

We chose Rust for the core functionality because of its high reliability, security, and performance, and a Python wrapper for ease of use. Both parts are documented following community best practice and available online at https://docs.rs/kmedoids respectively https://python-kmedoids.readthedocs.io. We tried to keep library dependencies to a minimum, and some dependencies (e.g., rayon for optional parallelization) can be disabled via the Rust “feature” functionality. For efficiently sharing data from Python to Rust, we rely on the well-known numpy/ndarray pairing to avoid copying data.

Performance

The original FasterPAM prototype was implemented in Java and made available as part of the ELKI open-source toolkit (Schubert & Zimek, 2019). Java often is not the best choice for a numerically heavy computation, to a large extent due to memory management; but it usually is still much faster than interpreted “pure” Python or R code (which can shine when used to drive compiled library code written, e.g., in C, Fortran, or Rust). To demonstrate the benefits of this new Rust implementation, we compare it to the original Java version (written by the same authors), and also study the additional speedup that can be obtained by parallelization using multiple threads.

We use the first N instances of the well-known MNIST data set. As the run times are expected to be quadratic in the number of instances, we report run times normalized by N^2 averaged over 25 restarts on an AMD EPYC 7302 processor with up to 16 threads for Rust. Even without parallelization, the FasterPAM in Rust with 4.48 ns per N^2, is about 4 times less than the original Java FasterPAM implementation with 21.04 ns per N^2. We primarily attribute this to being able to use a better memory layout than currently possible in Java (Project Valhalla's value types may eventually help). Using two threads in Rust, we achieve a 34% faster calculation with 2.95 ns per N^2, but we see diminishing returns when further increasing the number of threads for this data set size, caused by the overhead and synchronization cost. For small data sets, using a single thread appears beneficial, and the Python wrapper defaults to this for small data sets.

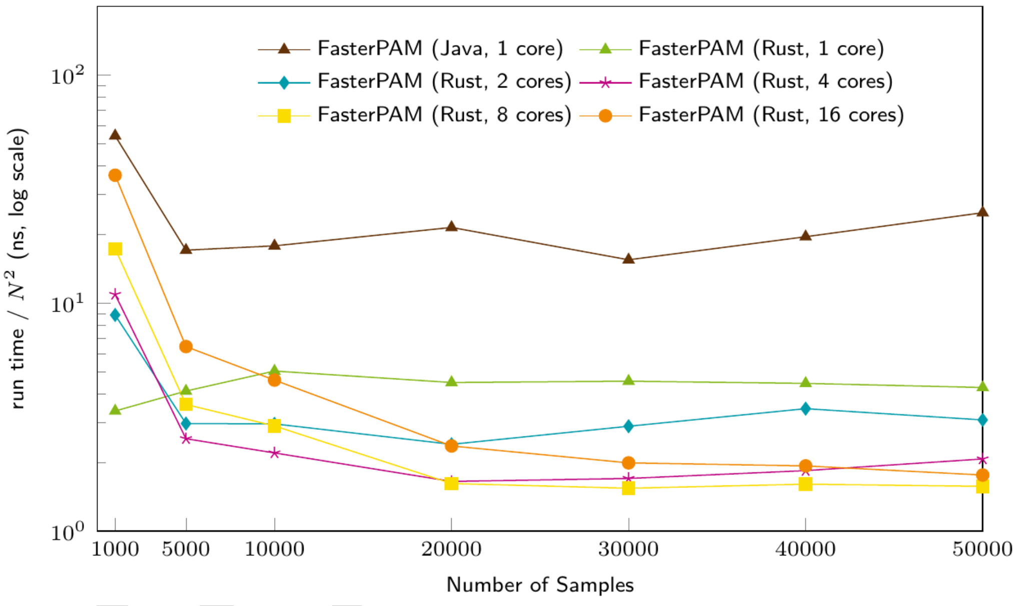

Figure 1: Results normalized by N^2 on MNIST data with k=10.

Comparison of Algorithms

Many existing libraries only implement the (worse) alternating algorithm, or the (slower) original PAM algorithm. We want to show that using this package makes it easy to find better solutions in less time. In practice, it is feasible to run multiple random restarts of FasterPAM, because the run time of the optimization is usually smaller than the time needed to compute the (reusable) distance matrix. Nevertheless, computing the distance matrix needs O(N^2) time and memory, making the algorithm only a good choice for less than 100,000 instances (for large data sets, it likely is reasonable to use subsampling).

We compare our implementation with alternative k-medoids implementations and algorithms: `sklearn_extra.cluster.KMedoids` (v0.2.0, Mathieu et al., 2020), `PyClustering` (v0.10.1.2, Novikov, 2019), `biopython` (v1.79, Cock et al., 2009), and `BanditPAM` (v3.0.2, Tiwari et al., 2020).

Our implementations (via the python wrapper) are the fastest for all algorithms (PAM, Alternating, and FasterPAM). As expected, the "Alternating" algorithm shows a significantly worse loss than PAM and FasterPAM in all implementations; while PAM has a substantially worse run time than FasterPAM and Alternating. FasterPAM achieves a similar loss to PAM (the measured differences are due to random initialization) at the shortest run time.

