ForestFire: Label Propagation Clustering Inspired by Forest Fire Dynamics

Jeremy Goldwasser[1]

Senior Thesis Advised by Professor Mark Gerstein[1,4,5]

May 13, 2021

With the help of Flynn Chen[1], Philip Tuckman[2], and Jing Zhang[3]

1. Department of Statistics and Data Science, Yale University
2. Department of Atmosphere, Oceans, and Climate, Massachusetts Institute of Technology
3. Assistant Professor, Department of Computer Science, University of California, Irvine
4. Department of Molecular Biophysics and Biochemistry, Yale University
5. Department of Computer Science, Yale University

Abstract

Clustering is an unsupervised learning technique that partitions a dataset into groups of similar points. Most current clustering algorithms make assumptions about the data that don't necessarily hold up in practice - for example, the number of clusters, their shape, and the amount of space between them. In contrast, our method ForestFire is able to learn the data manifold with only one effective parameter. The algorithm, per its name, "burns" each cluster by selecting an initial centroid and iteratively propagating its label outwards as far as possible. In spite of its simplicity, ForestFire outperforms 11 popular methods on 6 benchmarks including Louvain, K-Means, and DBSCAN with regards to accuracy and runtime. Further, it can run on both coordinate and graphical data, and does not need to be retrained with the arrival of new data. Lastly, its results can be internally validated with a Monte Carlo-like simulation that evaluates the stability of each point's predicted label. This enables us to examine which other clusters each point may be in, as well as which predicted clusters are really the same. In this paper, we describe our algorithm and showcase its performance on six benchmarks. We also highlight its usefulness in single-cell sequencing data, particularly for discovering rare cell types and mitigating the multiplet cell phenomenon.

1. Introduction

1.1 Motivations

Clustering is a unsupervised learning method used widely in statistical data analysis and data mining. These algorithms seek to find the optimal way to split a set of unlabeled points into two or more groups, or clusters. Clusters are distinguished from one another based on some salient characteristics that the algorithm is able to discern implicitly from the data. For example, say a team of engineers at Facebook wants to figure out whether its American users identify as Democrats, Republicans, or neither. They don't have access to individual voting records, of course, so instead they take an algorithmic approach. Given a giant dataset of politically-charged information - where people are from, how old they are, what news articles they've liked, etc. - they find the best way to split it into three groups.

While this example may make clustering seem intrusive or downright nefarious, its applications can be much more beneficial. One study, from Professors Parvinder Singh and Mandeep Singh, detected fraudulent bank activity with the K-Means algorithm [1]. Clustering also has important biomedical applications. Modern single-cell sequencing techniques allow researchers to examine genetic information down to the resolution of a single cell [6, 7]; applied to this kind of data, clustering algorithms can categorize cells based on their genomic features.

This is particularly useful in molecularly targeted therapy, which requires the detection of subtypes of cancer cells [8].

Not all single-cell datasets, however, are easy to work with. They could include measurements of rare or even unknown cell types that researchers did not expect to find. These cell types may be vitally important, but any prior assumptions about cellular composition would make them difficult to detect [9]. Another problem in single-cell sequencing is the multiplet cell phenomenon, which occurs when two or more cells are mistakenly sequenced as one [10-12]. Distinguishing between real and "artifactual" cells is a difficult task; as a result, often these faulty measurements cannot be reliably removed.

Analysis of these datasets may be severely confounded if they have more cell types than anticipated [5]. Algorithms that input the number of clusters as a hyperparameter are bound to perform poorly, as the data will actually have at least one more cluster. Because of this, clustering methods for single-cell sequencing analysis should not require this number to be known.

1.2 ForestFire

Our method formulates clustering as a forest fire, where points represent trees, clusters represent groves, and labels represent fires. A fire starts when a single tree, the flint, catches fire. The fire propagates to its neighbors, then to the neighbors of those points, and so on. When no new points are hot enough to ignite, we reiterate with a new flint. This process continues until all of the points have been assigned to a cluster. Lastly, we reassign all the points in miniscule clusters to larger ones.

Crucially, ForestFire does not make any assumptions about the distribution of the clusters. While there are a few parameters of lesser importance, ForestFire only has one effective parameter: The "fire temperature," which determines how quickly the fire spreads through the graph. This framework is highly robust, as it permits labels to propagate through a data manifold of any shape. It is even able to cluster graphs, rather than being limited to coordinate data.

The lack of assumptions is a key strength of our method. Using the example of single-cell sequencing, we do not know if genetic measurements of the same type of cell will adhere to a Gaussian distribution, or any kind of distribution for that matter. And as mentioned earlier, we should not make assumptions on the true number of clusters, due to the existence of multiplet artifacts and rare cell types. ForestFire is better equipped to process this data than most other clustering methods.

Another strength of ForestFire is its ability to calculate pointwise stability values for its predictions. Using a sort of Monte Carlo simulation, we re-run ForestFire many times, burning clusters in different orders and using different points as the flints. From these results, we can calculate the entropy of the labels associated with each point. If a point is almost always assigned to the same cluster, then the entropy is low and we have high confidence in the stability of our original prediction. The same cannot be said if its cluster labels are varied.

In Section 2, we review the different families of clustering algorithms, and explain where ForestFire fits into them. In Section 3, we explain the algorithm in greater detail. Finally, in Section 4, we demonstrate its effectiveness on common clustering benchmarks and real-world single-cell datasets.

2. Related Work

Clustering has been attempted with a number of different approaches. In general, these approaches fall within five broad categories: Connectivity-based, centroid-based, distribution-based, density-based, and spectral-based methods.

Connectivity-based methods, also known as hierarchical clustering, iteratively merge sets of data points to obtain a hierarchical structure [13]. They can be represented with a dendrogram, a tree that tracks the history of the merges. Unfortunately, each true cluster may take a different number of merges to construct, making their boundaries difficult to detect. Two algorithms known as Louvain and Leiden overcome this by optimizing modularity, a measure of the connectivity within each cluster [18 19] A resolution parameter determines whether the number of clusters will be low or high. These two methods are very effective and do not require the number of clusters to be known ahead of time - a property that makes them them appealing for single-cell analysis. However, their runtimes greatly exceed those of most other methods.

