Selecting Time Series Clustering Methods based on Run-Time Costs

Andreas Schörgenhumer[1], Paul Grünbacher[2], Hanspeter Mössenböck[3]
{firstname.lastname}@jku.at
[1] Christian Doppler Laboratory MEVSS, Johannes Kepler University Linz, Austria
[2] Institute for Software Systems Engineering, Johannes Kepler University Linz, Austria
[3] Institute for System Software, Johannes Kepler University Linz, Austria

Abstract

Clustering time series, e.g., of monitoring data from software systems, can reveal important insights and interesting hidden patterns. However, choosing the right method is not always straightforward, especially as not only clustering quality but also run-time costs must be considered. In this paper, we thus present an approach that aids users in selecting the best methods in terms of quality as well as computational costs. Given a set of candidate methods, we evaluate their clustering performance and robustly measure their actual run times, i.e., the execution time on a specific machine. We evaluate our approach using data from the UCR time series archive and show its usefulness in determining the best clustering methods while also taking costs into account.

1 Introduction

Clustering time series data provides rich information and opportunities, such as finding common patterns or general insights in the data landscape, and developing tools that can be used within clusters. Especially in large software systems, where a huge number of time series is extracted from multiple components, this can prove to be immensely useful. Commonly, researchers propose either clustering based on the raw time series or based on features [3], i.e., characteristics that describe certain properties of time series. Choosing which performs better for given datasets is not easy, especially because there are also multiple clustering models and various ways to post-process the data, which results in numerous methods. One possible solution is to run all combinations on representative or comparable data beforehand, check which one performed best and then use this best method for future data. In an industrial setting, however, run-time performance typically has to be taken into account since computations are often outsourced to cloud-computing infrastructures, which comes with additional costs. Simply using the highest performing method might thus not be the answer as it might be too slow and therefore too expensive.

In this paper, we present a way to incorporate a run-time cost model into the assessment of the clustering quality, which allows end users to determine the best methods given the additional information of their execution time. We accomplish this by measuring the actual run time of clustering models, feature calculations and post-processing options, and then merging those run times with the achieved clustering quality. Given some user-defined quality as well as run-time performance thresholds, we can easily extract the actually relevant methods. We evaluate our approach using the time series characteristics and groups introduced in [6] as well as the datasets from the UCR time series archive [4].

2 Time Series Characteristics (TSC)

Since understandability and interpretability are beneficial when dealing with time series clustering to better comprehend the resulting clusters, we use the following groups of characteristics listed in Table 1:

Group	Subgroup	#Feat.
Distributional	Dispersion	3
	Dispersion (blockwise)	10
	Duplicates	5
	Distribution	16
Temporal	Dispersion	2
	Dispersion (blockwise)	10
	Similarity	17
	Frequency	17
	Linearity	44
Complexity	Entropy	13
	Complexity (miscellaneous)	5
	Flatness	15
	Peaks	8
Statistical Tests	-	2

Table 1: TSC groups (cf. [6] for details). The number of features is the result of parameterization of some base characteristics (e.g., different block sizes).

The number of features per group determines the input size ($n \times p$, where n is the number of samples and p the number of features) for the different clustering models and post-processing options, which in turn affects the run-time measurements.

3 Approach

For time series data with n samples of length t and a given set of *methods*, the goal is to find the one best suited for clustering while also considering run-time costs. A method is a triplet that contains the clustering model, the used features (e.g., the groups listed in Table 1 with varying p features or the raw data) and the optional post-processing (which we refer to as *variants*) of these features. To assess the clustering quality, we require *labeled* data, where we calculate some external evaluation metric with the predicted clusters compared to the true clusters. Additionally, we measure the *actual* run time of all methods, i.e., the actual models, actual features and actual variants on a concrete machine, which yields the actual run-time costs that should be expected for future data. In contrast to evaluating the quality, randomly generated data suffices since the run times depend on the parameters t, n and p and not on the actual values, which is useful as we can easily adapt them to model future data. We use the actual run times rather than estimations which try to generalize. For example, the problem with run-time complexities is that similar code that results in identical estimates can still lead to run-time differences: Compiler optimizations, language intrinsics (e.g., loop iteration vs list comprehension in Python) or language mixtures (e.g., Java Native Interface) can cause a heavy impact.

