



Robust Assessment of Clustering Methods for Fast Radio Transient Candidates

Kshitij Aggarwal^{1,2} , Sarah Burke-Spolaor^{1,2,3} , Casey J. Law⁴ , Geoffrey C. Bower⁵ , Bryan J. Butler⁶ , Paul B. Demorest⁶ , T. Joseph W. Lazio⁷, Justin Linford⁶ , Jessica Sydnor^{1,2} , and Reshma Anna-Thomas^{1,2} 

¹ West Virginia University, Department of Physics and Astronomy, P.O. Box 6315, Morgantown, WV, USA; ka0064@mix.wvu.edu

² Center for Gravitational Waves and Cosmology, West Virginia University, Chestnut Ridge Research Building, Morgantown, WV, USA

³ CIFAR Azrieli Global Scholars Program, CIFAR, Toronto, Canada

⁴ Cahill Center for Astronomy and Astrophysics, MC 249-17 California Institute of Technology, Pasadena, CA 91125, USA

⁵ Academia Sinica Institute of Astronomy and Astrophysics, 645 N. A'ohoku Place, Hilo, HI 96720, USA

⁶ National Radio Astronomy Observatory, Socorro, NM, 87801, USA

⁷ Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, M/S 67-201, Pasadena, CA 91109, USA

Received 2021 March 23; revised 2021 April 13; accepted 2021 April 16; published 2021 June 15

Abstract

Fast radio transient search algorithms identify signals of interest by iterating and applying a threshold on a set of matched filters. These filters are defined by properties of the transient such as time and dispersion. A real transient can trigger hundreds of search trials, each of which has to be post-processed for visualization and classification tasks. In this paper, we have explored a range of unsupervised clustering algorithms to cluster these redundant candidate detections. We demonstrate this for REALFAST, the commensal fast-transient search system at the Karl G. Jansky Very Large Array. We use four features for clustering: sky position (l, m), time, and dispersion measure (DM). We develop a custom performance metric that makes sure that the candidates are clustered into a *small* number of *pure* clusters, i.e., clusters with either astrophysical or noise candidates. We then use this performance metric to compare eight different clustering algorithms. We show that using sky location along with DM/time improves clustering performance by $\sim 10\%$ as compared to the traditional DM/time-based clustering. Therefore, positional information should be used during clustering if it can be made available. We conduct several tests to compare the performance and generalizability of clustering algorithms to other transient data sets and propose a strategy that can be used to choose an algorithm. Our performance metric and clustering strategy can be easily extended to different single-pulse search pipelines and other astronomy and non-astronomy-based applications.

Unified Astronomy Thesaurus concepts: [Clustering \(1908\)](#); [Random Forests \(1935\)](#); [Radio transient sources \(2008\)](#); [Radio interferometry \(1346\)](#); [Extragalactic radio sources \(508\)](#); [Radio bursts \(1339\)](#); [Very Large Array \(1766\)](#)

1. Introduction

One of the significant difficulties when seeking fast-transient radio signals is the load of candidates that results from a transient search: it is common for a search algorithm to return millions to billions of candidates from a survey, only a few of which end up being genuine (the rest being thermal noise and radio-frequency interference (RFI)). Even one bright event, whether astrophysical or artificial, can generate many hundreds of separate candidates. This is because search algorithms iterate over a set of matched filters and identify transients that exceed the detection threshold. Clustering algorithms to account for this effect are of dire importance to any radio transient search pipeline. A rigorous study of an effective clustering algorithm for fast radio transient searches is the primary purpose of the study reported here.

To understand this paper's context, it is important to review the main procedural components of a typical search for fast radio transients. The term *fast* here specifically refers to transients for which the dispersion delay, caused by astrophysical plasma, is non-negligible and must be accounted for to optimize search sensitivity. The tenuous plasma that fills the space between stars, around galaxies, between galaxies, and elsewhere can have a strong influence on radio signals. The most prominent influence they have is inducing a frequency-dependent pulse sweep caused by a frequency-dependent refractive index of cold astrophysical plasma (Lorimer & Kramer 2004). The magnitude of this dispersive time delay for a pulse is quantified by the *dispersion measure* (DM).

When searching for a radio transient, neither its DM nor width (i.e., duration) are known. Therefore, one must search over a range of DMs and widths to carry out a full-sensitivity search. DM values at which to search are chosen by considering the expected decline in signal-to-noise ratio (S/N), due to pulse broadening, at the adjacent DMs (Cordes & McLaughlin 2003; Levin 2012). To summarize, a standard fast-transient-search pipeline dedisperses the data at various trial DMs, averaging all the frequencies to obtain a one-dimensional time series, followed by convolution using boxcar filters of various widths. Candidate pulses are identified by searching for peaks above a predecided threshold, with the S/N of a candidate determined from an estimate of the signal strength with respect to the standard deviation within the region defined by the boxcar filter width. Dedispersing at an incorrect DM, or using a boxcar filter of incorrect width, would reduce the S/N of the pulse. As previously noted, any event can lead to multiple candidates being detected by the search pipeline if the S/N remains above the threshold at the incorrect DM or boxcar width.

This process can lead to a substantial number of redundant candidates caused by a single event. Clustering is performed on these candidates to automatically combine such events at the end of the search pipeline. Some algorithms that are currently in common use are friends of friends and Density Based Spatial Clustering of Applications with Noise (DBSCAN; Ester et al. 1996; Deneva et al. 2009; Barsdell 2012). However, few clustering algorithms have been rigorously tested.

Throughout this work, our primary motivation was to identify the optimal clustering algorithm for single-pulse searches, in particular in searches for fast radio bursts (FRBs). FRBs are bright, millisecond-duration bursts of energy of extragalactic origin (Lorimer et al. 2007). Over 150 such sources have been seen so far (Petroff et al. 2016), and many radio telescopes worldwide (both single dish and interferometric) are now or will soon be outfitted with specialized hardware and software to carry out FRB searches.

The REALFAST system at the Karl G. Jansky Very Large Array (VLA) is one such commensal fast-transient search system (Law et al. 2018). It is currently the only real-time coherent-imaging interferometric search system, although because of the importance of precise FRB localization, a number of similar systems are in operation, commissioning, or planning phases (Bannister et al. 2019; Kocz et al. 2019; Michilli et al. 2021; Leung et al. 2021). REALFAST forms thousands of de-dispersed images every second to search for pulses in the image plane and can localize every FRB it detects to arcsecond precision (Law et al. 2018).

The prototype REALFAST system was used for the first localization of an FRB (Chatterjee et al. 2017). In its first year of commensal observation at the L (1–2 GHz), S (2–4 GHz), and C band (4–8 GHz), REALFAST detected five FRBs (S. Bhandari et al. 2021, in preparation; S. Tendulkar et al. 2021, in preparation; Aggarwal et al. 2020; Law et al. 2020, 2021). The REALFAST pipeline focuses on searching for transients at multiple DMs and trial widths, each of which is then post-processed. A total intensity (Stokes I) image is formed for each trial DM and width. Point sources in these images with an S/N greater than a preset threshold trigger the detection pipeline. The data corresponding to each candidate are then saved to disk and classified using a deep learning-based classifier (Agarwal et al. 2020). Visualizations that show the radio image, spectrogram, spectra, and profile of the candidate are then generated. These visualizations also consist of other relevant candidate parameters: S/N, DM, width, relative sky position with respect to the pointing center, scan number, etc. and are used for follow-up inspection.

In this work, we use REALFAST data as a test case to explore and compare candidate-clustering techniques. We also generalize our results to apply to single-dish telescopes, which do not have spatial (sky location) information to use in a clustering algorithm. This paper is laid out as follows: In Section 2, we provide a more detailed motivation for the need for clustering and a discussion of clustering methodologies. Section 3 describes the data used for testing the algorithm, followed by methods explained in Section 4. The results of the analysis are presented in Section 5, followed by a discussion and conclusion in Sections 6 and 7, respectively.

2. Clustering

As mentioned previously, clustering is implemented between the search and the candidate processing steps of the pipeline. In the REALFAST system, after clustering, we choose the maximum S/N candidate from each cluster, and only those are analyzed in the candidate processing step. We also consider all the unclustered candidates as individual clusters of size one and pass them onward for processing.

In this section, we discuss the need to use clustering in the context of a single-pulse search pipeline. Further, we use the following terminology throughout this paper:

1. Event: The actual physical occurrence of an astrophysical transient (e.g., FRB, pulsar) or RFI.
2. Candidate: A single detection reported by a search pipeline. It typically consists of a set of properties (sky location, DM, time, etc). Candidates may be random thermal noise or associated with an event. Multiple candidates can be associated to a single event.
3. Observation: A set of candidates generated after the search pipeline is run on some data. It can be real or simulated and can have candidates associated with FRB or RFI or both.
4. Data set: A set of observations.
5. Cluster: A group of candidates (or members) with the same labels assigned by a clustering algorithm.
6. Member: Candidates within a cluster.
7. True labels: Each member of a cluster is associated with an event. We refer to this event as the true label of that member.
8. Real/FRB/transient: Event, cluster, or member associated with an astrophysical transient.
9. RFI: Event, cluster, or member that is not astrophysical.