Centroid-based clustering aims to find the optimal location of some predetermined number of cluster centroids. K-Means, the most notable example of this type of clustering, assigns each point to the centroid for which the Euclidean distance is smallest. In doing so, it assumes that each point is drawn from a spherical distribution around a centroid; it also assumes that neighboring clusters have similar variances. Such assumptions limit the robustness of centroid-based clustering. These algorithms also lack convergence guarantees, as their standard optimization method, Lloyd's algorithm, finds only a local minimum.[1] As it turns out, finding the global minimum can be an NP-hard task.[2]

In distribution-based clustering, distributional assumptions are explicitly made at the onset. Each cluster is defined as a set of samples drawn from a distribution at some to-be-determined location. Most prominently, the Gaussian Mixture Model assumes that data is generated from a known number of gaussian distributions interspersed throughout the graph. It then uses Expectation-Maximization (EM) to find their means and variances. In both distribution-based and centroid-based clustering, the distance between points and centroids can be interpreted as pointwise stability values: Points closer to the center of a cluster are more likely to belong to it. Once again, though, robustness is limited by assumptions on the number and shape of clusters.

Density-based clustering relies on sparse regions between clusters to distinguish one from the other. Two well-known algorithms in this category, DBSCAN and OPTICS, produce strong

[1] https://arxiv.org/abs/2002.06694
[2] https://cseweb.ucsd.edu/~avattani/papers/kmeans_hardness.pdf

results in only O(n log n) runtime [15,16]. However, they tend to perform poorly when there is little or no space between clusters. They are also not guaranteed to assign every point to a cluster, and depend heavily on a range parameter.

Lastly, spectral methods rely on principles from graph theory. These methods use kernels to convert pairwise distances into affinities, which are equivalent to the adjacencies between vertices in a graph. Clustering can then be reformulated as finding sub-communities in this graph. The characteristic method, Spectral Clustering, computes the Laplacian from this affinity matrix, then runs K-Means on the eigenvectors whose eigenvalues are smallest [17]. Spectral Clustering can sometimes work when the number of clusters is unknown. This number is equal to the number of eigenvalues before the spectral gap, the position at which sorted eigenvalues jump from low to high. This gap will only recognizable if the dataset's clusters are distinct.

Our algorithm, ForestFire, combines principles from three of these five approaches. Like hierarchical clustering, it grows each cluster by greedily adding points over multiple iterations. How many clusters are grown is dependent on the fire temperature parameter, which is similar to the resolution parameter in Louvain. Like centroid-based methods, each cluster forms around a central point, which we refer to as the "flint." And like spectral-based methods, ForestFire makes heavy use of an affinity matrix. It can also be understood and applied in a graphical context.

2. Methods

2.1 Clustering

ForestFire burns clusters via label propagation. Each cluster begins with a single point, the flint. Then, using a breadth-first search-like approach, the algorithm iteratively calculates how hot the cluster's fire is at every unlabeled point. If the heat exceeds a certain threshold, the point is added to the cluster. When no more points are hot enough to ignite, a new flint is selected and a new cluster begins burning. This continues until all of the points in our dataset have been assigned to a cluster. Finally, all points in small clusters are reassigned to larger ones.

In the sections that follow, we will explain flint selection and the heat function in greater detail. Then, we show three features that decrease runtime. Lastly, we show how ForestFire reassigns points from small clusters to larger ones. We recommend looking at the pseudocode before proceeding.

ForestFire Pseudocode

```
Initializations:
    -   N_unburnt = N
    -   label_number = 0
    -   Labels = [-1, -1, …, -1] (n times)

Compute affinity matrix (Section 2.1.1)
Set heat threshold
While N_unburnt > 0:
    Select a flint (Section 2.1.2)
    Labels[ flint ] = label_number  # create a new cluster and add the flint)
    N_unburnt = N_unburnt - 1
    While the fire is spreading:
        For every point:
            If label_number == -1 # unburnt
                Compute the point's heat (Section 2.1.3)
                If heat > threshold:
                    Set its label to label_number # ignite the point
                    N_unburnt = N_unburnt - 1

        If no points caught fire, the fire stops spreading
    label_number = label_number + 1

Reassign points in small clusters to larger ones (Section 2.1.4)
```

2.1.1 Affinity Matrix

Computing the affinity matrix is the first step in our algorithm. This matrix, which represents pairwise distances in a non-linear fashion, enables us to spread a fire through each cluster. It comes into play for selecting flints (2.1.2) and computing the heat at each point (2.1.3)

Our dataset contains n points of equal dimension, arranged in a data matrix. From this, we calculate the n x n distance matrix, whose elements are the Euclidean (L2) Distance between each pair of points. Distances are then converted into affinities using a Gaussian kernel [20]. (We found that ForestFire performs much worse with an Adaptive kernel. When the bandwidth parameter is variable, effective flint selection is impossible and fires spread more easily between clusters.) To determine the spread of this Gaussian distribution, we want to choose a bandwidth parameter roughly equal to the distance between nearby points in the same cluster. The user can either specify this parameter or provide some small k such that it is equal to the average distance from a point to its k'th nearest neighbor. This latter option is preferable when the spread of the dataset is difficult to discern. In our experimentation, we had successful results on all datasets when we set k to 10.

In converting our data from actual measurements to pairwise affinities, we are implicitly representing our data as a fully connected graph G = (V, E). Here, each vertex represents a point; the edge connecting any two vertices represents their similarity, measured with affinity. Under this graphical formulation, pointwise affinity is analogous to adjacency. Indeed, as demonstrated in Section 3.2 ForestFire can successfully cluster any graph using its adjacency matrix.

2.1.2 Flint Selection

Flint selection is a crucial part of our algorithm. If our flint is relatively far from other points, then it will burn an unreasonably small cluster. Worse, if it is near the edge between two clusters, then the fire it ignites may spread to both at once. Because of this, we want to choose a flint F that is far from the nearest cluster centroid. We also want it to be in a densely-connected area of the graph, because this signifies that it is likely at the center of a cluster and should propagate outwards in multiple directions.

We call our algorithm for flint selection the "space laser." This name comes from Congresswomen Marjorie Taylor Green, who bafflingly claimed that the 2020 California wildfires had been ignited by Rothschild-funded space lasers.