Table 1: Results on first 10000 MNIST instances with k = 10.

implementation	algorithm	language	ns per N²	average loss
`kmedoids`	FasterPAM	Python, Rust	**5.53**	18755648
`ELKI`	FasterPAM	Java	17.81	**18744453**
`kmedoids`	Alternating	Python, Rust	**8.91**	19238742
`ELKI`	Alternating	Java	12.80	19238868
`sklearn_extra`	Alternating	Python	13.44	19238742
`biopython`	Alternating	Python, C	13.68	19702804
`kmedoids`	PAM	Python, Rust	**212.34**	18780639
`ELKI`	PAM	Java	787.55	18764896
`sklearn_extra`	PAM	Python	1506.52	18755237
`PyClustering`	PAM	Python, C++	76957.64	18753892

Because BanditPAM cannot handle precomputed distance matrices, we evaluate BanditPAM separately, including the run time for the distance computations. On average, for MNIST 5000, 10000, 15000, and 20000 samples, BanditPAM was 55 times slower than FasterPAM in Rust. While BanditPAM claims "almost linear run time" (Tiwari et al., 2020), whereas FasterPAM has quadratic run time, BanditPAM appears to have substantial overhead,[1] and a break-even point likely is beyond 500000 samples for MNIST (a size where the memory consumption of the distance matrix makes a stored-distances approach prohibitive to use).

Conclusions

We provide a fast Rust implementation of the FasterPAM algorithm, with optional parallelization, and an easy-to-use Python wrapper. K-medoids clustering is a useful clustering algorithm in many domains where the input data is not continuous, and where Euclidean distance is not suitable, and with these packages, we hope to make this algorithm easier accessible to data scientists in various fields, while the source code helps researchers in data mining to further improve clustering algorithms.

References

Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., & Hoon, M. J. L. de. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, *25*(11), 1422–1423. https://doi.org/10.1093/bioinformatics/btp163

Kariv, O., & Hakimi, S. (1979). An algorithmic approach to network location problems. II: The p-medians. *SIAM Journal on Applied Mathematics*, *37*(3), 539–560. https://doi.org/10.1137/0137041

Kaufman, L., & Rousseeuw, P. J. (1987). *Clustering by means of medoids* (Y. Dodge, Ed.; pp. 405–416). North-Holland. ISBN: 0444702733

Kaufman, L., & Rousseeuw, P. J. (1990). Partitioning around medoids (program PAM). In *Finding groups in data* (pp. 68–125). John Wiley&Sons. https://doi.org/10.1002/9780470316801.ch2

Maranzana, F. E. (1963). On the location of supply points to minimize transportation costs. *IBM Systems Journal*, *2*(2), 129–135. https://doi.org/10.1147/sj.22.0129

[1] c.f. bug report https://github.com/ThrunGroup/BanditPAM/issues/175

Mathieu, T., Yurchak, R., Birodkar, V., & Contributors. (2020). Scikit-learn-extra – a set of useful tools compatible with scikit-learn. In *GitHub repository*. GitHub. https://github.com/scikit-learn-contrib/scikit-learn-extra

Novikov, A. (2019). PyClustering: Data mining library. *Journal of Open Source Software*, *4*(36), 1230. https://doi.org/10.21105/joss.01230

Park, H.-S., & Jun, C.-H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert Syst. Appl.*, *36*(2), 3336–3341. https://doi.org/10.1016/j.eswa.2008.01.039

Reynolds, A. P., Richards, G., Iglesia, B. de la, & Rayward-Smith, V. J. (2006). Clustering rules: A comparison of partitioning and hierarchical clustering algorithms. *J. Math. Model. Algorithms*, *5*(4), 475–504. https://doi.org/10.1007/s10852-005-9022-1

Rosing, K. E., Hillsman, E. L., & Rosing-Vogelaar, H. (1979). A note comparing optimal and heuristic solutions to the p-median problem. *Geographical Analysis*, *11*(1), 86–89. https://doi.org/10.1111/j.1538-4632.1979.tb00674.x

Schubert, E., & Rousseeuw, P. J. (2019). Faster k-medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms. *Proc. Int. Conf on Similarity Search and Applications (SISAP)*, *11807*, 171–187. https://doi.org/10.1007/978-3-030-32047-8_16

Schubert, E., & Rousseeuw, P. J. (2021). Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms. *Inf. Syst.*, *101*, 101804. https://doi.org/10.1016/j.is.2021.101804

Schubert, E., & Zimek, A. (2019). ELKI: A large open-source library for data analysis - ELKI release 0.7.5 "Heidelberg". *CoRR*, *abs/1902.03616*. http://arxiv.org/abs/1902.03616

Teitz, M. B., & Bart, P. (1968). Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, *16*(5), 955–961. https://doi.org/10.1287/opre.16.5.955

Tiwari, M., Zhang, M. J., Mayclin, J., Thrun, S., Piech, C., & Shomorony, I. (2020). Bandit-PAM: Almost linear time k-medoids clustering via multi-armed bandits. *Neural Information Processing Systems (NeuRIPS)*, 368–374.