Astronomy & Astrophysics manuscript no. pera_etal
November 9, 2020

pyUPMASK: an improved unsupervised clustering algorithm

M. S. Pera[1], G. I. Perren[1], H. Navone[2], A. Moitinho[3], and R. A. Vazquez[4]

[1] Instituto de Astrofísica de La Plata (IALP-CONICET), La Plata, Argentina
e-mail: mspera@gmail.com
[2] Facultad de Ciencias Exactas, Ingeniería y Agrimensura (UNR-IFIR-CONICET), 2000 Rosario, Argentina
[3] CENTRA, Faculdade de Ciências, Universidade de Lisboa, Ed. C8, Campo Grande, 1749-016 Lisboa, Portugal
[4] Facultad de Ciencias Astronómicas y Geofísicas (UNLP-IALP-CONICET), 1900 La Plata, Argentina

Received August XX, 2020; accepted XXXX XX, 2020

ABSTRACT

Aims. We present pyUPMASK, an unsupervised clustering method for stellar clusters that builds upon the original UPMASK package. Its general approach makes it plausible to be applied on analysis that deal with binary classes of any kind, as long as the fundamental hypotheses are met. The code is written entirely in Python and is made available through a public repository.
Methods. The core of the algorithm follows the method developed in UPMASK but introducing several key enhancements. These enhancements not only make pyUPMASK more general, they also improve its performance considerably.
Results. We thoroughly tested the performance of pyUPMASK on 600 synthetic clusters, affected by varying degrees of contamination by field stars. To assess the performance we employed six different statistical metrics that measure the accuracy of probabilistic classification.
Conclusions. Our results show that pyUPMASK is better performant than UPMASK for every performance metric, while still managing to be many times faster.

Key words. open clusters and associations: general – methods: data analysis – methods: statistical – open clusters and associations: individual: NGC2516

1. Introduction

Galactic open clusters are of vital importance in the study of the Galaxy's chemical evolution, structure, and dynamics; as well as providing test beds for astrophysical codes that model the evolution of stars. Located largely on the disk of the Milky Way, their analysis is severely hindered by the presence of contaminating field stars, located in the foreground and background of the object of interest. These stars are projected on the observed frame and end up deeply mixed with the true members. The process of disentangling these two classes of elements (members from non-members, i.e. field stars) can be referred to as "decontamination". A proper decontamination of the cluster region is a key previous step to the analysis of the –hopefully now– clean cluster sequence in search of the fundamental parameters (metallicity, age, distance, extinction, etc.) that characterize the OC. This analysis is performed over the photometric space and requires a sequence that is as complete as possible, but also as free of contaminating field stars (non-members) as possible. The goal of a decontamination algorithm is to obtain a subset of stars that fulfills both these conditions simultaneously.

Over the years, a handful of decontamination algorithms were presented in the stellar cluster literature. Most of them are variations of the Vasilevskis-Sanders method (Vasilevskis et al. 1958; Sanders 1971) applied over proper motions, which are generally considered to be much better member discriminators than photometry. The "Unsupervised Photometric Membership Assignment in Stellar Clusters" algorithm (UPMASK), originally presented in Krone-Martins & Moitinho (2014, henceforth KMM14), has the advantage of being not only non-parametric, but also unsupervised. This means that no a priori selection of field stars is required to serve as a comparison model, as it is in the previously mentioned methods. The KMM14 article has been referenced almost 40 times in the six years span since it was published, which indicates a wide adoption by the astrophysical community.

We present here an improved version of the original UPMASK algorithm, which we call pyUPMASK as it is written entirely in Python. We believe this new package can be of great use, particularly with the advent of the third Gaia data release in the very near future. This package is made available as a standalone code, but it will also be included in an upcoming release of our Automated Stellar Cluster Analysis tool (`ASteCA`; Perren et al. 2015). Throughout the article we will refer as statistical clusters as simply "clusters", and explicitly distinguish them from stellar clusters when required.

This paper is organized as follows: in Section 2 we give a brief summary of the UPMASK algorithm, and present the details of the enhancements introduced in our code. Section 3 introduces the synthetic cluster sample used in the analysis, and the selected statistical performance metrics employed to asses the behavior of both UPMASK and pyUPMASK. Results are summarized in Section 4. Finally, our conclusions are given in Section 5.

2. Methods

We present here a brief description of the general algorithm used by UPMASK, as well as the two major enhancements introduced

in pyUPMASK. Both methods are open source and their codes can be found in their respective public repositories.[1,2]

2.1. The UPMASK algorithm

The UPMASK package is described in full in KMM14 and we will not repeat it here. We give instead a summary of its most relevant parts and of its core algorithm. The reader is directed to the original article for a more detailed description.

Assigning probability memberships to the two classes of elements within a stellar cluster (members and field stars) is a notably complicated problem, for two main reasons. First, the classes are usually very much imbalanced. This means that one of the classes (field stars) can make up a lot more than 50% of the total dataset. In some extreme cases, the frame of an observed stellar cluster can consist of over 90% of field stars, and less than 10% of actual true members. Even worse, this information (i.e., the true balance) can not be assumed to be known a priori. Second, the two classes are deeply entangled. This is particularly true in the two-dimensional coordinates space where members and field stars are mixed throughout the entire cluster region. Off the shelf clustering methods normally assume that there is some kind of frontier that largely separates the classes, with minimal overlap. This is not the case in stellar clusters analysis. UPMASK deals with both these issues in a clever and effective way, by taking advantage of the fact that we can approximate the distribution of field stars in the coordinates space with a uniform model. This is further discussed in Sect. 2.2.2.

UPMASK is composed of two main blocks: an outer loop and an inner loop. The outer loop is responsible for taking into account the uncertainties in the data (which is optional and turned off by default), and of re-running the inner loop a manually fixed number of times. The latter is required due to the inherent stochasticity of the K-Means method (MacQueen 1967), employed by the inner loop. The number of runs for the outer loop is one of the two most important parameters in the algorithm. The inner loop holds the two main parts that make up the core of the algorithm: the clustering method (K-Means, as stated before), and the "random field rejection" method (henceforth: RFR). The clustering method is applied on the non-positional features (photometry, proper motions, etc), and separates the cluster data into N clusters. The N value is determined by a parameter which determines the number of elements that should be contained in each cluster. I.e. dividing the total number of stars by this value gives N, the final number of clusters that are generated.

After the clustering method is applied, the RFR method serves the purpose of filtering those clusters identified by the K-means that are consistent with a random distribution of elements. This consistency is assessed in UPMASK by means of a two-dimensional kernel density estimation (KDE) analysis. In short: the KDE of the coordinates space of each cluster (identified by the K-Means in the previous step) is compared with the KDE of a two-dimensional uniform distribution in the same range. If these are deemed to be similar enough, the cluster is discarded as a realization of a random selection of field stars, and all its stars are assigned a value of 0. Those clusters that survive the RFR process are kept for a subsequent iteration of the inner loop. When no more clusters are rejected, the inner loop is finished all the stars within surviving clusters are assigned a value of 1. After this, a new iteration of the outer loop is initiated. The final probabilities assigned to each star are simply the averages of the (0, 1) values assigned by the inner loop, at each run of the outer loop.

The two parameters mentioned above are the most important parameters in UPMASK, since varying their value can substantially affect the method's performance. We will comment on how we selected these parameters in Sect 3.3.

2.2. The pyUPMASK algorithm

An obvious difference between pyUPMASK and UPMASK is that the former is written entirely in Python[3] instead of R[4], as is the case with UPMASK. We believe that this is a considerable advantage given the noticeable shift of the astrophysical community towards the Python language in recent years. This is made evident by large Python-based projects like Astropy[5] (Astropy Collaboration et al. 2013; Price-Whelan et al. 2018), and international conferences such as "Python in Astronomy"[6]. A recent survey found that Python is the most popular programing language in the astronomical community (Momcheva & Tollerud 2015; Tollerud et al. 2019).