If no clusters have been burnt, then we simply choose the point at the densest part of the graph. Otherwise, then we want to start a fire at a point far from all the other clusters. To do this, we find the affinity from each unlabeled point to the nearest cluster centroid. We then consider only the points for which these affinities are in the 25th percentile or lower (i.e. the distances are in the 75th percentile or higher). Among this subset of points, the one with the maximum density is chosen.

2.1.3 Heat Function

In a forest fire, heat decays non-linearly as distance increases from the fire. Affinity, as described in Section 2.1, is a perfect way to model this decay. Heat is also proportional to the fire temperature, the main hyperparameter in our algorithm. We can thus calculate the total heat at a certain point x as follows:

Heat = fire_temp * (sum(A[x, x_i]) for i in 1:m)

where A is the affinity matrix and x_1, …, x_m are the points in the burning cluster.

A point catches on fire if its heat exceeds the temperature threshold. Note that this heat threshold corresponds to the flash point, the temperature at which a tree in a forest will burst into flame.

2.1.4 Decreasing the Runtime

We can burn a cluster more quickly if we sort points by distance to the flint. Recall that ForestFire grows each cluster from the flint alone, passing repeatedly through all N points until none ignite. Every time a point catches fire, it is immediately added to the cluster. This contributes extra heat for all subsequent points in the pass, thus increasing the chance that they ignite. Because of this, each pass is likelier to ignite more points if it orders points by their proximity to the flint. (We can safely assume that in general, points closer to the flint are more likely to be in its cluster than points further away.) It will then take fewer passes until an equilibrium is reached.

Adding on to this method, we can further decrease the runtime by cutting short each pass through all N points. If we sort by distance and many consecutive points don’t ignite, then it’s safe to stop the pass. An optional hyperparameter, n_before_break, determines just how many consecutive points are required. In our experimentations, we typically set this hyperparameter to 200. It should not be too small, as the shape of the cluster may be unusual.

Another technique for accelerating ForestFire works only when the true number of clusters, k, is known. If the k’th smallest cluster that we have burnt is greater than the number of points remaining, then there is no need to continue. The clusters that would burn would all be smaller than the k biggest existing ones; as a result, all their points would inevitably be relabeled anyway.

None of these methods are listed in the pseudocode, as they are not integral to the functionality of the algorithm. However, all three of them are implemented in the code. Only the second requires a hyperparameter to be selected.

2.1.5 Removing Small Clusters

ForestFire is free to burn more than the true number of clusters, K. This may happen even when K is specified. If this occurs, then we need to relabel every point in a small cluster to one the K largest ones. We will also need to assign unlabeled points to these large clusters if we stopped burning clusters early for runtime reasons (see Section 2.4). To relabel a point, we calculate the heat from each large clusters. We then assign it the label of the cluster that produced the maximum heat.

We may still want to remove small clusters even when K is not known. ForestFire has two ways of doing this. First, it takes a hyperparameter that specifies the minimum size of a cluster. All points in clusters under this size will be relabeled. Secondly, another hyperparameter ensures that no clusters are significantly smaller than the rest. This hyperparameter is the minimum ratio between sorted cluster sizes. So if the size of j+1th largest cluster is below a certain fraction of the jth's, then it and all smaller clusters will be relabeled.

2.2 Monte Carlo Verification

We can validate ForestFire's predicted clusters without access to the ground truth by means of a Monte Carlo-like simulation. In each Monte Carlo simulation, we burn clusters in a different order, using a different set of flints. The aggregated results inform us of the extent to which each original label is stable. More specifically, they tell whether perturbations to the fire cause the labels to change.

Let K_model denote the number of clusters burnt in the original running of ForestFire. First, we take a random ordering of our K_model clusters. For each cluster in this ordering, we randomly select a point originally labeled in this cluster, then burn outwards from it. Repeating this for many trials yields a wide range of labels.

Parameters:

- K_model = number of clusters in original ForestFire clustering
- S = number of simulations

Pseudocode:

```
For _ in S:
    cluster_order = random permutation of {1, …, K_model}
```

```
For each cluster in cluster_order:
    Flint = random point originally put in that cluster
    Burn a cluster from that flint (See section 2.2)
```

We can get two kinds of pointwise stability values from the simulation's labels: The entropy, and the probability that each point will be put in its original cluster. Regardless of which stability metric we choose, we can take a histogram of these values and choose a threshold above which points are considered unstable. This allows us to filter our points by stability.

There are two main reasons for a point to be labeled as unstable. One is that it is near the boundary between two or more clusters. These points are unstable because whenever one of these clusters comes earlier in the permutation, its fire will likely engulf the point. The other reason is that there are not multiple clusters to begin with - that what ForestFire has seen as separate clusters are really the same cluster.

3. Results

3.1 Gaussian Blobs

We constructed two datasets of isotropic Gaussian blobs interspersed around a unit circle. Every blob has a standard deviation of $\sigma = 0.15$ and a roughly equal share of the 1,500 points. We used a fire temperature of 0.025 in both cases. Figure 1 illustrates ForestFire burning one cluster at a time; Figures 2 and 4 show the results when K is known, and Figure 3 and 5 show the results when it is not. ForestFire is able to identify each cluster's exact shape, even when they are right beside each other. Notice that when min_cluster_size and/or min_size_ratio is high enough, the labels when K is unknown is the same as those when it is.