To get robust run-time measurements, we execute the corresponding code (feature and variant calculation, model fitting) r times resulting in a set of measurements R. The actual run time is then calculated as the mean over a specified quantile subset of R, more formally: $\frac{1}{|R'|}\sum_{r\in R'} r$ with $R' = \{r \in R \mid r \geq q_l(R) \wedge r \leq q_u(R)\}$, where q_l is the lower quantile and q_u the upper quantile. We measure the run times of the method-triplet parts as follows:

(i) Features: We measure all features of each group with n samples of length t, i.e., the run time of the different groups in Table 1 is simply the sum of their feature calculation run times. Depending on whether we want to address parallelization, we can enable multiprocessing with a fixed number of processes, where the parallelization is done on a per-sample basis.

(ii) Variants: For each variant and feature group with p features, we measure the time it takes to post-process the n samples, i.e., the run time varies between different numbers of features.

(iii) Models: For each model and feature group with p features, we measure the time it takes to fit this model on the n samples, i.e., the run time varies between different numbers of features. If models support parallelization, we can choose to enable it.

Once we obtain the clustering quality and cost results, we can represent them with a quality-cost trade-off graph, where we plot the evaluation metric of the tested methods on the x-axis and the corresponding measured run time on the y-axis. We also introduce an optionally specifiable quality-threshold and/or cost-threshold, which help to identify those methods that are of interest to the user.

Note that evaluating both the quality and run-time costs does require significant run time in itself, however, this can be done in an offline phase.

4 Evaluation

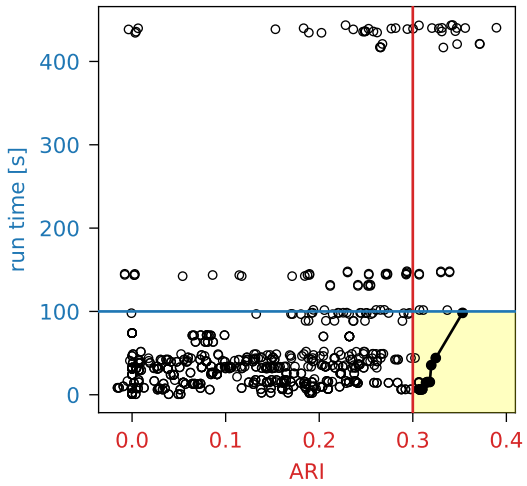
We chose the publicly available UCR time series archive [4] to evaluate our approach. The archive contains 128 labeled datasets with varying time series lengths and number of samples, split into training and test data. Since clustering is an unsupervised task, we merged the training and test data to utilize the full dataset. As the external evaluation metric, we selected the adjusted Rand Index [1] with values between $[-1, 1]$, where -1 indicates the worst possible clustering, 0 random clustering and 1 perfect clustering. For the run-time measurements, we set the number of runs to $r = 30$ and the quantile to $q_l = 0.1$ and $q_u = 0.9$. Finally, we selected the following methods to test for quality and run-time cost:

(i) Features: We tested all 4 groups and 13 subgroups (varying p) as listed in Table 1 (cf. [6] for details), as well as all groups merged (= all TSC), which resulted in 18 TSC groups. Moreover, we also ran the raw time series data ($p = t$).

(ii) Variants: For our TSC groups, we had the following options: dropping correlated features, clipping data to $[0, 1]$, logarithm-based clipping (scaling each value v above 1 with $1+\log_{10}(v)$ and each value v below 0 with $-\log_{10}(|v|+1)$) and tangent-based clipping (scaling each value v with $\frac{\tanh(2\cdot v-1)}{2\cdot(\tanh(1)+1)}$). We also combined dropping with the clipping options, and we also tested not to do any post-processing (= 8 variants in total). The raw time series were not post-processed.

(iii) Models: We use k-means (default scikit-learn implementation, parallelized), BIRCH (default scikit-learn implementation) and agglomerative clustering (default scipy `linkage` implementation) in three variants (Ward's linkage with Euclidean distance, weighted average linkage with Euclidean and with cosine distance), which resulted in 5 models.