The general structure of pyUPMASK follows closely the UPMASK algorithm, this is: an outer loop containing an inner loop that applies the cluster identification and rejection methods. What sets apart these two algorithms is that: 1. pyUPMASK supports almost a dozen clustering methods (UPMASK only supports K-Means); 2. pyUPMASK contains three added analysis blocks that are not present in UPMASK. In Fig. 1 we show the complete flow-chart of the pyUPMASK algorithm. The blocks colored in violet are those that are either enhanced or added in this work. The enhanced clustering methods block and the three added blocks are detailed in Sects 2.2.1, 2.2.2, 2.2.3, and 2.2.4, respectively. The remaining portions of the code are mostly equivalent to those described in KMM14 for UPMASK and, for the sake of brevity, we will not repeat their details or purpose here.

2.2.1. Clustering methods

While UPMASK supports the K-Means method exclusively (at least for now), pyUPMASK relies on the Python library `scikit-learn`[7] (Pedregosa et al. 2011a) for the implementation of most of the supported clusterings methods. This library includes around a dozen different clustering methods for unlabeled data, which are all available to use in pyUPMASK. Eventually this can be extended to support even more methods in future releases of the code, via the `PyClustering` library[8].

Once chosen, the clustering method processes the non-spatial data at the beginning of the inner loop as shown in Fig. 1. The number of individual clusters to generate is fixed indirectly through a user-selected input parameter, as done in UPMASK. Each of these clusters is then analyzed by the RFR method and kept or rejected given its similarity with a random uniform

[1] UPMASK: https://github.com/cran/UPMASK
[2] pyUPMASK: https://github.com/cran/UPMASK

[3] https://www.python.org/
[4] https://www.r-project.org/
[5] http://www.astropy.org
[6] http://openastronomy.org/pyastro/
[7] https://scikit-learn.org/
[8] https://pyclustering.github.io

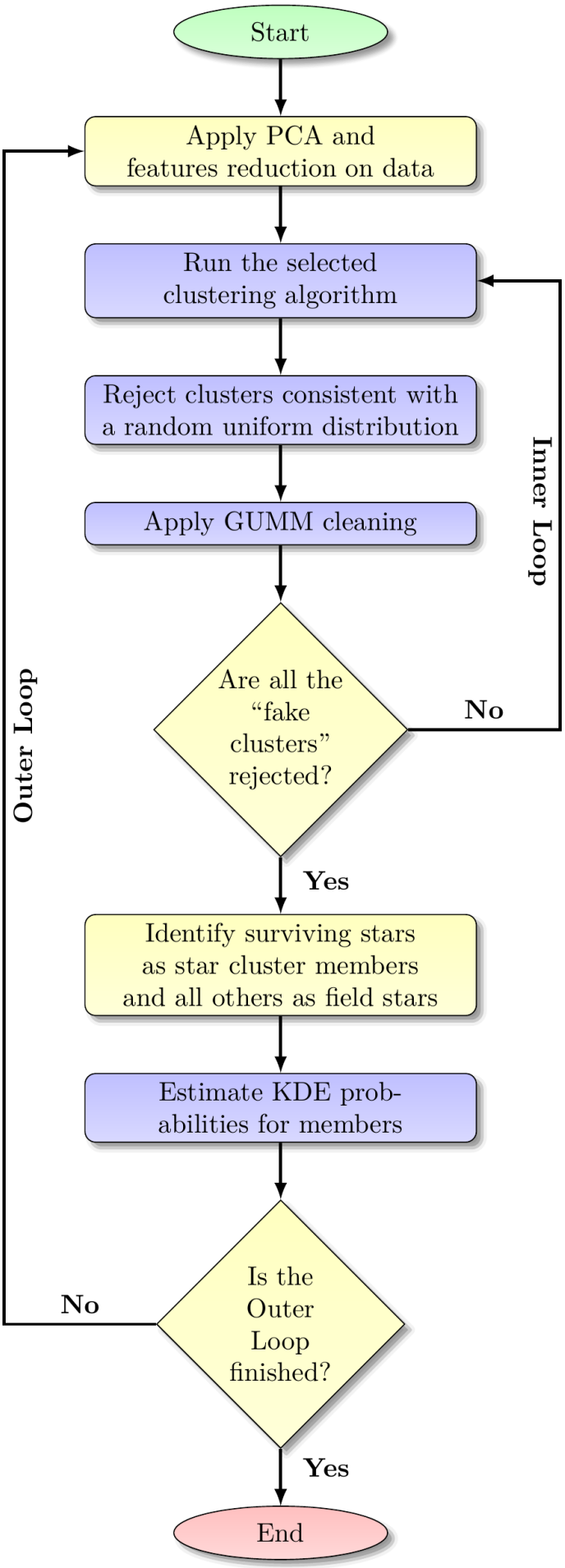

Fig. 1. Flow chart of the pyUPMASK code. The enhanced clustering block and the analysis blocks added in this work are colored in violet.

distribution of elements. This is further discussed in Sect. 2.2.2.

In Sect 4 we present a suit of tests performed with four of the methods provided by `scikit-learn`: K-Means (KMS), Mini Batch K-Means (MBK, Sculley 2010), Gaussian mixture models (GMM, Baxter 2010), and Agglomerative clustering (AGG, Zepeda-Mendoza & Resendis-Antonio 2013). In addition to these we include tests performed with two methods developed in this work: the nearest neighbors density method (KNN), which is based in the density peak approach introduced in Rodriguez & Laio (2014); and the Voronoi (VOR) method which is based in the construction of N-dimensional Voronoi diagrams (Voronoi 1908). The last three methods (AGG, KNN, VOR) have a characteristic in common: no stochastic process or approximation is employed by either of them. In other words, they are deterministic. This means that, for the same input data, the exact same result (i.e., clustering) will be obtained for different runs. Assuming that no data resampling is performed (the default setting in both UPMASK and pyUPMASK) the outer Loop then needs to be run only once, as subsequent runs would produce the same probabilities each time. For this reason we refer to these as "single-run" methods. As can easily be inferred, these are significantly faster than both UPMASK and the rest of the tested methods, which require multiple outer Loop runs.

The results obtained with the six selected methods are compared to UPMASK results obtained on the same dataset of synthetic clusters. The synthetic clusters dataset is described in Sect. 3.1.

2.2.2. Ripley's K function

After the clusters are generated on the non-spatial data, the RFR block is used to filter out those that are consistent with the realization of a random uniform distribution on the spatial data (i.e., coordinates). The hypothesis at work here is that field stars will be randomly scattered throughout the two spatial dimensions of the frame, following somewhat closely a uniform distribution. Actual star cluster members on the other hand, will present a more densely packed spatial distribution. The latter is of course an approximation to the real –and unknown– probability distribution of field stars but it is still a very reasonable one, as the results show.

UPMASK employs a KDE-based method to characterize the distribution of each cluster found in the spatial dimensions. This distribution is then compared to that of thousands of random uniform distributions, generated in the same two-dimensional range and with the same number of elements. After that, a "KDE distance" is obtained by comparing their means, maximum, and standard deviation values. If the distance between both distributions is less than a user-defined threshold parameter, the cluster is considered to be close enough to a realization of a random uniform distribution. When this condition is met, the cluster is rejected as a "fake cluster" (see Fig 1).

In pyUPMASK we introduce Ripley's K function (Ripley 1976, 1979) to asses the "closeness" of a cluster to a random uniform distribution. This function is defined as:

$$\hat{R}(r) = \frac{A}{N^2} \sum_{i}^{N} \sum_{j \neq i}^{N} I(d_{ij} < r) e_{ij} \quad (1)$$

where A is the area of the domain (our observed frame), N is the number of points within it, d_{ij} is the distance between points i, j, I is a function that returns 1 if the condition is met and 0

otherwise, e_{ij} is the edge correction (if required), and r is the scale at which the $\hat{R}$ function is calculated.