2.1. Expected Number of Candidates from a Single Astrophysical Event

As explained in the previous section, the following search parameters are reported for each candidate detected by the REALFAST search pipeline: DM, time of occurrence of the candidate, relative sky position with respect to the pointing center (l , m), S/N, and width. Moreover, for each event, the pipeline can return multiple candidates at nearby (incorrect) values of DM, width, and time. The observed S/N of a candidate detected at a trial DM, width, and sky location is given by (assuming no other losses, etc.)

$$S/N_{\text{observed}} = S/N_{\text{optimal}} \cdot F_{\text{widthloss}} \cdot F_{\text{beamloss}} \cdot F_{\text{DMloss}}, \quad (1)$$

where S/N_{optimal} is the S/N of the candidate when there is no loss. $F_{\text{widthloss}}$ is the loss due to incorrect boxcar width (Cordes & McLaughlin 2003), F_{beamloss} is the loss due to position of candidate within the primary beam of the telescope, and F_{DMloss} is the loss due to an incorrect DM value (Levin 2012 Section 2.3).

Using this equation, we can calculate the number of candidates that will trigger a single-pulse search system for an astrophysical event. For instance, we assume a VLA L-band (1–2 GHz) configuration with 256 frequency channels and a time resolution of 5 ms, and that REALFAST system is used to search for transients. We then search for DMs from 0–3000 pc cm⁻³ and set $F_{\text{dmloss}} = 0.95$, i.e., up to a maximum of a 5% loss in sensitivity between DM trials. Using this, and assuming an intrinsic pulse width of 30 ms, we can compute the DM array (Levin 2012, Section 2.3). We also assume $t_{\text{scatt}} = 0$ as it is line-of-sight dependent and is typically small. We set our boxcar search widths to 5, 10, 20, and 40 ms and the S/N detection threshold at 8. Figure 1 shows the number of candidates detected with respect to input S/N of the transient for two different values of transient DMs and widths. Here, we have also assumed that the candidate is at the center of the beam, therefore the F_{beamloss} is 1. This figure clearly shows that the number of candidates detected by the pipeline can be large even for one event. This can overwhelm the real-time systems that are responsible for post-processing these candidates and writing their data to disk, hence motivating the use of clustering algorithms.

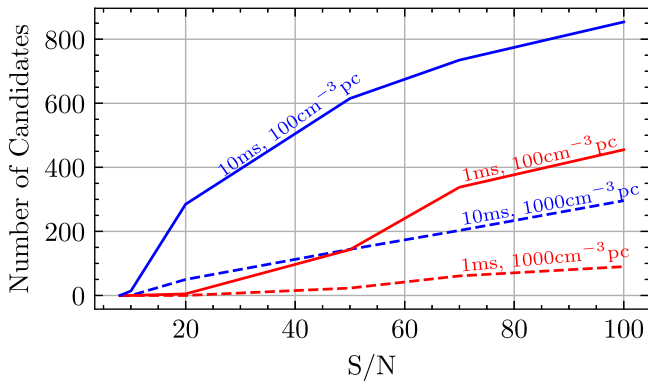


Figure 1. The number of candidates generated by a single-pulse search pipeline for events with varying S/N. Different colors represent different intrinsic widths of the transient, and different line styles show different DMs. Observing and search configuration similar to that of REALFAST at the L band was chosen (sampling time: 5 ms, number of frequency channels: 256, bandwidth: 1 GHz, DM range: 0–3,000 pc cm^{-3} , boxcar widths: [5, 10, 20, and 40 ms], S/N threshold: 8). See Section 2.1 for more details.

2.2. Unsupervised Clustering

In this paper, we have taken the approach of unsupervised clustering. Here, we briefly discuss unsupervised clustering and some of its caveats. Unsupervised clustering is the method used to find meaningful clusters from an unlabeled data set, i.e., a priori information about the number of clusters and the true cluster assigned for each candidate is not known (Jain et al. 1999). This is in contrast to supervised clustering for which this information is available. In this analysis, we do not know the clustering information of the candidates, therefore we opted for unsupervised clustering.

Unsupervised clustering is typically done by estimating the *distance* or *similarity* between different candidates, with the aim that candidates with low distance might be similar and belong to the same cluster. The clustering algorithms we discuss in this paper use standard search pipeline features, without any expensive preprocessing, and can find reliable clusters in real time. In some cases, physically meaningful relations between features could also be computed to enhance clustering performance, however we do not employ these here (Pang et al. 2018, Section 3.2).

The caveat to this approach is that unsupervised clustering techniques can be harder than supervised methods to tune for specific data sets. Also, due to the lack of true labels, it is difficult to evaluate the clustering performance. We discuss the above caveats further in Section 4.

2.3. Clustering RFI

Strong RFI events can also overwhelm the real-time pipelines by generating a large number of candidates, sometimes at all DM trials. Even though we use multiple filtration techniques to mitigate RFI, some signals still reach the pipeline’s clustering step. Narrow-band RFI can lead to many candidates at all DMs in the DM grid (and because of the resulting time shift of the peak, corresponding time bins). In some cases, the RFI appears as a strong localized source in the radio image and hence is present as a dense cluster of points in the image plane (for instance, this can happen with a sufficiently high-altitude satellite). Therefore, we can leverage the clustering algorithm to cluster those thousands of triggers into one cluster, reducing the computational load by orders of magnitude. As it is not feasible to manually label RFI

examples into multiple clusters, we cannot evaluate clustering algorithms’ performance on identifying separate RFI clusters. Instead, we only estimate clustering performance on FRB clusters. This is further discussed in Section 4.3.

3. Data

Here, we describe the details of our data set and the features we used for clustering. We used REALFAST data to generate a data set containing representative RFI and used simulated FRBs to generate a data set with representative candidates. We then combined the two data sets and applied four different preprocessing techniques (downsampling and normalization) on the features of the candidates to simulate 250 observations, consisting of candidates from both real and RFI events.

3.1. Feature Selection for Clustering

As mentioned in Section 1, the pipeline reports a set of measured parameters for each candidate that satisfies the S/N threshold criterion. For our clustering analysis, we cluster based on DM, time (as is the standard with most past FRB searches), and sky position (with relative direction cosines represented by l and m as angular distances from the observation’s pointing center) of the candidates.

Candidates associated with an FRB event are expected to be densely located in l and m , as the FRB originates from a specific location in the sky. They would also show an expected S/N decrease in adjacent DM and time values (see Section 2.1) and would be closer for those parameters. On the other hand, RFI is randomly spread across this parameter space, but strong RFI can show a trend in DM and time if detected at multiple DMs.

3.2. RFI Database

In this analysis, we used data from various commensal and commissioning observations of the REALFAST system (project codes: 19A-242, 20A-330, and 19B-223, 20A-163), in which the standard REALFAST pipeline detected only RFI candidates. This data spans a range of array configurations and other observing and search parameters (frequency, bandwidth, image pixel size, etc.). To create candidates representative of the real-time pipeline, we reran the REALFAST transient search on this data with the pipeline using default search parameters (Law et al. 2018, 2020). As clustering performance is expected to be sensitive to the RFI environment, we selectively chose data sets with a variety of RFI types. These data sets are representative of the broad range of RFI we have seen at VLA and therefore form a robust sample of RFI for our analysis. This procedure was used to generate RFI candidates from 13 observations, with a few to ~ 6000 candidates each. We manually verified that all these candidates were RFI and saved parameters relevant for clustering for each candidate (Section 3.1). We will refer to this as the RFI data set.

3.3. Simulating and Injecting FRBs

We also generated a data set of *real* candidates, representing our signals of interest. This was done by generating simulated data (with standard radiometer noise for different array configurations, observing, and search parameters) and injecting simulated transients. The distribution of parameters used for injecting transients is given in Table 1. We searched this

Table 1
Parameter Distributions of Simulated FRBs

Parameter	Distribution	Range/Values
S/N	Uniform	10, 40
DM (pc cm ⁻³)	Uniform	10, max_search_dm ^a
Width (ms)	Uniform	1, 40 (ms)
Frequency	Uniform	L, S, C, X
Array configuration	Uniform	A, B, C, D ^b
Sky position (l, m)	Uniform	$-\text{fov}/2, \text{fov}/2^c$

Notes.