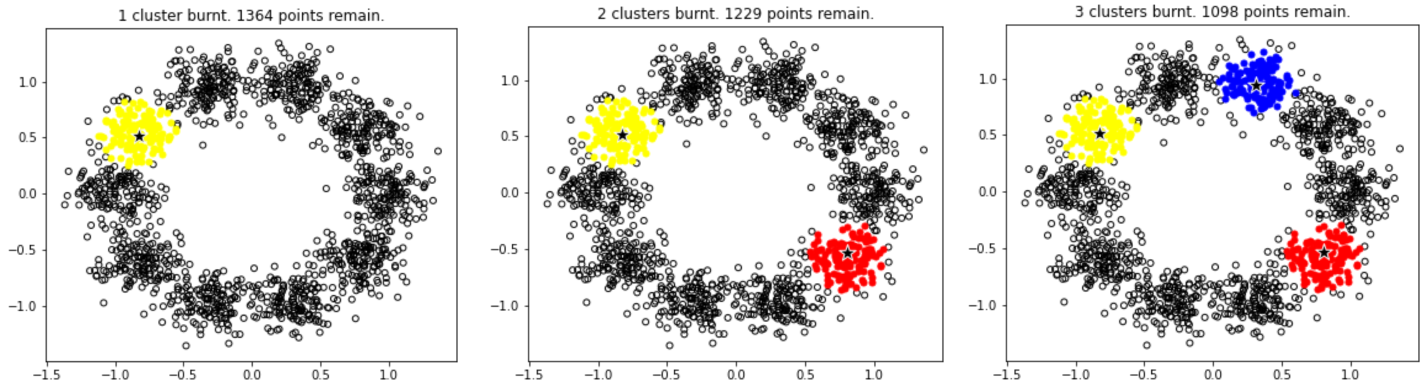


Figure 1: ForestFire burning the first 3 clusters in the K=10 dataset. The stars mark each cluster's flint.

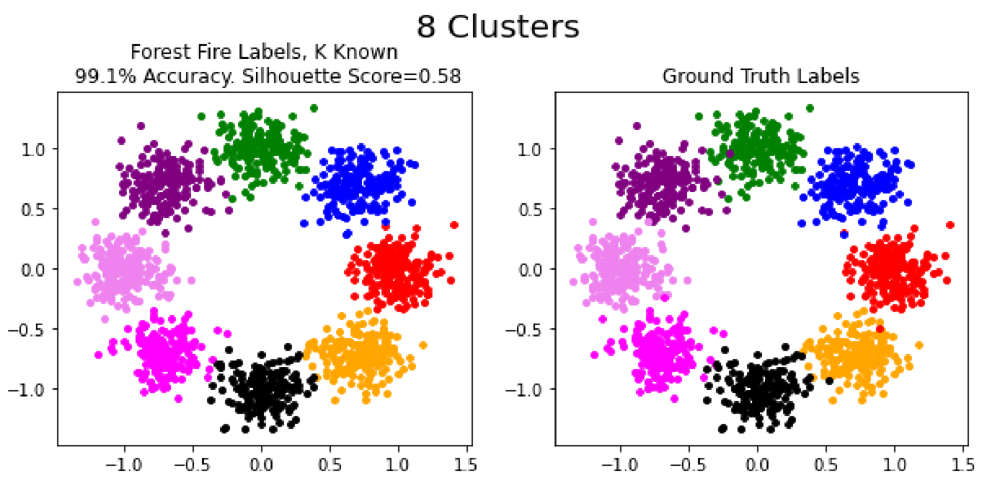


Figure 2: 8 clusters, K known. Close inspection reveals that the misclassified points are outliers overlapping in other clusters.

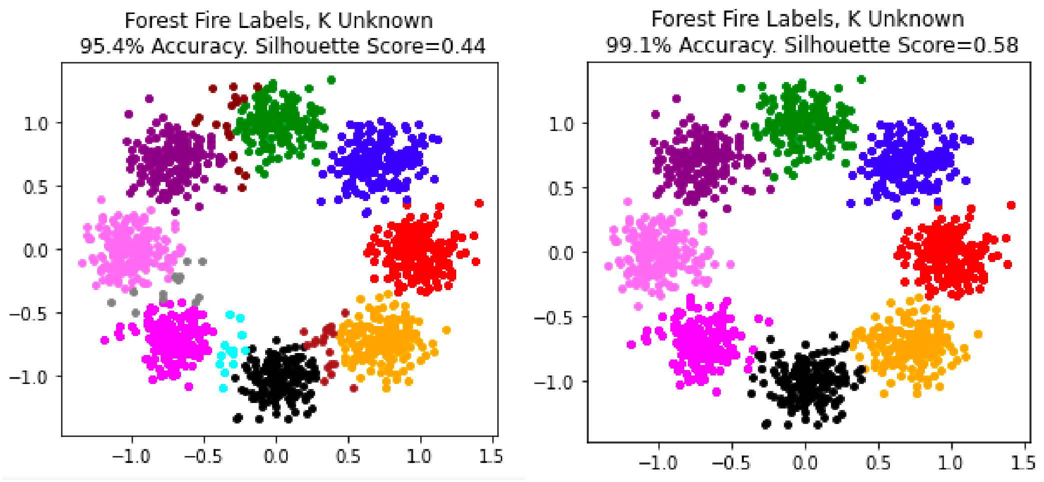


Figure 3: 8 clusters, K Unknown. In the plot on the left, min_cluster_size=10 and min_size_ratio=0.1; on the right, 20 and 0.2.