Ripley's K function is employed to test for complete spatial randomness (CSR), also called homogeneous Poisson point process, consisting basically of points randomly located on a given domain. In a two-dimensional space it is trivial to prove that if points are distributed following CSR, then $K(r)$ equals πr^2 (Streib & Davis 2011). The K function is thus a perfect match for our intended usage which is precisely to test if a set of points (stars) are distributed following uniform spatial randomness. We employ the form of the K function given by:

$$\hat{L}(r) = [\hat{K}(r)/\pi]^2 \tag{2}$$

which converges to r under CSR. Following Dixon (2014) we combine information from several distances (r values) in a single test statistic defined as:

$$\hat{L}_m = \sup_r |\hat{L}(r) - r| \tag{3}$$

(where "sup" is the supremum). Given that the observed frame's lengths are normalized by default to the range [0, 1] prior to processing, the list of distances where Eq. 3 is calculated are chosen to be in the range [0, 0.25]. This is the range advised in the `Kest` function of the `spatstat` package (Baddeley et al. 2015)[9].

The null hypothesis (H_0) for the $\hat{L}_m$ is that the points follow CSR. We need to select a critical value such that if the test is greater than that value, the test is considered to be statistically significant and H_0 is rejected. Such critical values were estimated by Monte Carlo simulations in Ripley (1979). pyUPMASK uses the 1% critical value –i.e., there is a 1% probability of erroneously rejecting H_0 (also called a Type I Error)– approximated for $\hat{L}_m$ as $1.68\sqrt{A}/N$ where A and N are the area and number of points, respectively. In future releases of the code we plan on integrating analytical expressions for the critical values, for example those obtained in Lagache et al. (2013) and Marcon et al. (2013).

pyUPMASK employs `astropy`'s implementation of the K function, which includes the required edge corrections for points that are located close to the domain boundaries. Compared to UPMASK's KDE test, the K function is not only a more natural choice for this task, it is also orders of magnitude faster.

2.2.3. Gaussian-Uniform mixture model

After the RFR block is finished and the "fake clusters" are rejected, only those stars that were found in clusters sufficiently different from a random uniform distribution of points are kept. These dataset of stars is nonetheless still affected by contamination from field stars that could no be removed. This is because these field stars were, by chance, associated with a cluster composed mainly of true star cluster members and thus not rejected. We developed a method to clean this region, applied on the 2-dimensional coordinates space which we call GUMM, as it based on fitting a Gaussian-Uniform mixture model to the dataset.

A D-dimensional Gaussian distribution can be written as:

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right) \tag{4}$$

where $\mathbf{x}$ is the D-dimensional data vector, and (μ, Σ) are the mean and covariance matrix. A GMM with K components (i.e., Gaussians) is defined as:

$$\rho_{GMM} = \sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{5}$$

where π_i are the weights (or mixing coefficients) associated with each of the K components. Similar to the GMM, we define the GUMM as a 2-dimensional mixture model composed of a Gaussian, representing the stellar cluster, and a uniform distribution, representing the noise due to contaminating field stars. The full model is then written as:

$$\rho_{GUMM} = \pi_0 \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) + \pi_1 U[0, 1] \tag{6}$$

where $U[0, 1]$ is the uniform distribution in the range [0, 1], and $\pi_x(x = 0, 1)$ are the unknown weights for each model. No restrictions are imposed on the position, shape, or extension of the 2D Gaussian representing the stellar cluster. Following the recipe employed by the classic GMM, we use the iterative Expectation-Maximization algorithm (EM, Dempster et al. 1977) to estimate these weights, as well as the mean and covariance of the 2D Gaussian. After the EM algorithm converges to a solution, each star is assigned a probability of belonging to the 2D Gaussian (i.e., to the putative cluster). We then need to decide which stars to reject as field stars, based on these probability values. To do this the percentile distribution of the probabilities is generated, and the value where the curve begins a sharp climb towards large probabilities is automatically identified as the probability cut. The value corresponding to the climb in the percentile curve is estimated with the method developed in the `kneebow` package[10]. The user can input a manual value for this probability cut (or even skip the GUMM altogether), but after extensive testing this method has proven to give very good results and it is thus the recommended default. Stars below this value are rejected as contaminating field stars, and the surviving stars are kept as cluster members.

The results of processing a group of stars with the GUMM can be seen in Fig. 2. The a. plot shows the 2D coordinates space after the RFR block rejects those clusters consistent with a random uniform distribution. It can be seen that, even after clusters mainly composed of field stars are rejected, the central overdensity is still visibly contaminated by the surrounding field stars. The b. plot shows the probabilities assigned to each star of belonging to the 2D Gaussian, by the GUMM process. In plot c. we show the percentile diagram for the probabilities, where the red line shows the value where the cut is imposed. Finally, plot d. shows the region after those stars with probabilities below the aforementioned cut are rejected.

This process, although almost trivial at first glance, greatly improves the purity of the final sample of estimated true members at very little cost regarding completeness. The hypothesis at work is of course that the putative stellar cluster is more concentrated in the coordinates space than regular field stars, as previously stated.

2.2.4. Kernel density estimator probabilities

Once a run of the inner loop is finished, each star in the observed field is classified to be either a cluster member or a field star.

[9] http://spatstat.org/

[10] https://github.com/georg-un/kneebow

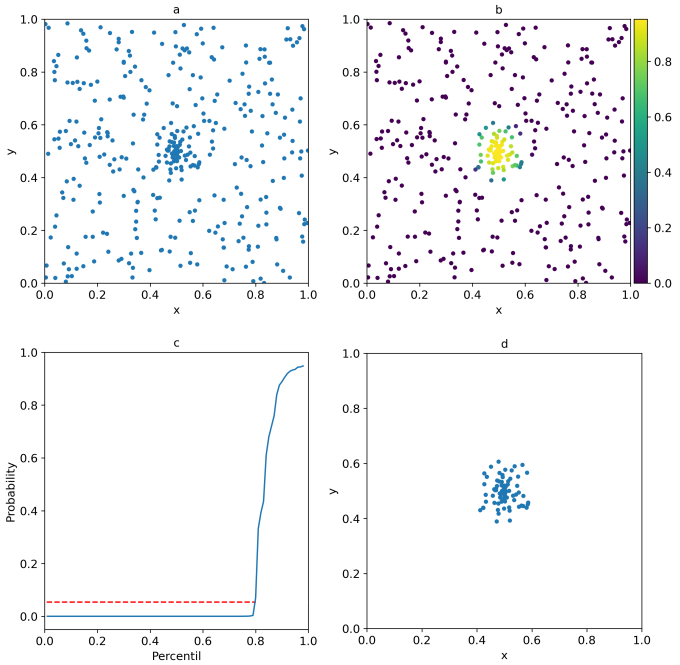

Fig. 2. The GUMM process in four steps. a: The set of stars that survived the RFR block. b: Probabilities assigned by the GUMM to all the stars in the frame. c: The method for selecting the probability cut value using a percentile plot. d: The final set cleaned from most of the contaminating field stars.

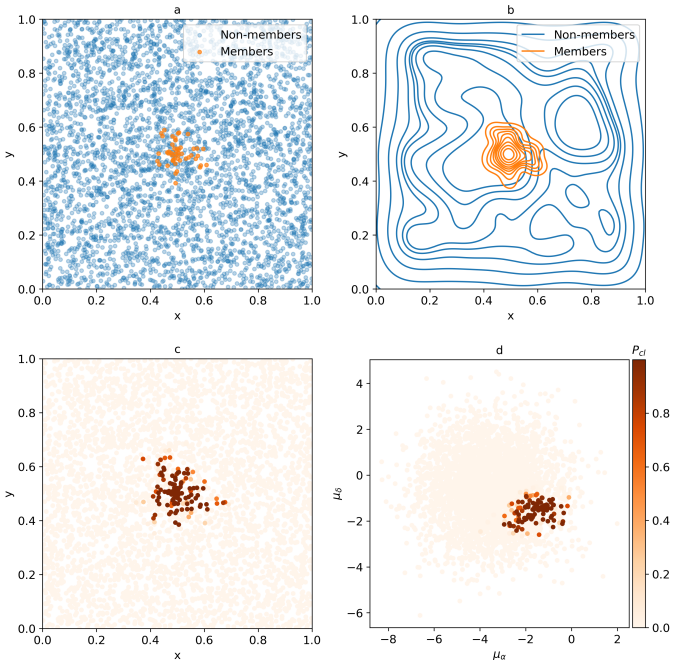

Fig. 3. KDE probabilities method, shown in the coordinates space. a: members and non-members, as estimated by the inner loop process. b: KDEs for both classes. c: final P_{cl} probabilities assigned in the coordinates space. d: same as c. but for proper motions.