^a max_search_dm is the maximum DM searched for a given configuration.

^b Maximum baseline lengths for the four configurations (A, B, C, and D) are 36.4, 11.1, 3.4, and 1.03 km.

^c fov is the field of view at the randomly chosen frequency.

simulated data using the REALFAST system with a real-time search configuration to generate candidates. We saved the relevant parameters of all the candidates, and manually verified them to make sure that each candidate corresponds to the injected transient. We discarded any observation with less than four candidates. This procedure was used to generate real candidates from 114 simulated observations (with one transient injected in each observation). We will refer to this as the FRB data set.

3.4. Test Data Set

To evaluate the performance of various clustering methods (described in Section 4.1), we generated a test data set. We used this data set to compare the performance of different preprocessing techniques and also during hyperparameter tuning (see Section 4.2).

The test data set consists of multiple observations, each containing some RFI candidates and some FRB candidates. We enforce that each observation has one transient event, and so all FRB candidates in an observation would be associated to that single event. Therefore, a perfect clustering algorithm should form only one FRB cluster per observation. To generate such an observation, we randomly chose one observation each from the RFI and FRB data set pool. We then randomly select $X\%$ of RFI candidates (where X is sampled from a uniform distribution between 20 and 100) from the RFI observation, all the FRB candidates from the FRB observation, and concatenate their features. We then randomize the order of the examples. This creates a set consisting of both RFI and FRB candidates. All observations with less than 10 total number of candidates were discarded. Using this process, we created 250 observations containing RFI and FRB candidates, which formed our test data set.

3.5. Preprocessing

Preprocessing is the procedure that takes the event features and converts them into indexed parameter ranges such that all the parameters will be equally weighted in terms of their importance in the clustering.

As explained in Section 3.1, we use DM, time, l , and m as features for clustering. Therefore, for each candidate in each observation of our database, we save these four parameters along with the imaged S/N of the candidate. After clustering, we use the S/N to decide the representative candidate from

each cluster. We convert the absolute value of DM to an index based on its index in the DM array for each candidate. Similarly, we also convert the time value (in seconds) to an index based on the sample number corresponding to that time from the start of that processing segment. We also convert l and m (which is the offset of the candidate from primary beam center) to corresponding pixel values, using the synthesized beam size. Therefore, we convert all the features to corresponding indices. This is necessary as otherwise, different scales of different features might bias the distance estimates required in clustering.

The transient events we are interested in appear as a point source in the sky. Therefore, all the candidates from that transient should constitute a small range of l and m index values. We downsampled the l and m values of all the candidates to increase the sky density of candidates, which might enhance the clustering performance. We tried downsampling factors of 1, 2, and 4 (henceforth referred to as DS1, DS2, and DS4, respectively). Although we have scaled all the features to their corresponding indices, we also evaluated clustering performance on standardized data (i.e., with zero mean and unit variance, hereafter referred to as Norm). Throughout the paper, we report the performance of all the algorithms on all these different preprocessing techniques. We also try to determine the preprocessing technique that leads to the best clustering performance.

4. Methods

4.1. Clustering Algorithms

We compare eight algorithms to cluster our test data set: K-Means, Mean Shift, Affinity Propagation, Agglomerative Clustering, DBSCAN, Ordering Points To Identify the Clustering Structure (OPTICS), Hierarchical Density Based Spatial Clustering of Applications with Noise (HDBSCAN), and Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH). For all except HDBSCAN, we use the implementation of these algorithms in `scikit-learn` (Pedregosa et al. 2011; Buitinck et al. 2013). We use the Python implementation of HDBSCAN by McInnes et al. (2017). We briefly discuss the details of these algorithms and their hyperparameters in Appendix A. We refer the reader to the respective papers and `scikit-learn` documentation for more details.

4.2. Hyperparameter Tuning

Each clustering algorithm has several input parameters that can be used to control the algorithm's clustering process and speed. These input parameters are called hyperparameters. Some algorithms are very sensitive to the choice of these hyperparameters, while others are robust to a range of hyperparameters. Our aim is to find the hyperparameters for each algorithm that lead to the best clustering performance (called optimal hyperparameters). The following three techniques are typically used to obtain the optimal hyperparameters: brute force grid search, random sampling, and Bayesian optimization. Grid search involves generating a grid of points covering the whole parameter space uniformly. The performance metric is then calculated on all the grid points, and the hyperparameter combination with the maximum value of the metric is chosen. In random sampling, the hyperparameter combinations are randomly chosen from a distribution of parameters. Bayesian optimization uses Bayesian

techniques to parse the parameter space and obtain the optimal hyperparameter combination.

Random sampling has been shown to be better than brute force grid search (Bergstra & Bengio 2012). This is because, in most cases, only a few hyperparameters really matter, the importance of which changes with different data sets. This makes grid search a poor choice for searching for hyperparameters for different data sets. Therefore, we opted to use random sampling. Also, because our parameter space is not very large, we decided not to use Bayesian optimization. Table 3 in Appendix B shows the ranges and various possible values of different hyperparameters that were tried for each algorithm. Wherever necessary, we used a random state of 1996 in the algorithms for reproducibility.

4.3. Performance Metric

The general idea of using a performance metric is to have a common reference point to rank the effectiveness of clustering algorithms (and their hyperparameters). The one with the maximum value of the metric has the best general performance.

Critical to this idea is a clear statement of our goals. Our primary measurable goals with clustering are the following:

1. *Avoid missing a genuine event due to clustering.* This can happen due to overaggressive clustering that identifies FRB candidates as false members of an RFI cluster. As only the highest S/N member from each cluster is processed further, assigning FRB members to RFI clusters will lead to an FRB candidate not passing further in the pipeline. This can happen if there are RFI candidates with an S/N higher than that of the FRB candidates.
2. *Each event of interest should be singly identified.* All candidates from one FRB event should be clustered into one cluster, rather than many small separate clusters representing a single event of interest. This is to minimize the number of candidates that are passed to the pipeline for post-processing and classification.

To represent these goals, we have developed the following metric. A higher value of the metric is favorable. We use *homogeneity*, *completeness*, *v measure*, and *recall* to calculate the metric (hereafter referred to as *score*). In the following, an FRB cluster is defined as a cluster containing one or more FRB candidates, obtained after clustering. In the following discussion, we follow the terminology defined in Section 2.

4.3.1. Homogeneity

Homogeneity is the measure of purity of the clusters with respect to true labels, i.e., it estimates if each cluster contains only members of a single class (i.e., either FRB or RFI). We calculate homogeneity for each observation in the test data set. As we are primarily interested in performance on FRB candidates, and as RFI can be clustered into multiple clusters (for which we do not have true information), we define homogeneity only for FRB clusters in the observation. For each FRB cluster, we calculate the ratio of the number of FRB candidates in that cluster to total number of candidates in that cluster. Homogeneity (h) is the weighted average of all these ratios, weighing them by the number of candidates in the

cluster. Hence,

$$\begin{aligned} h &= \frac{1}{N_T} \sum_i \frac{n_{\text{FRB}}^i}{n_T^i} n_T^i \\ &= \frac{1}{N_T} \sum_i n_{\text{FRB}}^i, \end{aligned} \quad (2)$$

where i represents the i th FRB cluster, and the sum is over all the FRB clusters. n_{FRB}^i is the number of FRBs in the i th FRB cluster, and n_T^i is the total number of candidates in that cluster. $N_T = \sum_i n_T^i$ is the total number of candidates in all FRB clusters. h can be between 0 (when all FRB candidates are left unclustered) and 1 (when all FRB clusters contain only FRB candidates).

4.3.2. Completeness

Completeness is used to estimate if all members of a given class are assigned to the same cluster. We calculate completeness for each observation in the test data set. We define completeness for FRB clusters, and a high completeness score will minimize the number of clusters the FRB candidates are clustered to. For each FRB cluster, we calculate the ratio of the number of FRB candidates in that cluster to the total number of FRB candidates. Completeness (c) is equal to the weighted average of all these ratios, weighing them by the number of candidates in the cluster.⁸ Hence,

$$c = \frac{1}{N_T} \sum_i \frac{n_{\text{FRB}}^i}{N_{\text{FRB}}} n_T^i, \quad (3)$$

where i represents the i th FRB cluster, and the sum is over all the FRB clusters. n_{FRB}^i is the number of FRBs in the i th FRB cluster, and N_{FRB} is the total number of FRB candidates in that observation. $N_T = \sum_i n_T^i$ is the total number of candidates in all FRB clusters. c will be very small if all FRB candidates are assigned different clusters and 1 when all FRB candidates are clustered into one cluster.