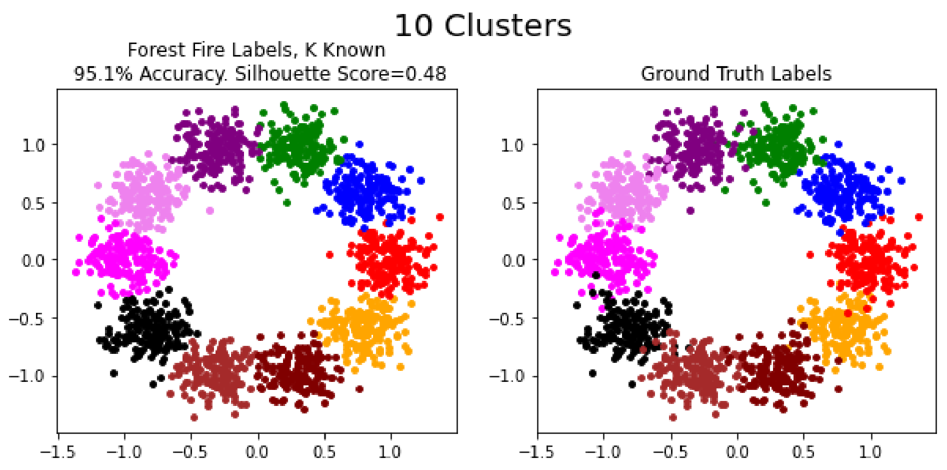


Figure 4: 10 clusters, K Known

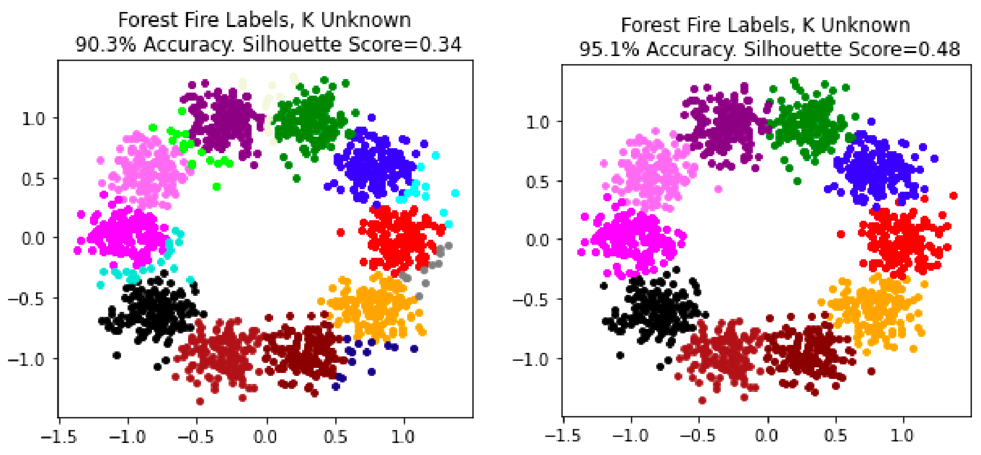


Figure 5: 10 clusters, K Unknown. Min_cluster_size and min_size_ratio are 10 and 0.1 on the left, respectively, and 20 and 0.2 on the right

Interestingly, the number of clusters can be inferred from the heat over time plot (Fig. 6). The plot records the heat at each point when it is labeled. Every spike corresponds to the beginning of a fire. The first points that are burnt are all near the center of the fire, so their heat is high. As newly lit points become further and further from the center, the heat falls accordingly.

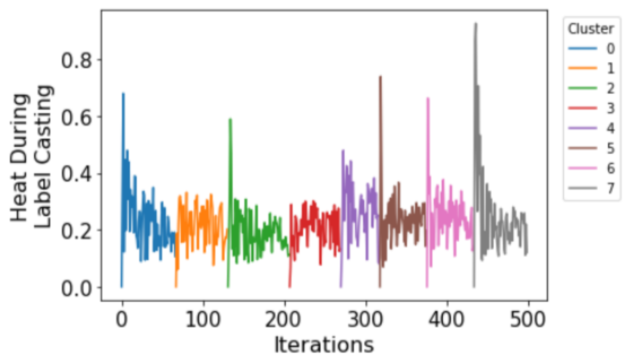


Figure 6: Heat of points when they catch fire. Clustered on Gaussian blobs dataset with 8 clusters.

For internal validation, we ran 100 Monte Carlo simulations to evaluate the stability of our labels. We then computed the entropy of these labels, displayed in Figure 7. It shows that there is more label ambiguity on the edges of clusters, exactly as one would expect.

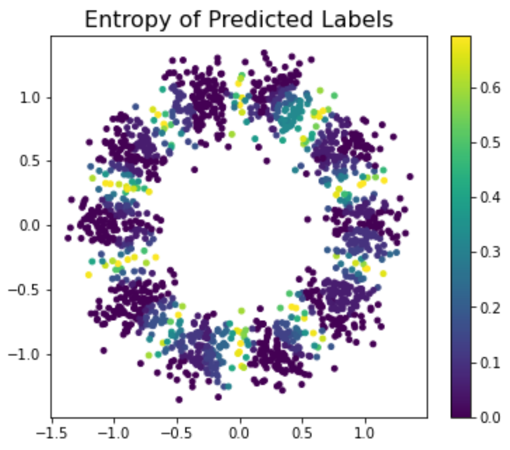


Figure 7: Entropy of K Unknown labels, K=10

3.2 Graphical Data

We ran ForestFire on a simple graph generated with a Stochastic Block Model. Each pair of points shares an edge with probability P_ii if they are in the same cluster, and P_ij if they are not. We set P_ii to 0.5 and P_ij to 0.1, generating 1500 points in 8 clusters (Fig. 8). A fire temperature of 0.1 yielded an accuracy of 96.9% for models with K known and K unknown.