This is a hard binary classification, meaning that only probability values of 0. and 1. are assigned. The KDE block takes these binary probabilities and turns them into continuous probabilities in the range [0, 1]. This improves the final results in general by assigning more realistic probability values. Furthermore, this block is essential for single-run clustering methods (defined in Sect 2.2.1). Clustering methods like K-Means or GMM require multiple outer loop runs. The final probabilities are then estimated by averaging all the binary probability values, which breaks the binarity. Single-run methods work, as the name indicates, on a single run of the outer loop. This means that without this block, single-run methods would assign probabilities of 0 and 1 exclusively.

The KDE probabilities are assigned after a full run of the inner loop, with all stars classified as either members or non-members. The process is as follows:

1. Separates each of those two classes into different sets
2. Estimate the KDE for each class, using all the available data i.e.: coordinates plus the data dimensions used for clustering (photometry, proper motions, etc.)
3. Evaluate all the data in the frame in the KDE obtained for each class
4. Use the above evaluations as likelihood estimates in the Bayesian probability for two exclusive and exhaustive hypotheses (i.e., a star belongs to either the members distribution or the field stars distribution)

The final cluster membership probability (using uniform equal priors) is written as:

$$P_{cl} = KDE_m/(KDE_m + KDE_{nm}) \quad (7)$$

where KDE_m and KDE_{nm} are the KDE likelihoods for the members and non-members (field), respectively. The process can be seen in Fig. 3 for the coordinates dimensions (even though it is applied on all the data dimensions). The a. plot shows the two classes, members and non-members, generated after the inner loop is finished. In plot b. we show the 2-dimensional coordinates KDEs for both classes, reminding again that this is applied on all the data dimensions. The c. plot shows the non-binary P_{cl} probabilities assigned by the method in the coordinates space. Finally, the d. plot is equivalent to the c, plot but for the proper motions space.

3. Validation of the method

In order to perform a thorough comparison of the performance of pyUPMASK versus that of UPMASK, we applied both methods to a large number of synthetic clusters and quantified the results using numerous statistical metrics. We describe here the set of synthetic clusters, the selected metrics, and the reasoning behind the choice of input parameters.

3.1. Synthetic datasets

We employed a total of 600 synthetic clusters to analyze the performance of UPMASK and pyUPMASK, the latter in the six configurations mentioned in Sect 2.2.1. This set is divided into a sub-set of 320 clusters, and another one of 280 clusters. The first sub-set is equivalent to that used in the original UPMASK article (KMM14), as it is composed of clusters with synthetic photometry generated with the same process as that used in KMM14. We will refer to this sub-set as PHOT hereinafter. The second sub-set contains 280 clusters and, although it also contains synthetic photometry, it was generated adding synthetic proper motions to all the stars in the frame. The idea is then to see how both algorithms handle the case were only photometry is available, and

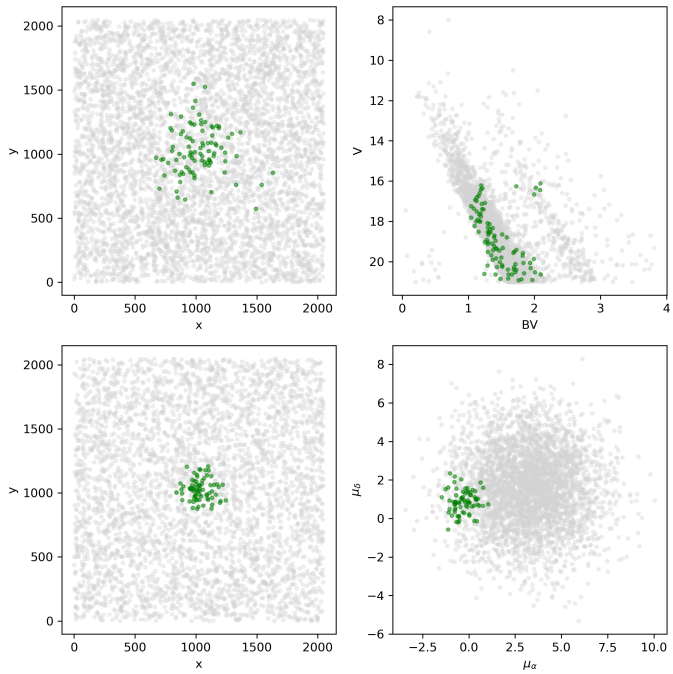

Fig. 4. Top row: coordinates and color-magnitude diagram for a PHOT synthetic cluster with moderate CI. Bottom row: coordinates and vector-point diagram for a PM synthetic cluster with moderate CI.

the increasingly common case (thanks to the Gaia survey) were proper motions with very reasonable quality are available. Clusters were generated with a wide range of field star contamination. The level of contamination is measured by the contamination index (CI), defined as the number of field stars to cluster members in the frame. The maximum CI in our set of synthetic clusters is 200.

In Fig. 4 we show examples of a PHOT (top) and PM (bottom) synthetic clusters, generated with moderate contamination (CI≈50).

3.2. Performance metrics

A proper choice for evaluating the classification performance of a probabilistic model (such as UPMASK or pyUPMASK) is a debate that carries on even today (Hand 2009; Hernández-Orallo et al. 2012). Different metrics or scoring rules will yield different results regarding the model's performance (Merkle & Steyvers 2013), which means that relaying on a single metric is not recommended. This is particularly true when dealing with datasets that can be highly imbalanced, as is our case. We thus chose to employ multiple metrics. By combining all of them we can be sure that we obtain a non-biased assessment of the overall performance of pyUPMASK versus UPMASK.

We selected six metrics that can be divided into two groups of three each. The first group consists of strictly proper scoring rules, which guarantee that they are only optimized when the true classification is obtained. This group is composed of the following metrics:

• *Logarithmic Scoring Rule*:

$$LSR = 1 + \frac{1}{N}\sum_{i=1}^{N} y_{true}\log(p) + (1 - y_{true})\log(1 - p) \tag{8}$$

where N is the number of elements, $y_{true} \in \{0, 1\}$ is the true label, and $p = \Pr(y{=}1)$ is the probability that $y{=}1$, i.e., the probability that the element belongs to the class identified with a 1 (Good 1952). The LSR (also called Log-loss or Cross-entropy) heavily penalizes large differences between y_{true} and p.

• *Brier Score Loss*:

$$BSL = 1 - \frac{1}{N}\sum_{i=1}^{N}(p - y_{true})^2 \tag{9}$$

equivalent to the mean squared error for binary classification, it was originally introduced in Brier (1950).

• *H Measure*:

$$HMS = 1 - \frac{L}{L_{max}} \tag{10}$$

where L is the loss function, and L_{max} is the maximum loss (the expression for the loss function is much too mathematically involved to be presented here, it can be seen in full in Hand 2009). This is a relatively new metric. It was developed as a replacement of the popular AUC (area under the receiver operating characteristic curve) score; now known to be an incoherent performance measure and thus not recommended (Lobo et al. 2008; Parker 2011; Hand & Anagnostopoulos 2014). The HMS automatically handles unbalanced classes by treating the misclassification of the smaller class (in our case almost always true members, except for extremely low CI values) as more serious than those of the larger class.

It is worth noting that the definitions of LSR and BSL were altered from their original forms by inverting and adding a +1. This way all the metrics defined assign 1 to a perfect score.