4.3.3. V measure

V measure is the harmonic mean between homogeneity and completeness. This is used as we want all the clusters to be pure and favor the minimum number of clusters. Therefore, we want to maximize both homogeneity and completeness. V measure (v) will be 1 when both h and c are 1 and will be 0 if either of those is 0. Hence,

$$v = \frac{2hc}{h+c}. \quad (4)$$

We calculate h , c , and v for each observation in the test data set and take a weighted average of all v 's (weighting by the total number of candidates in that observation) to get an estimate of V measure for the whole data set (V).

4.3.4. Recall

Recall is the fraction of FRBs that are recovered after clustering. After clustering, only the candidates with maximum S/N in each cluster are processed further in the pipeline. Therefore, if the clustering algorithm clusters FRB candidates

⁸ We include unclustered candidates as a single cluster in this case, while unclustered candidates were ignored while calculating homogeneity.

together with high S/N RFI candidates, then the FRB will not be recovered from that cluster and might be missed. Therefore, recall (R) is defined as the ratio of the number of observations in the data set for which FRB was recovered to the total number of observations in the data set.

4.3.5. Score

Score is defined as the product of recall (R) and total V measure (V).

$$\text{Score} = R \times V. \quad (5)$$

We use this score to compare the clustering performance of different algorithms and find the optimal hyperparameters for each clustering algorithm.

4.4. Advantages of This Metric

We have defined the above metric with respect to FRB and RFI clusters, but it can be easily generalized to any application with goals generally similar to those laid out in Section 4.3. This metric ensures that the information in relevant clusters is not missed by overaggressive clustering while still minimizing the number of clusters formed. There are some advantages of this metric over other clustering metrics available in the literature (for a detailed comparison using a similar metric see Rosenberg & Hirschberg 2007): (1) It is independent of the clustering algorithm, size of the data set, number of classes, and clusters. (2) It can appropriately use one (or more) base class (here FRB) to evaluate the clustering performance with respect to true labels, considering all data points of the relevant class. (3) Using homogeneity and completeness, V measure gives importance to both pure clusters and the minimum number of clusters. (4) By adequately weighting individual metrics, it is possible to concisely evaluate the clustering performance across multiple examples, in the form of a simple number (score).

5. Results

5.1. Optimal Hyperparameters

We show the maximum score obtained for each algorithm after hyperparameter search in Figure 2. Henceforth in this paper, we will refer to these hyperparameters as *optimal hyperparameters*. This figure shows that Mean Shift has the maximum score, out of all the eight algorithms. Table 2 shows these optimal hyperparameters for each algorithm.

Figure 3 shows the distributions of scores at various hyperparameter values for each combination of algorithm and preprocessing. Although not crucial to hyperparameter selection, the distribution of scores gives an insight into the robustness of the algorithm to the choice of input hyperparameters. In some cases, the scores vary between the full range of 0 and 1, while in others, the distribution is very narrow around high scores. The algorithms for which the score distributions peak around a high value should be more robust to the input hyperparameters than those for which the peak is at a middle or even low value of score. In the latter case, only a small range of hyperparameters would lead to a high score.

5.2. Effect of Data Processing

As mentioned in Section 3.5, we also repeated the above experiment after preprocessing the data in two ways:

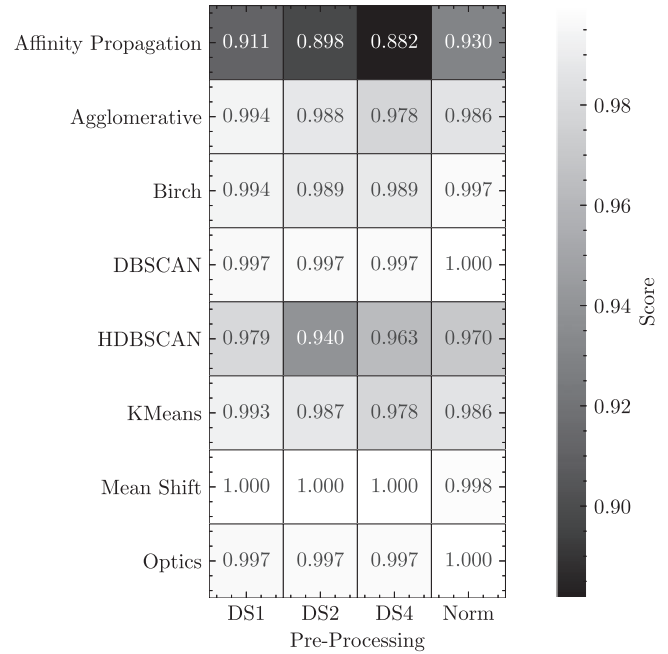


Figure 2. Maximum score obtained after random hyperparameter search (at the optimal hyperparameters) for each algorithm and preprocessing combination. Optimal hyperparameters for each case are given in Table 2. DS refers to downsampling applied to the l and m indices. Norm refers to normalization of the four features (see Section 3.5).

downsampling the l and m indices and data normalization. We show the results for this in Figures 2 and 3. As can be seen from Figure 2, there is no clear trend of clustering performance for different preprocessing cases. We also note that the shape of the score distribution remains the same across different downsampling factors for a given algorithm. This indicates that downsampling does not make a significant contribution to the clustering performance.

5.3. Evaluating Performance on Clean Data

So far, we have evaluated the performance of clustering algorithms on data with real RFI candidates along with simulated FRB candidates. This, as stated earlier, was a reasonable approximation of the candidates from real observations. Usually, in the case of a real transient, the pipeline only gets triggered at candidates from real transient, and no RFI is seen (either because of the amplitude of real transient or because low-level RFI is flagged). Therefore, here we report the performance of these clustering algorithms (at the optimal hyperparameters) on a data set containing candidates only from a real event and no RFI. This is done to test the generalizability of these clustering algorithms on data without RFI. This would also serve as an independent test on unseen data sets for which the hyperparameters of the algorithms were not tuned.

5.3.1. Completeness on Clean Data

We use the same procedure as described in Section 3.3 to generate a data set of 100 observations with candidates from one simulated FRB each. We randomly chose the parameters of the simulated FRBs and observing configurations, as explained earlier, and discarded any observation with less than 10 candidates.

We use *completeness* (see Section 4.3) to report the clustering performance on this data set. As there is no RFI candidate in this

Table 2
Optimal Hyperparameters Obtained for Different Algorithm and Preprocessing Combinations

Algorithm	Hyperparameter	DS1	DS2	DS4	Norm
Affinity Propagation	affinity	euclidean	euclidean	euclidean	euclidean
	random_state	1996	1996	1996	1996
	damping	0.974	0.965	0.985	0.881
	preference	-884	-222	-219	-202
Agglomerative Clustering	n_clusters	5	7	6	3
	affinity	euclidean	manhattan	euclidean	euclidean
	compute_full_tree	auto	auto	auto	auto
	linkage	ward	average	ward	ward
BIRCH	n_clusters	5.000	7.000	6.000	10.000
	threshold	0.341	0.876	0.676	0.957
	branching_factor	13.000	56.000	85.000	77.000
DBSCAN	min_samples	2	2	2	2
	eps	14.163	14.726	14.615	1.082
	metric	chebyshev	chebyshev	chebyshev	cityblock
	algorithm	auto	auto	auto	auto
	leaf_size	23	21	35	38
HDBSCAN	min_samples	5	5	5	5
	metric	euclidean	euclidean	euclidean	cityblock
	min_cluster_size	2	3	2	9
	cluster_selection_method	eom	eom	eom	eom
	allow_single_cluster	True	True	False	True
K-Means	algorithm	full	elkan	full	auto
	n_clusters	5	6	6	3
	n_init	13	15	28	26
	random_state	1996	1996	1996	1996
Mean Shift	bandwidth	16.416	32.750	19.350	1.229
	bin_seeding	True	False	True	True
	cluster_all	True	True	True	True
OPTICS	min_samples	2	2	2	2
	eps	14.782	14.376	14.551	1.095
	metric	minkowski	chebyshev	minkowski	cityblock
	min_cluster_size	8	6	6	8
	p	14.672	...	11.009	...
	cluster_method	dbscan	dbscan	dbscan	dbscan
	xi

data set, homogeneity would always be one and therefore is not a useful metric in this case. Here, a perfect clustering algorithm should generate just one cluster per observation for which completeness would be maximum, declining as the number of clusters increase. The overall completeness for a data set is the average of all the completeness values from 100 observations, each weighted by the number of candidates in the observation.

Figure 4 shows the overall completeness score of each algorithm. DBSCAN, HDBSCAN, Mean Shift, and OPTICS have the highest completeness score. It is to be noted that the completeness score of these four algorithms was worse when the data was preprocessed to zero mean and unit standard deviation (i.e., Norm). On the contrary, downsampling the image features did not show any significant effect (with a notable exception of DS4 for HDBSCAN).