In total, we created $18 \cdot 8 \cdot 5$ (TSC) + 5 (raw) = 725 methods. All code was written with Python 3.6.10, and the required libraries (feature and variant calculation, model fitting) had the following versions: scikit-learn 0.22.1, tsfresh 0.15.1, pandas 1.0.3, numpy 1.18.1, scipy 1.4.1, nolds 0.5.2, arch 4.13, joblib 0.14.1. Everything was run on an Intel Xeon E3-1245 v3 3400Mhz CPU with 16GB DDR3 1600MHz memory, and we set the number of parallel processes/jobs to 4. Due to paper length constraints, we only present the results of a single UCR dataset, namely *ElectricDevices*, with $n = 16637$ and $t = 96$. For simplicity, we used the same number of samples for measuring the run times, however, we could use any arbitrary n, if we expect future data to come in larger batches,



Model	Features	Variant	ARI	Run time [s]
l	complexity		0.35	98.25
l	c_entropy	01_d	0.32	44.10
k	c_entropy	log_d	0.32	35.51
l	d_dispersion_b	log	0.32	15.30
l	d_dispersion_b	01	0.32	15.27
l	t_dispersion_b	01	0.31	14.49
k	t_dispersion_b	tan_d	0.31	6.55
k	t_dispersion_b	tan	0.31	6.53
k	d_dispersion_b	log	0.31	6.38
k	d_dispersion_b	01_d	0.31	6.38
k	d_dispersion_b	01	0.31	6.36

Figure 1: Quality-cost trade-off graph of dataset *ElectricDevices* with the relevant methods (ARI ≥ 0.3, run time ≤ 100 seconds) highlighted with yellow background, where the connecting line is the Pareto front. The table on the right lists these methods. Abbreviations: models: `l` = linkage, `k` = k-means; features: `group(_subgroup)` (group abbreviated to first letter for subgroups), `_b` = blockwise; variants: empty = no post-processing, `01` = clipping to $[0, 1]$, `tan`/`log` = tangent-/logarithm-based clipping, `_d` = dropping of correlated features.

for example. The results after running all 725 methods are shown in Figure 1, including a lower-limit quality-threshold (ARI) of 0.3 and an upper-limit cost-threshold of 100 seconds (chosen for demonstrational purposes) that both define the methods considered relevant. Out of these relevant methods, we calculate the Pareto front, which we show in more detail in the table next to the plot (sorted by the ARI).

We can now easily determine, which method we want to choose for future data, considering both clustering quality as well as run-time costs. For dataset *ElectricDevices*, the results indicate that out of the relevant methods, hierarchical clustering (linkage) with the `complexity` feature group performed best. However, with only a slight decrease in quality (< 0.05 ARI) but a significant decrease in computation time, we can also use the `entropy` subgroup (over 2x faster) or even k-means with distributional or temporal `dispersion` (blockwise) subgroups (over 15x faster). Ultimately, the users must choose depending on which metric is more important to them.

5 Conclusion

In this paper, we presented how to choose the best clustering methods (triplets of features, post-processing options and clustering models) while also considering their actual run-time costs. For a given labeled dataset and a set of methods, our approach calculates any external clustering score (such as the adjusted Rand index) and robustly measures the actual run time. Using a quality-cost trade-off graph, we can visualize the results and can easily identify those methods that performed well on the dataset but also are within the computational budget. We demonstrated our approach's usefulness on a UCR dataset.

For future work, we could extend our approach with the concepts of meta-learning [2, 5], where we could predict both the expected quality as well as the run-time performance of our methods to replace our computationally expensive offline search. However, predictions of this sort are not trivial and could easily lead to unreliable results.

Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and Dynatrace is gratefully acknowledged.

References

[1] L. Hubert and P. Arabie. "Comparing Partitions". In: *Journal of Classification* 2.1 (1985), pp. 193–218.

[2] P. B. Brazdil, C. Soares, and J. P. Da Costa. "Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results". In: *Machine Learning* 50.3 (2003), pp. 251–277.

[3] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah. "Time-series clustering - A decade review". In: *Information Systems* 53 (2015), pp. 16–38.

[4] H. A. Dau et al. *The UCR Time Series Classification Archive*. `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`. 2018.

[5] B. A. Pimentel and A. C. de Carvalho. "Statistical versus Distance-Based Meta-Features for Clustering Algorithm recommendation Using Meta-Learning". In: *Procs. of the 2018 Int'l Joint Conf. on Neural Networks*. IEEE. 2018, pp. 1–8.

[6] A. Schörgenhumer et al. *Time Series Characteristics*. `https://github.com/cdl-mevss-m3/Time-Series-Characteristics`. 2020.