The three metrics in the first group can be used directly on the membership probabilities in the [0, 1] range, resulting from UPMASK or pyUPMASK. The second group defined below, consists of scoring rules that are applied to binary classifiers. These are the types of metrics used in the original KMM14 article and we employ them here for consistency[11]. In the definitions that follow TP is a true positive (a member star correctly classified as such), TN is a true negative (a field star correctly classified as such), FN is a false negative (a member star incorrectly classified as field), and FP is a false positive (a field star incorrectly classified as member):

• *True Positive Rate*:

$$TPR = \frac{TP}{TP + FN} \tag{11}$$

also called sensitivity or recall, it measures the proportion of true members that are correctly identified.

• *Positive Predictive Value*:

$$PPV = \frac{TP}{TP + FP} \tag{12}$$

also called precision, it measures how many stars classified as members are in fact true members.

[11] Notice that in KMM14 the statistical measures TPR and MMR are incorrectly defined. What the authors call "TPR" is in fact the PPV, and what they call "MMR" is actually the properly defined TPR.

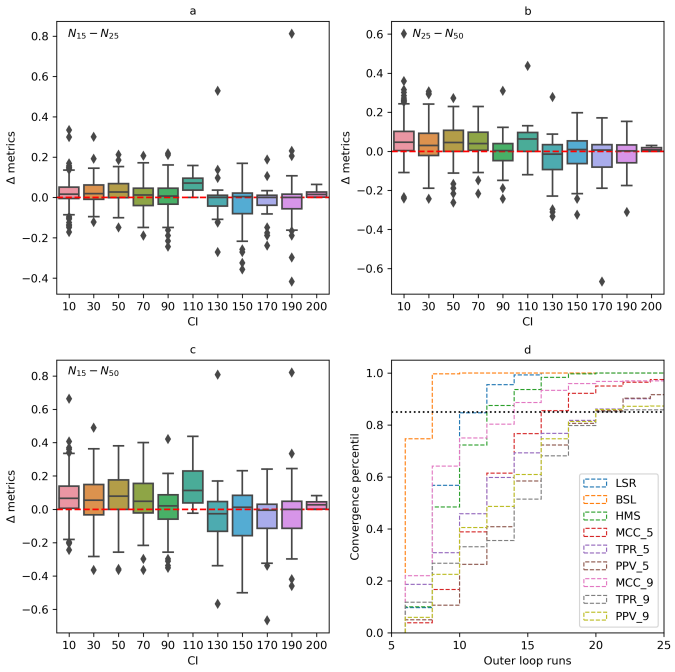

Fig. 5. a, b, c: boxplot of the combined metrics difference versus CI for the 100 synthetic clusters used in the test. Combinations for the N_{15}, N_{25}, N_{50} values are shown. d: outer loop convergence analysis. The convergence percentile of the nine metrics versus the number of outer loop run is shown. The black dashed line marks the 90% convergence point.

• *Matthews Correlation Coefficient*:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (13)$$

introduced in Matthews (1975), it can be thought of as an equivalent to Pearson's correlation coefficient for binary classifiers. Unlike the TPR and PPV, the MCC also takes the TNs into account. It is recommended when dealing with imbalanced classes, as is our case.

To turn the problem into one of binary classification and be able to use the three metrics defined in the second group, we must first select a probability threshold that separates the stars into both classes (members and non-members). In KMM14 a single threshold of 90% was used. Since the choice of a threshold can affect the results from these three metrics, we decided to use two different thresholds: 50% and 90%. This way we end up with nine metrics to test the performance of UPMASK and pyUPMASK: LSR, BSL, HMS, TPR_5, PPV_5, MCC_5, TPR_9, PPV_9, and MCC_9; where the sub-index 5 and 9 indicate the 50% and 90% thresholds respectively.

3.3. Input parameters selection

There are two main parameters in UPMSAK and pyUPMASK that affect the outcome of the methods: the number of stars per cluster, and the number of runs of the outer loop. The former, which we will refer to as N_{clust}, was investigated in KMM14 where the authors concluded that a value between 10 to 25 is appropriate. In the latest version of the UPMASK code, depending on how it is run, the default value for N_{clust} is either 25 or 50[12]. We performed our own tests using 100 synthetic clusters (50 PHOT and 50 PM) covering the full CI range, selected at random from the full list of 600 mentioned in Sect 3.1. This set was analyzed with the nine performance metrics described in Sect 3.2. In Fig. 5 we show the results obtained for three N_{clust} values 15, 25, and 50. We combined all the metrics for the 100 synthetic clusters into one set and subtracted these (900) values for a given N_{clust} value from another. The results are plotted versus the CI of the synthetic clusters. From a. to c. the combinations $N_{15} - N_{25}$, $N_{15} - N_{50}$, and $N_{25} - N_{50}$ are shown where a positive value means that the N_{clust} value on the left performed better than the one on the right, and vice-versa for negative values. As can be seen, the differences are rather small and don't tend to change for different CI values. We thus decided to use the middle value $N_{clust} = 25$ for all the UPMASK and pyUPMASK runs, as a reasonable number of default stars per cluster for all the CI range.

Deciding how many times the outer loop should run is the other important parameter: a low number will terminate the code before it is able to present fully converged probability values, and a large number will waste processing time. We processed the same set of 100 synthetic clusters with $N_{clust} = 25$ and analyzed when each of the nine metrics converged to a stable value. The stabilization point is defined as the outer loop run where the metric changes inside the ±0.025 range for 5 consecutive runs. Results are shown in Fig. 5 d. plot as a the convergence percentile (i.e., the percentage of clusters that have converged) for each metric, versus the outer loop run. It can be seen that almost all the metrics reach a convergence above 90% (horizontal black dashed line) before the 25th outer loop run. The two exceptions are TPR_9 and PPV_9 which still show a convergence above 85% before the 25th run. Given these results we will use 25 runs in the outer loop for all the UPMASK and pyUPMASK analysis, with the obvious exceptions of the single-run methods described in Sect 2.2.1.

The PHOT set was processed using all the available photometry as input ($V, B-V, U-B, V-I, J-H, H-K$) but selecting only the four principal dimensions after the principal component analysis dimensionality reduction. For the PM set we used only the proper motions ($\mu_\alpha, \mu\delta$), and no photometry. Proper motions are generally regarded as better cluster members discriminators than photometry. We were able to confirm this by checking that the results (with either UPMASK or pyUPMASK) degraded if photometry was added to the proper motions as input data for the PM set.

4. Results

To ensure that the results are comparable between the pyUPMASK and UPMASK runs, all the analysis were performed on the same computer cluster. In what follows results are classified as a pyUPMASK win if the metric value is greater than that obtained with UPMASK, and a pyUPMASK loss (or UPMASK win) if it is smaller. We allow for a small range of ±0.005 to act as a "tie zone" where both methods can be thought of as performing equally well.

In Fig 6 we show the metrics for the combined PHOT and PM sets of synthetic clusters, for each of the six clustering methods used in pyUPMASK. Green, yellow, and red bars depict the

[12] It is 50 if one runs the code using the `UPMASKfile` function, and 25 if one uses the `UPMASKdata` function.