5.4. Benchmarking

We evaluated the clustering speed of all clustering algorithms at their optimal hyperparameters. To do this, we generated an observation with a varying number of candidates consisting of random values for the four features. We then ran

all the clustering algorithms on those observations and recorded the time taken for just the clustering step. We did this test with optimal hyperparameters obtained for all four preprocessing cases. As the clustering speed is primarily dependent on the number of candidates to be clustered, we did not use real data for this test. We show the result of this test in Figure 5. The time taken did not vary significantly with parameters from different preprocessing cases, so we only show results using optimal hyperparameters for DS1 in this figure. DBSCAN and HDBSCAN are the fastest of these algorithms, while Affinity Propagation, Mean Shift, and OPTICS are the slowest, by at least an order of magnitude.

6. Discussion

6.1. Feature Importance

It is worth understanding the impact of feature selection on our outcome, as some features are expected to be more important than the others (Dash & Liu 2000; Guyon et al. 2005). We use a random forest classifier (Breiman 2001), implemented in scikit-learn, to estimate the relative feature importance of

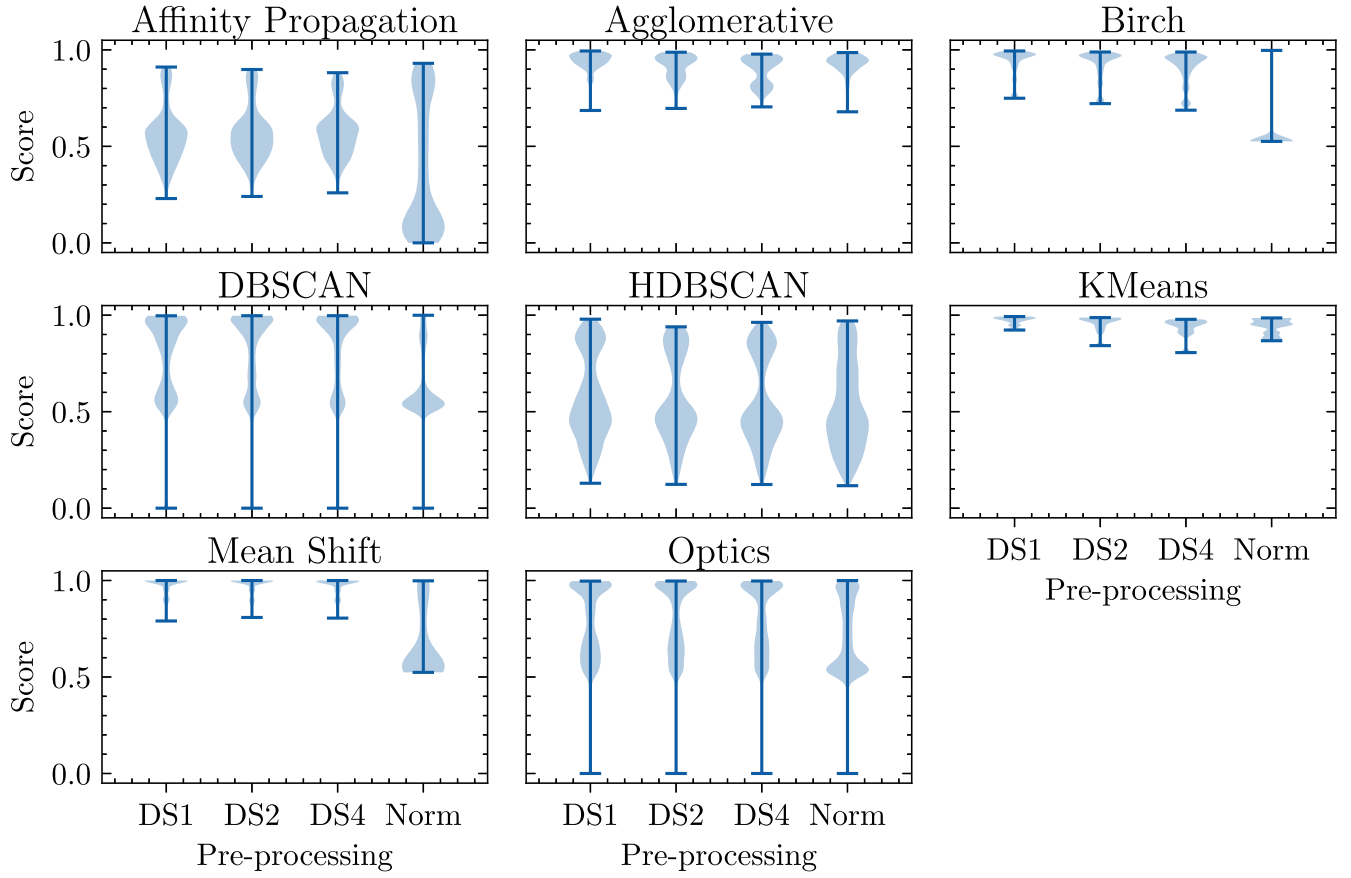


Figure 3. Violin plots of score vs. preprocessing for different clustering algorithms. Each violin plot shows the distribution of scores obtained at various hyperparameters evaluated during the random hyperparameter search. Different subfigures represent different algorithms (Sections 4.2 and 5.1). DS refers to downsampling applied to the l and m indices. Norm refers to normalization of the four features (see Section 3.5).

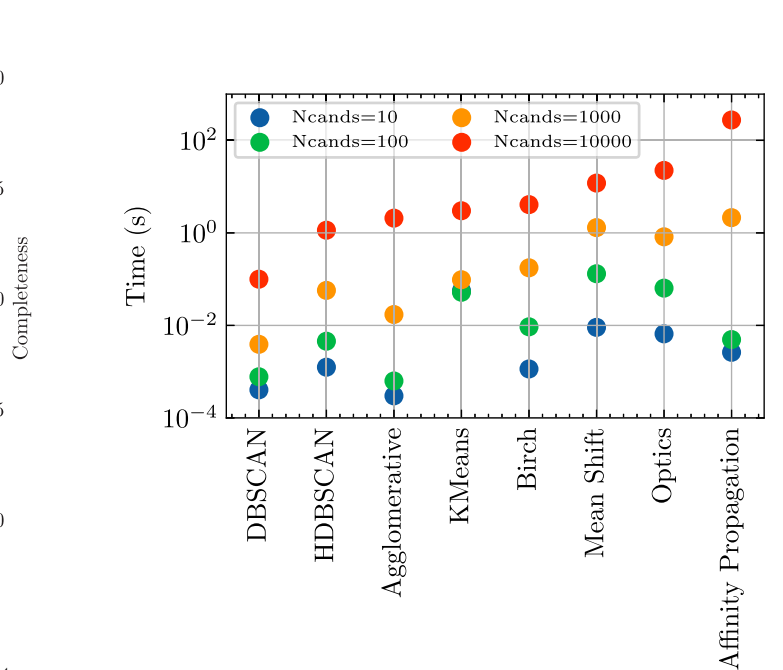
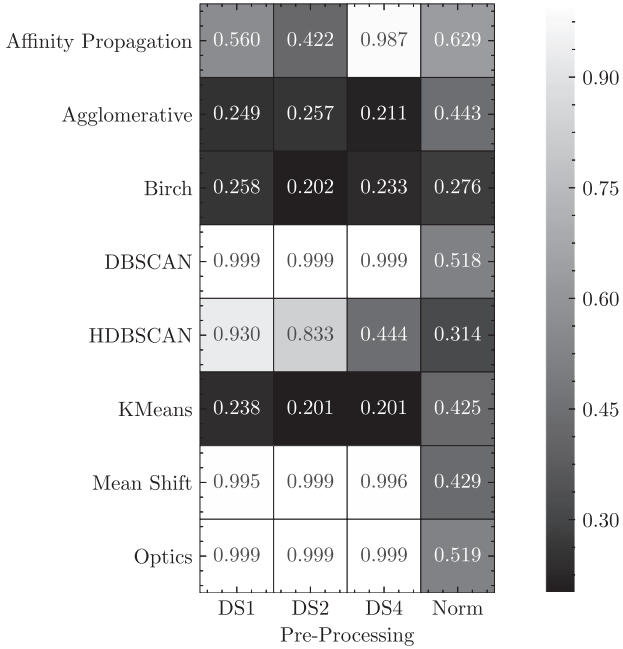


Figure 4. Completeness of different algorithms on clean data, i.e., without any RFI candidate (Section 5.3). A high completeness score is better and would imply that the FRB candidates are clustered in a minimum number of clusters for each of the 100 observations. Each algorithm was evaluated at its optimal hyperparameters (Table 2). DS refers to downsampling applied to the l and m indices. Norm refers to normalization of the four features (see Section 3.5).