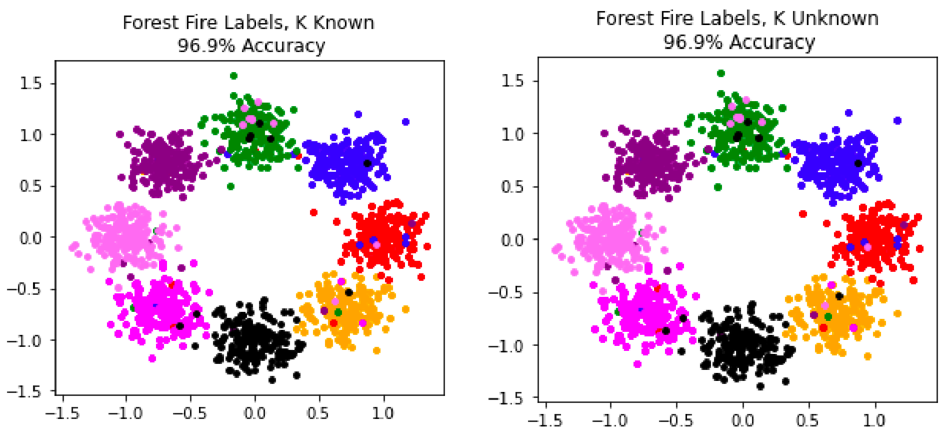


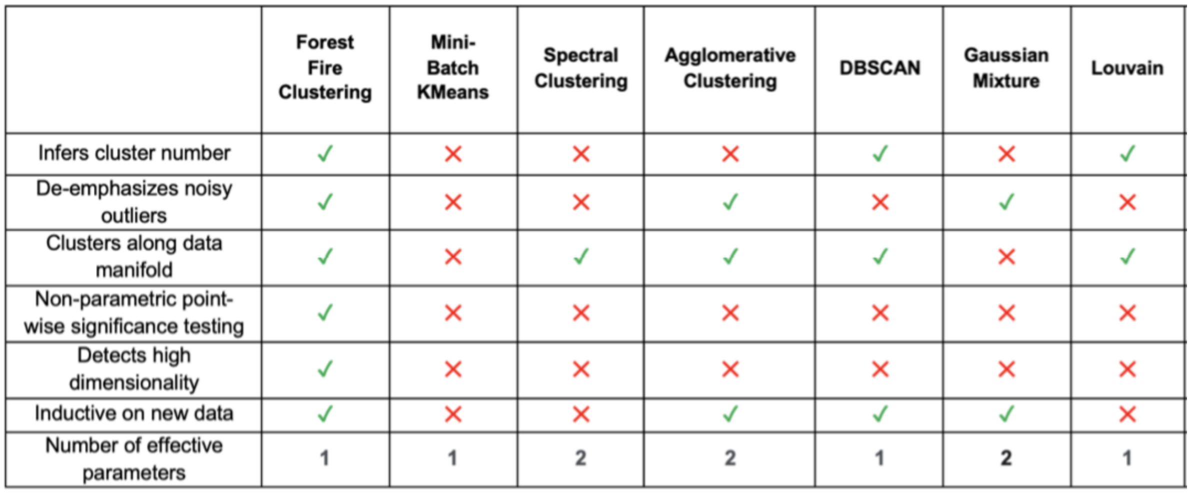
Figure 8: ForestFire on graphical data. K=8, N=1500, P_ij=0.1, P_ij=0.1

3.3 Benchmarking on Toy Datasets

ForestFire incorporates many advantages from existing clustering algorithms (Table 1). We benchmarked it against 11 such algorithms on 6 toy datasets (Fig. 9), with impressive results. ForestFire correctly clusters over 95% of points on all six datasets - something only Louvain and Spectral Clustering are also capable of doing. Louvain, however, is incredibly slow, taking between 30 and 70 seconds to cluster each dataset. ForestFire, in contrast, runs in well below a second for all but one dataset. (It takes 1.77 seconds on the concentric circles.) While Spectral Clustering is comparable in terms of runtime, it consistently gets slightly lower accuracy. More importantly, it does not have a mechanism for validating its predictions like ForestFire does. In addition, it must be completely re-run whenever new points are added, whereas ForestFire can quickly label new data.

Connectivity- and density-based methods perform well on most datasets. On some, though, they struggle to distinguish between clusters that are close together. Two hierarchical methods, Agglomerative Clustering and Ward, deem the side-by-side rods as belonging to the same cluster. And in the fourth dataset, DBSCAN fails to separate neighboring blobs of different variances. Agglomerative Clustering and OPTICS perform particularly poorly on the dataset of ten Gaussian blobs, placing them all in the same cluster. We could not solve this problem no matter how much we tinkered with the hyperparameters. Lastly, the density-based methods DBSCAN and OPTICS fail to label points in 3 of the 6 datasets.

None of the other 11 methods perform particularly well. Like ForestFire, Affinity Propagation and MeanShift do not require the true number of clusters to be known. However, they do not find the correct number of clusters on 3 datasets, and get below 90% accuracy on 4. Birch has similarly low accuracy even though it inputs K as a hyperparameter. As mentioned in Section 2, K-Means and Gaussian Mixture Models are capable of providing a significance metric: Proximity to cluster centroids. However, they are limited in the assumptions they make on the shape of the data. Here, they fail on the concentric circles and interlaced crescents because these shapes cannot be modeled with Gaussian distributions.



	Forest Fire Clustering	Mini-Batch KMeans	Spectral Clustering	Agglomerative Clustering	DBSCAN	Gaussian Mixture	Louvain
Infers cluster number	✓	✕	✕	✕	✓	✕	✓
De-emphasizes noisy outliers	✓	✕	✕	✓	✕	✓	✕
Clusters along data manifold	✓	✕	✓	✓	✓	✕	✓
Non-parametric point-wise significance testing	✓	✕	✕	✕	✕	✕	✕
Detects high dimensionality	✓	✕	✕	✕	✕	✕	✕
Inductive on new data	✓	✕	✕	✓	✓	✓	✕
Number of effective parameters	1	1	2	2	1	2	1

Table 1. Table comparing the features of forest fire clustering with other clustering methods