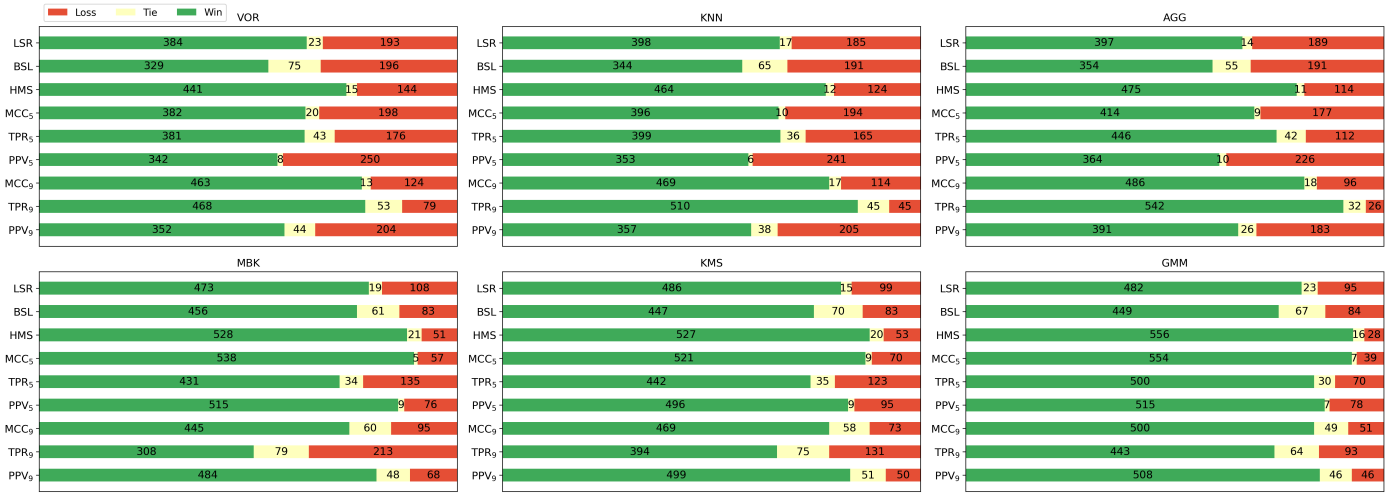

Fig. 6. Results for the 600 synthetic clusters processed with the six clustering methods used in pyUPMASK versus UPMASK. For each metric green, yellow, and red bars represent the cases where pyUPMASK won, tied, and loss to UPMASK, respectively.

proportion of cases where pyUPMASK performed better (win), equally well (tie), and worse (loss) than UPMASK, respectively. It is easy to see that, although with some variation across clustering methods, pyUPMASK performed better than UPMASK for all the methods and all the metrics. This is an outstanding result that unmistakably shows the power of pyUPMASK. The three methods that apply multiple outer loop runs (MBK, KMS, GMM) show a clear advantage over the remaining single-run methods, regarding the proportion of cases won by pyUPMASK.

The results can be compressed into a simpler single matrix plot as shown in Fig. 7. We display here the pyUPMASK win minus loss percentage difference for each clustering method and metric. This difference ranges from -100, which would indicate that UPMASK performed better on all 600 synthetic clusters, to 100, indicating that pyUPMASK was the better performer for the 600 clusters. A value of 0 would indicate an equal number of cases won by both methods. As can be seen, all the squares in the matrix are positive (the smallest one being the TPR_5 metric for the VOR method) which again shows that pyUPMASK is a much stronger method than UPMASK, measured by any of the employed metrics. The advantage of the MBK, KMS, GMM methods over the single-run methods is easier to see here. The only exception is the TPR_9 metric where the VOR, KNN, and AGG methods show a larger differential than the remaining multiple-runs methods; i.e., more true members are classified as such. This comes at the expense of the PPV_9 metric where the MBK, KMS, GMM methods show much larger values; i.e., fewer field stars are incorrectly classified as members. Other than this, there is no visible relation between any clustering method and a given performance metric.

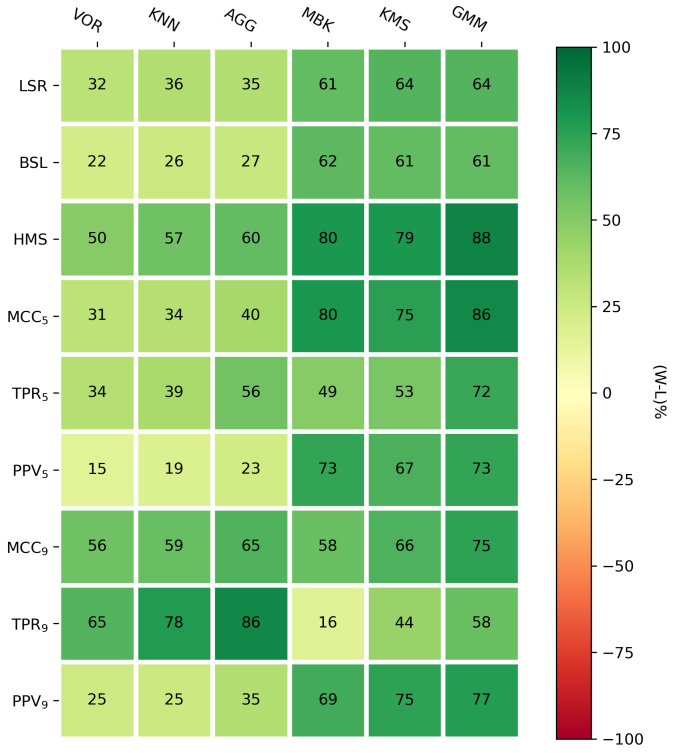

Fig. 7. Matrix plot of the clustering methods versus the nine defined metrics for the 600 synthetic clusters. Each square shows the win minus loss percentage difference referring to cases where pyUPMASK won or lost to UPMASK, respectively.

In Fig. 8 we show the dependence with CI for the pyUPMASK minus UPMASK difference for all the metrics, for each clustering method. A positive value (green region) means that pyUPMASK performed better while a negative value (red region) means that it performed worse than UPMASK. The PHOT and PM sets are shown with triangles and circles, respectively. There is no apparent trend with CI for the results of any clustering method. What is clear is that pyUPMASK performs even better for the PM set as evidenced by the overall larger (more positive) differences, particularly for clusters with large CI values. This is a very desirable result taking into account that high quality proper motions are becoming more accessible very fast.

We can further compress the results by combining the win percentages for each metric into a single value, for each of the clustering methods tested in pyUPMASK. I.e., we take the 5400 results for each clustering method (600 synthetic clusters times nine metrics) and calculate the percentage where pyUPMASK won over UPMASK. The same process can be applied to the percentage of synthetic clusters where UPMASK won, to obtain

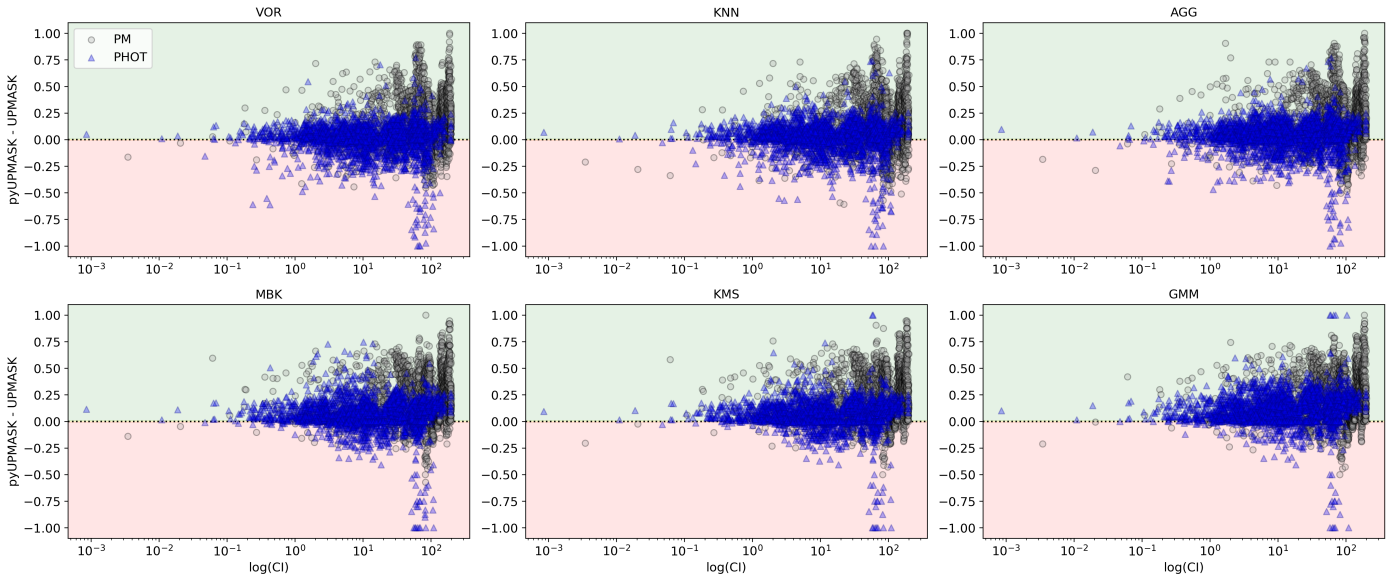

Fig. 8. Differences between pyUPMASK versus UPMASK results for all the metrics combined, versus the CI (shown as a logarithm). Each clustering method is shown separately, as are the PHOT and PM sets using blue triangles and black circles, respectively. The red and green regions correspond to the regions where pyUPMASK loses and wins to UPMASK, respectively.