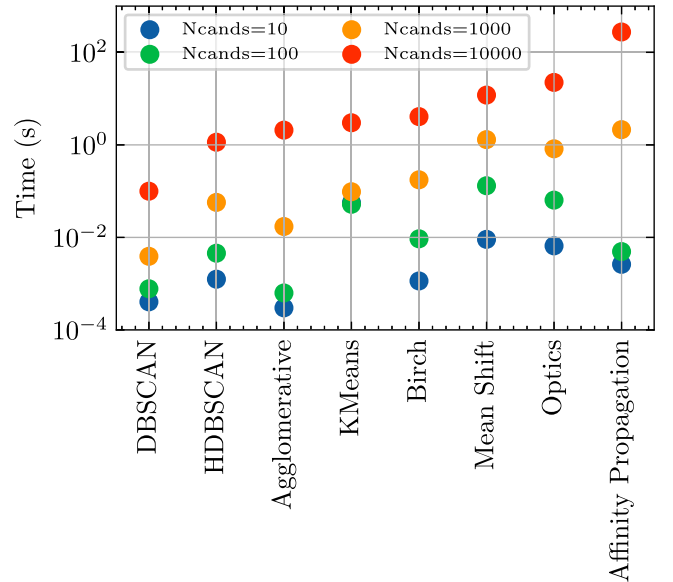


Figure 5. Time taken to cluster (in seconds) for each algorithm at their optimal hyperparameters. Different colors represent input data with a different number of candidates. Results are shown only at optimal hyperparameters for DS1. DBSCAN and HDBSCAN are much faster than algorithms like Mean Shift and Affinity Propagation (Section 5.4).

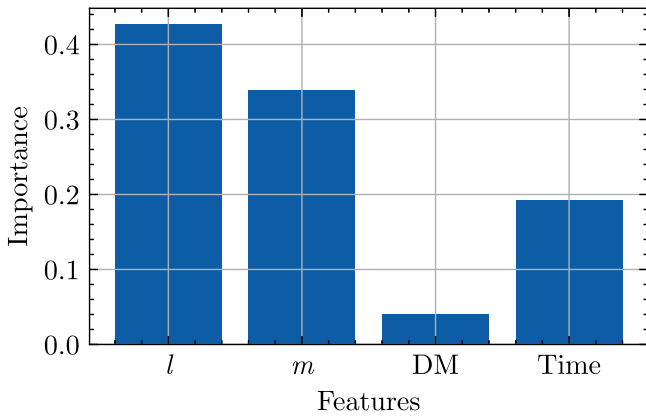


Figure 6. Importance of each feature, determined by training a random forest classifier to classify each observation into RFI and FRB. We trained the classifier individually on all observations in the test data set and took a weighted average of the individual feature importance to obtain the above plot. l and m contribute much more toward classification than DM and time (Section 6.1).

the four features, (DM, time, l , and m), in determining accurate clusters.

We used the test data set (see Section 3.4) without any preprocessing, containing candidates from 250 observations (hereafter we refer to this data as DS1). We knew the true labels (RFI and FRB) for each candidate in those 250 observations. For each observation, we trained a random forest classifier (at the default input parameters) using all the candidates in that observation. From the trained classifier, we then used `feature_importances_` to obtain the relative feature importance of each of the four features. This attribute of the random forest classifier calculates Gini importance (Breiman 2001) for each feature, which is representative of the importance of feature during classification. We repeated this for all the observations in our test data set. To estimate the total feature importance, we averaged all the importance for each feature weighing each by the number of candidates in that observation. The total importance obtained is shown in Figure 6. As can be seen from this figure, l and m (sky position indices of the candidate) contribute much more toward classification than the DM and time indices of the candidates.

A caveat to this simplistic analysis is that classification of all candidates into two classes, FRB and RFI, is not the same as clustering them into multiple clusters. The two cases would have been similar if all the RFI candidates could be assigned to a single cluster, which is not true. Therefore, even though this analysis shows that sky position contributes much more to classification, we suspect that the relative contribution of DM and time for the clustering task would be higher than what is obtained here.

6.2. What if I Only Use DM and Time for Clustering?

In the REALFAST system, we search for transients on the radio image. Therefore, for each candidate we get DM, time, l , and m information. But in many experiments, typically the ones using a single-dish telescope or the ones not performing an image-based transient search, only DM and time information is available for each detected candidate. Therefore, in those cases, only DM and time can be used to cluster the candidates together.

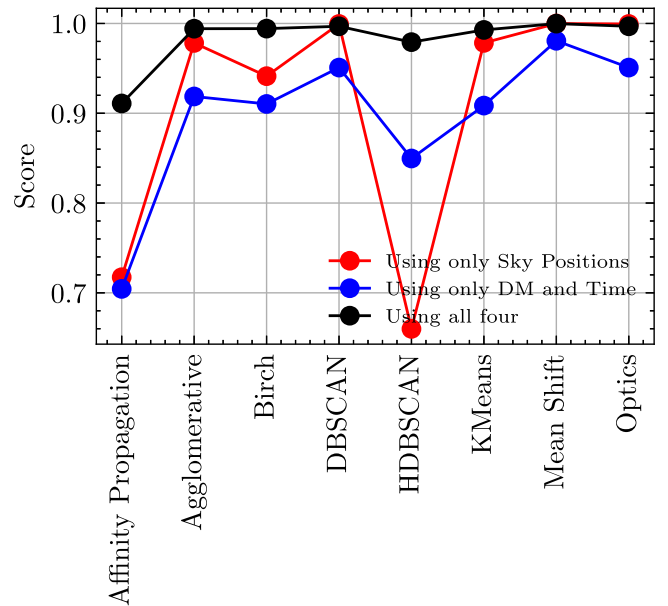


Figure 7. Score vs. algorithms for two feature clustering. Different colors represent different sets of features used to perform the clustering. We evaluated the scores on the test data set. Results with DS1 preprocessing are shown here (Section 6.2).

We tested the clustering performance using only DM and time to cluster the observations in our test data set. We used the optimal hyperparameters (listed in Table 2) on the test data set to evaluate this for all preprocessing cases. We also tested the clustering performance using only the l and m indices to cluster our test data set. As discussed in the previous section, the relative importance of sky positions is much higher than that of DM and time for a classification task. Therefore, clustering using only sky positions should give better scores than using just DM and time.

We show the results of these two tests in Figure 7 along with the scores when all four features are used for clustering. We only show scores for one preprocessing case (DS1), as results with other preprocessing techniques were also similar. As can be seen from this figure, scores obtained using just sky positions (red curve) or DM and time (blue curve) follow each other closely. Using sky positions shows minor improvement in score for most of the algorithms. Using all four features, as expected, gives the highest score, which is $\sim 10\%$ better than the other two cases.

This test highlights the importance of using sky positions along with the standard DM-time features to identify clusters of candidates originating from the same event. Therefore, if the sky position information is available for a candidate, it should also be used while clustering in the pipeline. With more and more interferometers (like Australian Square Kilometre Array Pathfinder (ASKAP) and Deep Synoptic Array (DSA)-110) implementing a REALFAST-like search for transients on radio images in the future, it would be useful for them to incorporate sky position information to cluster candidates in their respective pipelines.

As careful readers would have noticed, a caveat to this test is that in clustering with two parameters, we did not do a hyperparameter search to obtain the optimal hyperparameters that maximize the score using those two features. Instead, we used the hyperparameters that were optimal when four features

were used. A full hyperparameter search using two parameters might lead to a different set of parameters that might improve the score further. But even with this simple test, it can be noted that only using sky positions for clustering gives an improvement in score in almost all cases.

We have demonstrated in this and the previous section that sky positions are overall more important for clustering than DM and time. This could be because RFI candidates are more likely to span a wide range of time and DM values, which might overlap with those of FRB candidates while they are still localized in the radio image. Therefore, it is less likely (though still possible) for RFI to be very close to an FRB in the radio image. Similarly, RFI may be highly variable in frequency/time space, whereas in a radio image even unfocused (near-field) RFI will show up as contiguous streaks or other similarly structured patterns in images. Regardless of the reason for this, however, we have demonstrated here that when possible, sky positions should be used for clustering candidates.

6.3. But Which Algorithm Should I Use?

There are several considerations when deciding what algorithm to use based on the comparative analysis we have presented here.