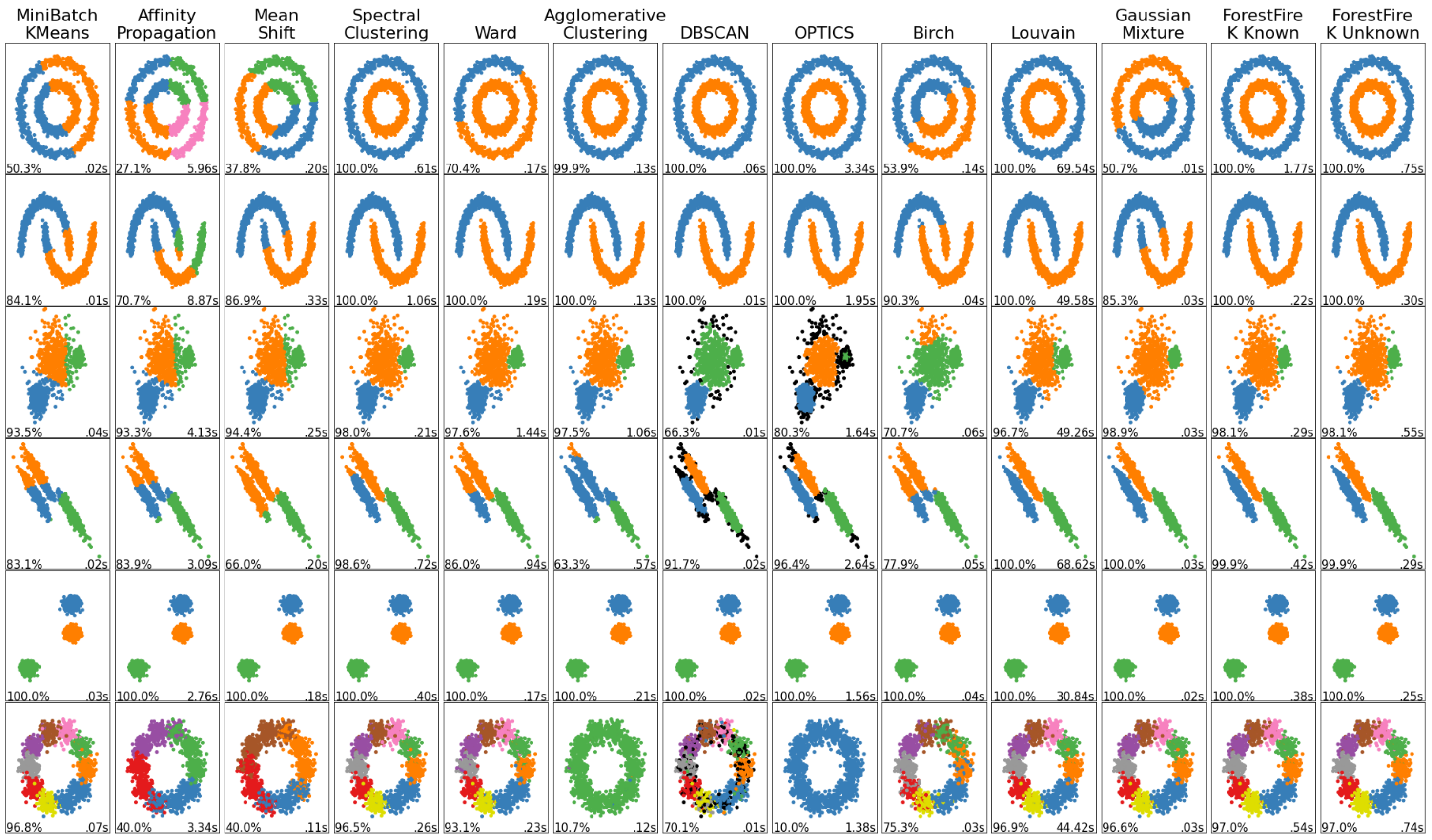


Figure 9: Benchmarking ForestFire Clustering with 11 clustering methods on 6 toy datasets.

3.4 Single-Cell Transcriptomics Data

To demonstrate the real-world utility of our method, we analyzed three single-cell transcriptomic experiments [22–24] (Fig. 10). These datasets all contain genomic measurements of neurons in mouse brains. They posed an interesting challenge because they all contain dozens of clusters, unlike the toy datasets. The high-dimensional genomic features were reduced with tSNE to embeddings we clustered with ForestFire. While the results were strong for all three, they were best for the Shekhar dataset - ForestFire clusters matched the ground truth for around 90% of cells.

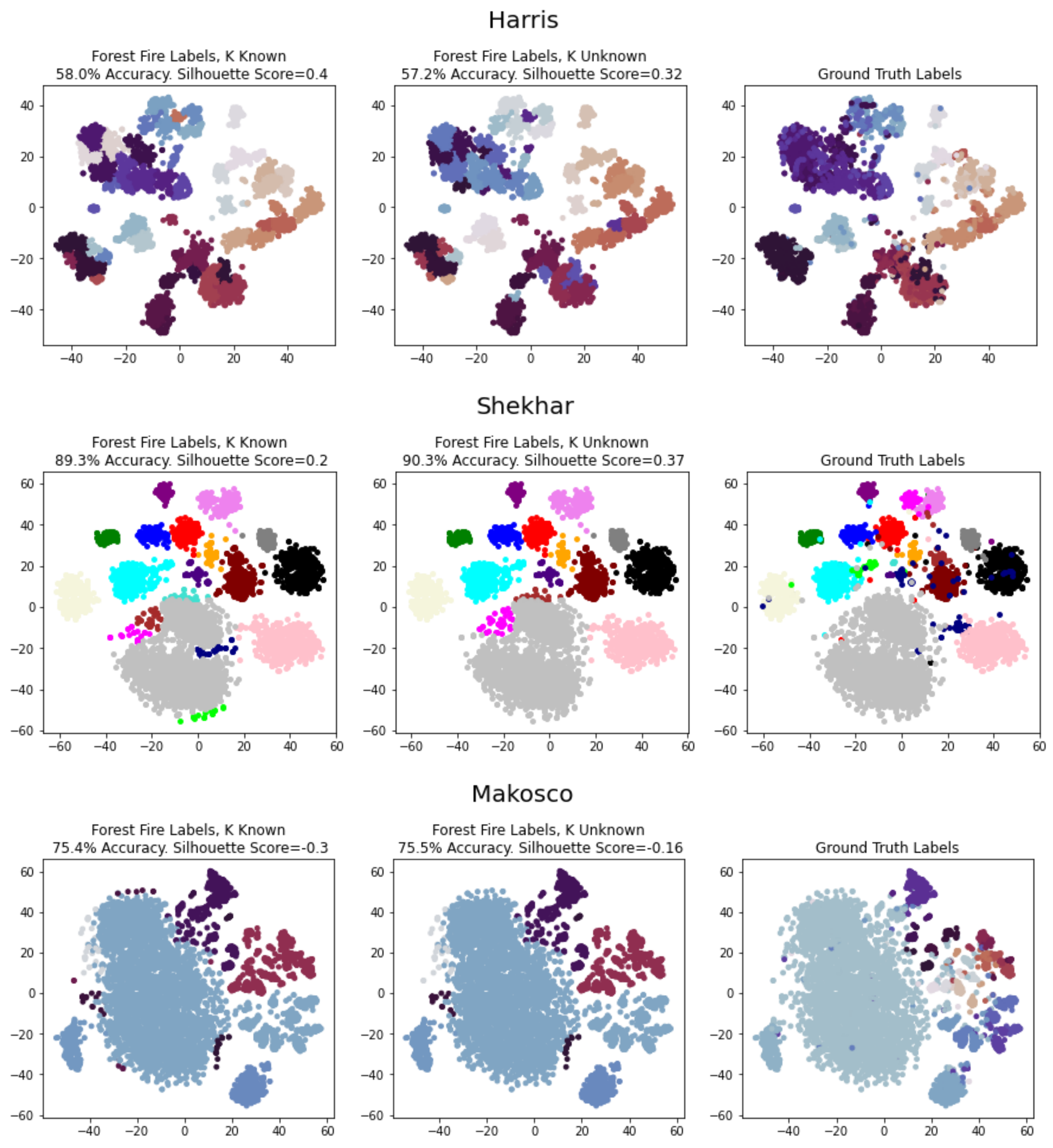


Figure 10. Single-cell RNA-seq Analysis: Single-cell RNA-seq data from Harris et al., Shekhar et al., and Macosko et al. were clustered using ForestFire.