Table 1. Aggregated results for all the metrics and all the synthetic clusters, for the six pyUPMASK clustering methods used, as percentage of results won by pyUPMASK and UPMASK, respectively. The missing percentage to add up to 100 corresponds to ties. The second rows for each method show the minimum and maximum values obtained for any metric (shown in parenthesis), for that method.

Method	pyUPMASK min | max	UPMASK min | max
VOR	66	29
	55 (BSL) | 78 (TPR_9)	13 (TPR_9) | 42 (PPV_5)
KNN	68	27
	57 (BSL) | 85 (TPR_9)	8 (TPR_9) | 40 (PPV_5)
AGG	72	24
	59 (BSL) | 90 (TPR_9)	4 (TPR_9) | 38 (PPV_5)
MBK	77	16
	51 (TPR_9) | 90 (MCC_5)	8 (HMS) | 36 (TPR_9)
KMS	79	15
	66 (TPR_9) | 88 (HMS)	8 (PPV_9) | 22 (TPR_9)
GMM	83	11
	74 (TPR_9) | 93 (HMS)	5 (HMS) | 16 (LSR)

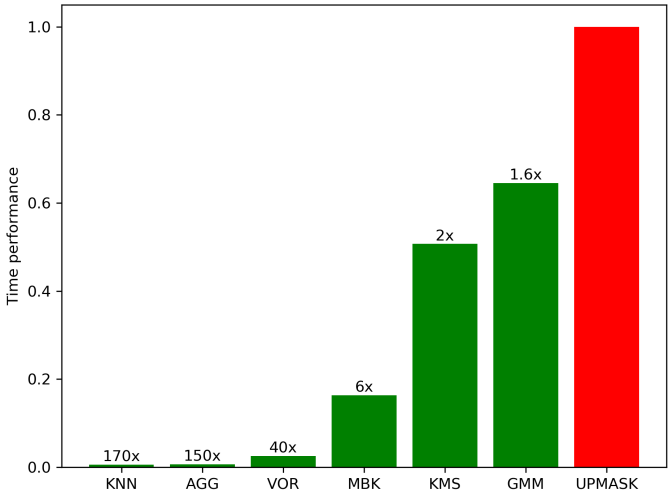

Fig. 9. Amount of time employed in processing the 600 synthetic clusters by each pyUPMASK method and UPMASK. Bars are normalized so that UPMASK corresponds to a value of 1.

a similar win percentage. The results are shown in Table 1. This table shows that even the "worst" pyUPMASK performer (VOR) gives better metrics than UPMASK 66% of the times. The method with the highest pyUPMASK win percentage (GMM) outperforms UPMASK 83% of the times; a massive advantage. The worst individual metric result is obtained for TPR_9 in the MBK method. Still the value is larger than 50% which means that the majority of the cases were better handled by pyUPMASK. On the other end of the analysis the best metric result is found for HMS in the GMM method, where pyUPMASK manages to outperform UPMASK for almost all of the cases.

An important aspect other than the performance measured by the statistical metrics is the performance measured in computing time. This is shown in Fig. 9 as a bar plot normalized to the total time used by UPMASK to process the 600 synthetic clusters. The numbers on top of the bars displays how many times faster each clustering method in pyUPMASK is compared to UPMASK. The fastest method is expectedly a single-run method, KNN, which performs 170 times faster than UPMASK. This is an enormous margin of difference. Even the slowest method, GMM, is faster than UPMASK: it manages to process the set of synthetic cluster employing almost 38% less time than UPMASK, or 1.6 times faster. On average we can say that pyUPMASK using a single-run method is over 100 times faster than UPMASK, and more than 3 times faster for the multiple run methods.

The choice between which clustering method to employ in pyUPMASK will then depend on the specific requirements of the analysis. If the absolute best performance measured by a classification metric is sought after, then clearly GMM is the method to

chose (with the added advantage of being faster than UPMASK). If we can trade some performance for a faster process, then KMS or MBK can be used. And if we are willing to trade even more classification performance (while still performing much better than UPMASK), then VOR, KNN, or AGG are by far the fastest approaches.

4.1. Computational resources

Finally, we consider the issue of computational resources requirements. We have found that for very large input data files the memory and processing power requirements can be too much for most methods to handle. Although the VOR clustering method is the worst performer out of the six tested methods (measured by classification metrics) it has an advantage compared to all the others, including UPMASK, when it comes to analyzing large files. To obtain the Voronoi diagram of an N-dimensional set of points, Python's `scipy` package relies on the Qhull library (Barber et al. 1996)[13]. This library is written in the C language which makes it extremely efficient, thus making pyUPMASK's VOR method very efficient for large datasets.

To test this we downloaded a large 6×6 deg region around the NGC2516 cluster from Gaia's second data release. The resulting field contains over 420000 stars up to a maximum magnitude of $G = 19$ mag. This limit was imposed because beyond this value the photometric errors grow exponentially. The frame was processed with the six tested pyUPMASK clustering methods and UPMASK, using proper motions and parallax as input data. We used 25 outer loop iterations for all the methods, except of course for the single-run methods, and a value of 25 for the parameter that determines the number of elements per cluster (i.e., the default values for both parameters as explained in Sect. 3.3).

Only three methods were able to complete the process: VOR, KNN, and MBK. The methods AGG, KMS, and GMM failed due to memory requirements as they attempted to allocate arrays of ~640 Gb, ~31 Gb, and ~31 Gb on memory, respectively. UPMASK was not able to finish even the first iteration of the inner loop within the first iteration of the outer loop after a full week of running, so it was halted. The results of the VOR, KNN, and MBK methods can be seen in a color-magnitude diagram (CMD) in Fig. 10. We plotted the 1500 stars with larger membership probabilities given by each method, as this is the approximate number of cluster members in the frame (given by a simple stellar density analysis). It is evident that the VOR method returns the most reasonable and less contaminated CMD out of the three. Furthermore, this method managed to process the cluster almost 4 and 40 times faster than KNN and MBK, respectively. It is worth noting that in a personal computer, which has far less resources than a computational cluster, VOR was the only method able to run.

A smaller field containing this same cluster was analyzed with UPMASK in Cantat-Gaudin et al. (2018). The processed area contains only ~1100 stars associated to this cluster up to a magnitude of G=18 mag. The analysis done in this work resulted in less than 800 stars with membership probabilities (MPs) above 0.5, and ~100 stars with MPs> 0.9. In contrast, using the same magnitude cut, we are able to obtain with the VOR method on our very large field over 1700 stars with MPs> 0.99 tracing a well defined sequence. The advantage of studying a cluster using almost all of its members versus using less than 10% of them (comparing the large MPs subsets), is obvious.

The VOR method is thus the only one that was able to produce quality results for this very large dataset, and it did so while using the least amount of processing time by a wide margin.

5. Conclusions

In this work we introduced pyUPMASK, a tool based on the general algorithm presented in KMM14 for the UPMASK code with several added enhancements. The primary aim of pyUPMASK is the assignment of membership probabilities to cluster stars observed in a field contaminated by field stars. We tested our code extensively using six hundred synthetic clusters affected by a large range of contamination. Six performance metrics were employed, three of them in two different configurations, to ensure sufficient coverage when assessing statistical classification. Results from six different clustering methods in pyUPMASK were compared to those from UPMASK. The pyUPMASK tool proved to outperform UPMASK under every metric and with every clustering method used, while still managing to be faster (and, for the single-run methods, extremely faster).