1. *Maximum score:* As discussed in Section 4.3, we want the clustering algorithm to meet our application-specific goals of not missing a genuine event and singly identifying FRB candidates. Our performance metric (called score) maximizes when these goals are met. Therefore, we could search for a set of optimal hyperparameters for each clustering algorithm that gives the maximum score. All algorithms, except Affinity Propagation, have an optimal score above 0.95 (Figure 2).
2. *Generalizable:* The clustering algorithm needs to generalize to various types of data it can encounter in the pipeline. By testing the algorithms and optimal hyperparameters obtained in the previous step on an independent data set, one could quantify the algorithms' generalizability. To be more application specific, we tested this on a data set with observations containing candidates only from a real event, without any RFI and computed the completeness as the performance metric. Only four algorithms, DBSCAN, HDBSCAN, Mean Shift, and OPTICS gave completeness above 0.9 in this test (Figure 4).
3. *Speed:* Finally, the clustering algorithm would only have a limited amount of time to cluster candidates. Therefore, even for a large number of candidates, it should not exceed the limited time constraint. In our specific application for REALFAST, clustering is performed on candidates generated from small segments of data that are tens of seconds long. Based on the other pipeline steps, clustering should not take longer than a few seconds. The number of candidates detected by the search step typically varies between a few to thousands of candidates for a segment. Based on these requirements DBSCAN, HDBSCAN, Agglomerative Clustering, and K-Means can be used (Figure 5).

As an example using the REALFAST system, selecting the algorithms using the above three steps, we conclude that either DBSCAN or HDBSCAN can be used for clustering REALFAST data. Based on the results in Figures 4 and 7, we can further infer that DBSCAN is better than HDBSCAN. As reported earlier, we did not notice any improvement by using different preprocessing

techniques, therefore, no preprocessing is favored (Figure 2). A similar procedure can also be used to choose the clustering algorithm for any other single-pulse search pipeline or even for a more general clustering application.

7. Conclusions

In this paper, we have compared eight different unsupervised algorithms to cluster candidates generated by single-pulse search pipelines. We have also analyzed the effects of various preprocessing techniques on the data. We used real RFI from the REALFAST system and simulated FRB candidates to test different algorithms. We have developed a performance metric to quantify clustering performance. This metric makes sure that FRBs are not missed due to overaggressive clustering while still minimizing the number of clusters formed. Using a random hyperparameter search, we obtained optimal hyperparameters, which maximizes this metric for different algorithms. We test all the algorithms with optimal hyperparameters on an independent data set consisting of only FRB candidates to evaluate the generalizability of different algorithms. We also estimated the average clustering time for various algorithms on a data set of varying sizes. Finally, we have proposed a strategy that can be used to choose a clustering algorithm, using various tests mentioned earlier. We apply this strategy to obtain a clustering algorithm appropriate for the REALFAST system. This strategy can also be used at other single-pulse search systems to obtain the optimal clustering algorithm. Our strategy is generic enough to be used for other clustering applications. Our performance metric can also be used in other clustering applications where clustering information for only one cluster of interest is available, out of an unknown number of true clusters. We have also demonstrated that using spatial features for clustering improves the clustering performance compared to the traditional approach of just using DM and time features. All the scripts used in this analysis are openly available in a Github repository.⁹

K.A. would like to thank Shalabh Singh for useful discussions regarding the performance metric. K.A. and S.B.S acknowledge support from NSF grant AAG-1714897. S.B.S is a CIFAR Azrieli Global Scholar in the Gravity and the Extreme Universe Program. Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The NANOGrav project receives support from National Science Foundation (NSF) Physics Frontiers Center award number 1430284. The National Radio Astronomy Observatory is a facility of the National Science Foundation operated under cooperative agreement by Associated Universities, Inc.

Facility: EVLA.

Software: NumPy (Harris et al. 2020), Matplotlib (Hunter 2007), Pandas (Pandas Development Team 2020; McKinney 2010), scikit-learn (Pedregosa et al. 2011; Buitinck et al. 2013), HDBSCAN (Campello et al. 2015), rfpipeline (Law 2017).

Appendix A Clustering Algorithms

Here, we give a brief overview of all the clustering algorithms used in this analysis and some details and potential advantages/disadvantages of each algorithm for our clustering application.

⁹ <https://github.com/KshitijAggarwal/rfclustering>

A.1. K-Means

The K-Means (Macqueen 1967) algorithm is one of the most widely used clustering algorithms. Given an input number of clusters, the algorithm randomly initializes centroids for each cluster. Each example is then assigned a cluster based on the distance from that centroid. A new centroid is then computed for each cluster, and all examples are reassigned to the new centroids. This process is repeated until a convergence criterion is met. The main challenges with K-Means are that it is not good at identifying nonspherical clusters and requires the number of clusters as input, both of which limit its ability to generalize on different data sets.

A.2. Mean Shift

Mean Shift (Comaniciu & Meer 2002) is a centroid-based algorithm. The algorithm assumes that the data is drawn from an underlying probability density function and tries to estimate it using kernel density estimation. Then, it calculates a centroid for each data point using the kernel and iteratively updates the centroid using a mean shift vector. At convergence, the centroid will be placed at the nearest highest density peak of the density function. The same process is repeated for each data point, and the data points which lead to the same high-density peaks are then assigned to the same cluster. The only hyperparameter here is the bandwidth of the kernel. Mean Shift is not highly scalable as it requires multiple nearest neighbor searches.

A.3. Affinity Propagation

Affinity Propagation (Frey & Dueck 2007) is based on the concept of *message passing* between data points. It tries to find *exemplars*, i.e., members that are representative of clusters.

It starts by calculating a similarity matrix, which can be defined as the negative squared distance between two data points. The diagonal of this matrix is set to a constant, called *preference*, which is an input hyperparameter. Preference determines how likely a particular data point would be to become an exemplar. The algorithm then calculates three matrices, called Responsibility, Availability, and Criterion Matrix. These matrices are updated iteratively until a convergence criterion is met, and then clusters are assigned based on the information in Criterion Matrix. The details of the algorithm are given in Frey & Dueck (2007). Affinity Propagation's advantage is that it does not require the number of clusters as input, but the algorithm is computationally complex and can be slow on large data sets.

A.4. Agglomerative Clustering

Agglomerative Clustering (Franti et al. 2006) is a type of hierarchical clustering. Hierarchical clustering algorithms start with each example being a different cluster and then merge the ones that are closer until there is only one cluster. Therefore, they can form a hierarchy of clusters (at various distances), which is represented as a tree. A linkage criterion (see Section 5.1 of Jain et al. 1999) is used to decide the merge strategy. To determine the clusters from this cluster hierarchy, one has to choose a level or a cut in the tree. As was the case with K-Means, the main challenge with this algorithm is to choose the number of clusters.

A.5. DBSCAN

DBSCAN (Ester et al. 1996) is a density-based clustering algorithm. It assumes that clusters lie in dense regions. It primarily requires two input hyperparameters: a density threshold (*MinPts*)

of a core point and a radius (ϵ) of its neighborhood. A point that has at least *MinPts* adjacent points in its ϵ neighborhood is considered a core point. Core points and their neighborhood are considered dense regions that form clusters, and overlapping dense regions are merged into a single cluster. Any point that is neither a core point nor falls within the neighborhood of a core point is classified as noise. It does not require the number of clusters as input, although the clustering output is very sensitive to other input parameters.

A.6. OPTICS

OPTICS (Ankerst et al. 1999) is a density-based clustering algorithm. Similar to DBSCAN, OPTICS requires two hyperparameters: ϵ and *MinPts*, although ϵ is not necessary. It uses the following distances: core distance (minimum radius required to classify a given point as core point) and reachability distance (calculated by comparing the distance between two core points and their core distances) to order points. The reachability distance for points in a cluster would be low. The OPTICS algorithm builds a reachability graph, which assigns each sample a reachability distance. A post-processing procedure is applied to the reachability plot to determine clusters. This procedure can be very sensitive to the input parameters. An advantage of OPTICS is that it can find clusters of varying density. Like other density-based algorithms, OPTICS does not require the number of clusters as input and can also find nonspherical clusters.

A.7. HDBSCAN

HDBSCAN (Campello et al. 2015; McInnes et al. 2017; McInnes & Healy 2017) is very similar to OPTICS, i.e., it takes the approach of DBSCAN but extends it by varying the values of ϵ . It forms a hierarchical tree that shows the clustering output. By parsing through the tree, going from one large cluster to many smaller clusters, HDBSCAN constructs a tree with persistent clusters based on its only hyperparameter: minimum cluster size. It then uses a stability criterion to extract the final clusters from the cluster tree. Like OPTICS, HDBSCAN can also form clusters of varying density and do not require the number of clusters as input.

A.8. BIRCH

BIRCH (Zhang et al. 1996) is a hierarchical clustering algorithm used typically on very large data sets. It is local, which means that the clustering decision is made without scanning all data points and existing clusters. It uses a clustering feature (or CF) which consists of summary of statistics for a given sub-cluster. CF is used to calculate the distance between two sub-clusters. It creates a CF Tree (CFT) consisting of these CFs. The BIRCH algorithm has two hyperparameters: branching factor and threshold; the former limits the number of CFs in a node of CFT, while the latter limits the distance for a new sample to be a part of an existing CF. The terminal nodes of a CFT are then clustered using another clustering algorithm to obtain final clusters.