4. Conclusion

Clustering is an unsupervised learning technique that groups unlabeled data points together. Most previous methods require multiple prior assumptions about the data, and do not provide a way to internally validate the results. In addition, the high runtime of Louvain and

Leiden render these state-of-the-art algorithms poorly scalable to large datasets. These are critical shortcomings in applications like single-cell sequencing, which requires clustering methods to efficiently identify doublets and novel cell types.

In this paper, we propose a novel clustering algorithm inspired by forest fire dynamics. Our method, ForestFire, takes a unique label propagation approach, combining principles from hierarchical, spectral, and centroid-based clustering. With effectively only a single hyperparameter, it achieves comparable or improved performance to 11 existing methods on six clustering benchmarks. Moreover, it can use Monte Carlo simulations to produce pointwise confidence scores for its clustering results.

Putting ForestFire in a real-world setting, we found it performs robustly on single-cell transcriptomics datasets. This is extremely promising, as it offers the possibility of generating robust cell type definitions. It also suggests that ForestFire is generalizable to other domains whose clustering problems require flexibility and interpretability.

5. Acknowledgments

I'm immensely grateful for the help of Flynn Chen, whose hard work and imaginative thinking has been truly an inspiration. I would also like to thank Professor Smita Krishnaswamy for kickstarting this project, and Professor Mark Gerstein for seeing it through. Lastly, I'm forever indebted to the S&DS department for its tireless dedication to Yale students - even the ones who take "scandalously" few statistics classes, as Sekhar once lamented about me.

6. References

1. Singh P, Singh M. Fraud Detection by Monitoring Customer Behavior and Activities. International Journal of Computer Applications. 2015;111(11).
2. Shekhar K, Lapan SW, Whitney IE, Tran NM, Macosko EZ, Kowalczyk M, et al. Comprehensive Classification of Retinal Bipolar Neurons by Single-Cell Transcriptomics. Cell. 2016;166(5):1308–1323 e30. doi:10.1016/j.cell.2016.07.054.
3. Harris KD, Hochgerner H, Skene NG, Magno L, Katona L, Gonzales CB, et al. Classes and continua of hippocampal CA1 inhibitory neurons revealed by single-cell transcriptomics. PLoS biology. 2018;16(6):e2006387.
4. Macosko EZ, Basu A, Satija R, Nemesh J, Shekhar K, Goldman M, et al. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. Cell. 2015;161(5):1202–1214.

5. Kiselev VY, Andrews TS, Hemberg M. Challenges in unsupervised clustering of single-cell RNA-seq data. Nat Rev Genet. 2019;20(5):273–282. doi:10.1038/s41576-018-0088-9.
6. Tang F, Barbacioru C, Wang Y, Nordman E, Lee C, Xu N, et al. mRNA-Seq whole-transcriptome analysis of a single cell. Nat Methods. 2009;6(5):377–82. doi:10.1038/nmeth.1315.
7. Shalek AK, Satija R, Shuga J, Trombetta JJ, Gennert D, Lu D, et al. Single-cell RNA-seq reveals dynamic paracrine control of cellular variation. Nature. 2014;510(7505):363–9. doi:10.1038/nature13437.
8. Saadatpour A, Lai S, Guo G, Yuan GC. Single-Cell Analysis in Cancer Genomics. Trends Genet. 2015;31(10):576–586. doi:10.1016/j.tig.2015.07.003.
9. Chen B, Khodadoust MS, Liu CL, Newman AM, Alizadeh AA. Profiling Tumor Infiltrating Immune Cells with CIBERSORT. Methods Mol Biol. 2018;1711:243–259. doi:10.1007/978-1-4939-7493-1-12.
10. DePasquale EAK, Schnell DJ, Van Camp PJ, Valiente-Alandi I, Blaxall BC, Grimes HL, et al. DoubletDecon: Deconvoluting Doublets from Single-Cell RNA-Sequencing Data. Cell Rep. 2019;29(6):1718–1727 e8. doi:10.1016/j.celrep.2019.09.082.
11. McGinnis CS, Murrow LM, Gartner ZJ. DoubletFinder: Doublet Detection in Single-Cell RNA Sequencing Data Using Artificial Nearest Neighbors. Cell Syst. 2019;8(4):329–337 e4. doi:10.1016/j.cels.2019.03.003.
12. Bernstein NJ, Fong NL, Lam I, Roy MA, Hendrickson DG, Kelley DR. Solo: Doublet Identification in Single-Cell RNA-Seq via Semi-Supervised Deep Learning. Cell Syst. 2020;11(1):95–101 e5. doi:10.1016/j.cels.2020.05.010.
13. Ding C, He X. Cluster merging and splitting in hierarchical clustering algorithms. IEEE International Conference on Data Mining. 2002.
14. Hartigan JA, Wong MA. A K-Means Clustering Algorithm. Journal of the Royal Statistical Society. 1979;The 28(1):8.
15. Ester M, Kriegel H, Sander J, Xu X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Association for the Advancement of Artificial Intelligence Proceedings. 1996;.
16. Ankerst M, Breunig MM, Kriegel HP, Sander J. OPTICS: Ordering Points To Identify the Clustering Structure. International Conference on Management of Data. 1999.
17. Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2001. On spectral clustering: analysis and an algorithm. In Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic (NIPS'01). MIT Press, Cambridge, MA, USA, 849–856.
18. Blondel, Vincent D., et al. "Fast Unfolding of Communities in Large Networks." Journal of Statistical Mechanics: Theory and Experiment, vol. 2008, no. 10, Oct. 2008, p. P10008. DOI.org (Crossref), doi:10.1088/1742-5468/2008/10/P10008.

19. Traag VA, Waltman L, van Eck NJ. From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep. 2019;9(1):5233. doi:10.1038/s41598-019-41695-z.
20. Mika S, Sch¨olkopf B, Smola AJ, Mu¨ller KR, Scholz M, R¨atsch G. Kernel PCA and de-noising in feature spaces. In: Advances in neural information processing systems; 1999. p. 536–542.