This new tool is thus more configurable (around a dozen clustering algorithms supported), faster, and a much better performant than UPMASK. pyUPMASK is fully written in Python and it is made available for its use under a GPL v3 general public license[14].

Acknowledgements. The authors would like to thank Dr C. Anagnostopoulos for his help with the `hmeasure` (https://github.com/cran/hmeasure) R package. This research has made use of NASA's Astrophysics Data System. This research made use of the Python language (van Rossum 1995) and the following packages: NumPy[15] (Van Der Walt et al. 2011); SciPy[16] (Jones et al. 2001); Astropy[17], a community-developed core Python package for Astronomy (Astropy Collaboration et al. 2013; Price-Whelan et al. 2018); scikit-learn[18] (Pedregosa et al. 2011b); matplotlib[19] (Hunter et al. 2007). This work has made use of data from the European Space Agency (ESA) mission *Gaia* (https://www.cosmos.esa.int/gaia), processed by the *Gaia* Data Processing and Analysis Consortium (DPAC, https://www.cosmos.esa.int/web/gaia/dpac/consortium). Funding for the DPAC has been provided by national institutions, in particular the institutions participating in the *Gaia* Multilateral Agreement.

References

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33
Baddeley, A., Rubak, E., & Turner, R. 2015, Spatial Point Patterns: Methodology and Applications with R, Chapman & Hall/CRC Interdisciplinary Statistics (CRC Press)
Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. 1996, ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE, 22, 469
Baxter, R. A. 2010, Mixture Model, ed. C. Sammut & G. I. Webb (Boston, MA: Springer US), 680–682
Brier, G. W. 1950, Monthly Weather Review, 78, 1
Cantat-Gaudin, T., Jordi, C., Vallenari, A., et al. 2018, A&A, 618, A93
Dempster, A. P., Laird, N. M., & Rubin, D. B. 1977, Journal of the Royal Statistical Society: Series B (Methodological), 39, 1
Dixon, P. M. 2014, Ripley's K Function (American Cancer Society)
Good, I. J. 1952, Journal of the Royal Statistical Society. Series B (Methodological), 14, 107

[13] http://www.qhull.org/

[14] https://www.gnu.org/copyleft/gpl.html
[15] http://www.numpy.org/
[16] http://www.scipy.org/
[17] http://www.astropy.org/
[18] http://scikit-learn.org/
[19] http://matplotlib.org/

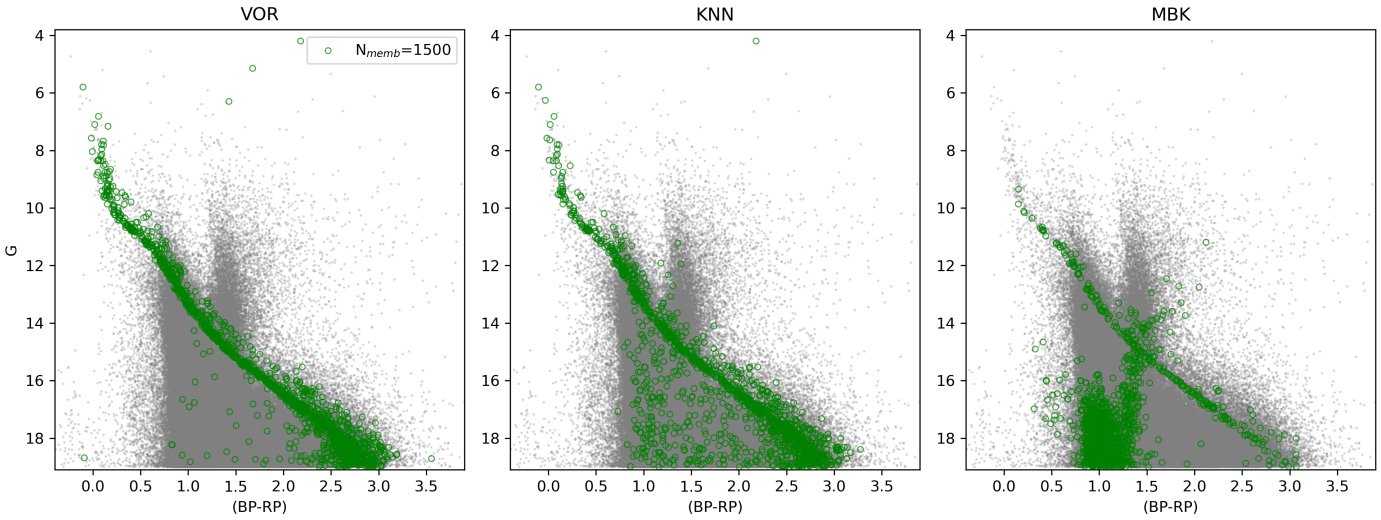

Fig. 10. Results for the NGC2516 cluster by the VOR (left), KNN (center), and MBK (right) methods. Estimated members are shown as green circles, field stars are shown as gray dots.

Hand, D. & Anagnostopoulos, C. 2014, Pattern Recognition Letters, 40, 41
Hand, D. J. 2009, Machine Learning, 77, 103
Hernández-Orallo, J., Flach, P., & Ferri, C. 2012, Journal of Machine Learning Research, 13, 2813
Hunter, J. D. et al. 2007, Computing in science and engineering, 9, 90
Jones, E., Oliphant, T., Peterson, P., et al. 2001, SciPy: Open source scientific tools for Python, [Online; accessed 2016-06-21]
Krone-Martins, A. & Moitinho, A. 2014, A&A, 561, A57
Lagache, T., Lang, G., Sauvonnet, N., & Olivo-Marin, J.-C. 2013, PLoS ONE, 8, e80914
Lobo, J. M., Jiménez-Valverde, A., & Real, R. 2008, Global Ecology and Biogeography, 17, 145
MacQueen, J. 1967, in Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics (Berkeley, Calif.: University of California Press), 281–297
Marcon, E., Traissac, S., & Lang, G. 2013, ISRN Ecology, 2013, 1
Matthews, B. 1975, Biochimica et Biophysica Acta (BBA) - Protein Structure, 405, 442
Merkle, E. C. & Steyvers, M. 2013, Decision Analysis, 10, 292
Momcheva, I. & Tollerud, E. 2015, arXiv e-prints, arXiv:1507.03989
Parker, C. 2011, in 2011 IEEE 11th International Conference on Data Mining (IEEE)
Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011a, Journal of Machine Learning Research, 12, 2825
Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011b, Journal of Machine Learning Research, 12, 2825
Perren, G. I., Vázquez, R. A., & Piatti, A. E. 2015, A&A, 576, A6
Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., et al. 2018, AJ, 156, 123
Ripley, B. D. 1976, Journal of Applied Probability, 13, 255266
Ripley, B. D. 1979, Journal of the Royal Statistical Society. Series B (Methodological), 41, 368
Rodriguez, A. & Laio, A. 2014, Science, 344, 1492
Sanders, W. L. 1971, A&A, 14, 226
Sculley, D. 2010, in Proceedings of the 19th International Conference on World Wide Web, WWW '10 (New York, NY, USA: Association for Computing Machinery), 11771178
Streib, K. & Davis, J. W. 2011, in CVPR 2011, 2305–2312
Tollerud, E. J., Smith, A. M., Price-Whelan, A., et al. 2019, Bulletin of the American Astronomical Society, 51, 180
Van Der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, Computing in Science & Engineering, 13, 22
van Rossum, G. 1995, Python tutorial, Report CS-R9526, pub-CWI, pub-CWI:adr
Vasilevskis, S., Klemola, A., & Preston, G. 1958, The Astronomical Journal, 63, 387
Voronoi, G. 1908, Journal für die reine und angewandte Mathematik, 1908, 97
Zepeda-Mendoza, M. L. & Resendis-Antonio, O. 2013, Hierarchical Agglomerative Clustering, ed. W. Dubitzky, O. Wolkenhauer, K.-H. Cho, & H. Yokota (New York, NY: Springer New York), 886–887