Appendix B

Parameter Ranges for Hyperparameter Tuning

Table 3 shows the hyperparameter ranges explored for different clustering algorithms during hyperparameter tuning.

Table 3

Hyperparameter Ranges Explored for Different Clustering Algorithms Using Random Sampling

Algorithm	Hyperparameter	Range/Values
Affinity Propagation	affinity	euclidean
	random_state	1996
	damping	0.5, 1
	preference	−1000, −200
Agglomerative Clustering	n_clusters	2, 10
	affinity	euclidean, manhattan, cosine
	compute_full_tree	auto
	linkage	complete, average, single, ward ^a
BIRCH	n_clusters	2, 10
	threshold	0.1, 20
	branching_factor	10, 100
DBSCAN	min_samples	2, 10
	eps	0.5, 15
	metric	euclidean, chebyshev, cityblock, manhattan, canberra, hamming ^b
	algorithm	auto
	leaf_size	20, 40
HDBSCAN	min_samples	2, 5
	metric	euclidean, chebyshev, cityblock, manhattan, canberra, hamming
	min_cluster_size	2, 10
	cluster_selection_method	com, leaf
	allow_single_cluster	True, False
K-Means	algorithm	auto, full, elkan
	n_clusters	2, 10
	n_init	10, 30
	random_state	1996
Mean Shift	bandwidth	10, 40 ^c
	bin_seeding	True, False
	cluster_all	True, False
OPTICS	min_samples	2, 10
	eps	0.5, 15
	metric	minkowski, euclidean, chebyshev, canberra, cityblock, manhattan, hamming ^b
	min_cluster_size	2, 10
	p	1, 15 ^d
	cluster_method	dbscan, xi ^e
	xi	0, 1

Notes. Random uniform sampling was used to sample hyperparameters for all the parameter ranges/values.

^a ward only works with euclidean affinity.

^b eps range of 0.1–1 was used with hamming metric, and a range of 0.1–4 was used with canberra metric.





^c Bandwidth of 1–10 was used for normalized preprocessing case.

^d p was used only with Minkowski metric.

^e Value of eps was used only with DBSCAN method, and value of xi was used when xi was selected as cluster method.

ORCID iDs

Kshitij Aggarwal  <https://orcid.org/0000-0002-2059-0525>
 Sarah Burke-Spolaor  <https://orcid.org/0000-0003-4052-7838>

Casey J. Law  <https://orcid.org/0000-0002-4119-9963>
 Geoffrey C. Bower  <https://orcid.org/0000-0003-4056-9982>
 Bryan J. Butler  <https://orcid.org/0000-0002-5344-820X>
 Paul B. Demorest  <https://orcid.org/0000-0002-6664-965X>
 Justin Linford  <https://orcid.org/0000-0002-3873-5497>
 Jessica Sydnor  <https://orcid.org/0000-0002-3360-9299>
 Reshma Anna-Thomas  <https://orcid.org/0000-0001-8057-0633>

References

- Agarwal, D., Aggarwal, K., Burke-Spolaor, S., Lorimer, D. R., & Garver-Daniels, N. 2020, *MNRAS*, **497**, 1661
- Aggarwal, K., Law, C. J., Burke-Spolaor, S., et al. 2020, *RNAAS*, **4**, 94
- Ankerst, M., Breunig, M. M., Kriegel, H.-P., & Sander, J. 1999, in Proc. 1999 ACM SIGMOD Int. Conf. on Management of Data, SIGMOD '99 (New York: Association for Computing Machinery), 49
- Bannister, K. W., Deller, A. T., Phillips, C., et al. 2019, *Sci*, **365**, 565
- Barsdell, B. R. 2012, PhD thesis, Swinburne Univ. Technology
- Bergstra, J., & Bengio, Y. 2012, *J. Mach. Learn. Res.*, **13**, 281
- Breiman, L. 2001, *Mach. Learn.*, **45**, 5
- Buitinck, L., Louppe, G., Blondel, M., et al. 2013, in ECML PKDD Workshop: Languages for Data Mining and Machine Learning, ed. H. Blockeel et al. (Berlin: Springer), 108
- Campello, R. J. G. B., Moulavi, D., Zimek, A., & Sander, J. 2015, *ACM Trans. Knowl. Discov. Data*, **10**, 5
- Chatterjee, S., Law, C. J., Wharton, R. S., et al. 2017, *Natur*, **541**, 58
- Comaniciu, D., & Meer, P. 2002, *ITPAM*, **24**, 603
- Cordes, J. M., & McLaughlin, M. A. 2003, *ApJ*, **596**, 1142
- Dash, M., & Liu, H. 2000, in Proc. IV Pacific-Asia Conf. Knowledge Discovery and Data Mining, Current Issues and New Applications, PADKK '00 (Berlin: Springer), 110
- Deneva, J. S., Cordes, J. M., McLaughlin, M. A., et al. 2009, *ApJ*, **703**, 2259
- Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. 1996, in Proc. Second Int. Conf. on Knowledge Discovery and Data Mining, KDD'96, ed. E. Simoudis et al. (Menlo Park, CA: AAAI Press), 226
- Franti, P., Virmajoki, O., & Hautamaki, V. 2006, *ITPAM*, **28**, 1875
- Frey, B. J., & Dueck, D. 2007, *Sci*, **315**, 972
- Guyon, I., Gunn, S., Ben-Hur, A., & Dror, G. 2005, in Advances in Neural Information Processing Systems, Vol. 17 ed. L. Saul, Y. Weiss, & L. Bottou (Cambridge, MA: MIT Press), 545
- Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, *Natur*, **585**, 357
- Hunter, J. D. 2007, *CSE*, **9**, 90
- Jain, A. K., Murty, M. N., & Flynn, P. J. 1999, *ACM Comput. Surv.*, **31**, 264
- Kocz, J., Ravi, V., Catha, M., et al. 2019, *MNRAS*, **489**, 919
- Law, C., Tendulkar, S., & Clarke, T. 2021, *ATel*, **14526**, 1
- Law, C. J. 2017, rfpipeline: Radio interferometric transient search pipeline, Astrophysics Source Code Library, ascl:1710.002
- Law, C. J., Bower, G. C., Burke-Spolaor, S., et al. 2018, *ApJS*, **236**, 8
- Law, C. J., Butler, B. J., Prochaska, J. X., et al. 2020, *ApJ*, **899**, 161
- Leung, C., Mena-Parra, J., Masui, K., et al. 2021, *AJ*, **161**, 81
- Levin, L. 2012, PhD thesis, Swinburne Univ. Technology
- Lorimer, D. R., Bailes, M., McLaughlin, M. A., Narkevic, D. J., & Crawford, F. 2007, *Sci*, **318**, 777
- Lorimer, D. R., & Kramer, M. 2004, Handbook of Pulsar Astronomy, Vol. 4 (Cambridge: Cambridge Univ. Press)
- Macqueen, J. 1967, in Proc. V Berkeley Symp. on Mathematical Statistics and Probability, ed. L. M. L. Cam et al. (Berkeley, CA: Univ. California Press), 281
- McInnes, L., & Healy, J. 2017, in 2017 IEEE Int. Conf. on Data Mining Workshops (ICDMW) (Piscataway, NJ: IEEE), 33
- McInnes, L., Healy, J., & Astels, S. 2017, *JOSS*, **2**, 205
- McKinney, W. 2010, in Proc. IX Python in Science Conf., ed. S. van der Walt & J. Millman (Austin, TX: SciPy), 56
- Michilli, D., Masui, K. W., Mckinven, R., et al. 2021, *ApJ*, **910**, 147

- Pandas Development Team 2020, pandas-dev/pandas: Pandas, latest, Zenodo, doi:[10.5281/zenodo.3509134](https://doi.org/10.5281/zenodo.3509134)
- Pang, D., Goseva-Popstojanova, K., Devine, T., & McLaughlin, M. 2018, *MNRAS*, **480**, 3302
- Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, *Journal of Machine Learning Research*, **12**, 2825
- Petroff, E., Barr, E. D., Jameson, A., et al. 2016, *PASA*, **33**, e045
- Rosenberg, A., & Hirschberg, J. 2007, in Proc. 2007 Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL) (Prague: Association for Computational Linguistics), 410
- Zhang, T., Ramakrishnan, R., & Livny, M. 1996, in Proc. 1996 ACM SIGMOD Int. Conf. on Management of Data, SIGMOD '96 (New York: Association for Computing Machinery), 103