RNN-DBSCAN Implementation Project Final Report

Thai Flowers

Abstract--In this project we attempted to reimplement the RNN-DBSCAN algorithm and verify the results within the original paper by Avory Bryant and Kryzstof Cios.[1] RNN-DBSCAN is a new DBSCAN like clustering algorithm that is based on reverse nearest neighbor estimates. It addresses two flaws in DBSCAN: it reduces parameter complexity, and it can correctly cluster classes with different densities within epsilon distance from each other. RNN-DBSCAN improves upon existing nearest neighbor techniques in quality of the resulting clustering and in simplicity of implementation. We were successful in implementing RNN-DBSCAN and several of its competitors it was compared against in the original paper, however we were not successful in reproducing the dramatic results in that paper, and the performance of the algorithm is substantively worse than traditional DBSCAN.

1. Introduction

“RNN-DBSCAN: A Density-Based Clustering Algorithm Using Reverse Nearest Neighbor Density Estimates” is a paper published in the June 2018 issue of IEEE Transactions on Knowledge and Data Engineering.[1] It is one of many papers offering various modifications and enhancements to the classic DBSCAN clustering algorithm. A simple search for DBSCAN in the ACM digital library or on the IEEE website yields a flurry of papers describing such algorithms, many of which are quite recent. In fact there are too many to do a proper literature review of the subject, DBSCAN is a venerable and popular algorithm, but it has several flaws that will be discussed in the literature review. The RNN variation was chosen due to its recent publication date, however RNN-DBSCAN does improve the state of the art in clustering algorithms. In particular its handling of density is novel among reverse nearest neighbor clustering algorithms. In this paper we will summarize our work on attempting to reimplement RNN-DBSCAN and verify the results in the original paper. The paper is organized as follows: section 1 is a short introduction leading into the literature survey of section 2. Section 2 is a summary of the literature review given in the original paper which also serves as an introduction to the flaws in existing methods that RNN-DBSCAN attempts to fix. The algorithms described in this section are also those compared against in the original paper’s experiments. Section 3 is a description of the RNN-DBSCAN algorithm. Sections 4 through 6 discuss our implementation, the original paper's experiments, and a comparison of our results respectively. Finally we close with conclusions in section 8.

2. Literature Review

In the original paper six DBSCAN like clustering algorithms are summarized, here we will attempt to summarize them further with an eye towards their problems that RNN-DBSCAN attempts to address. The bulk of the techniques, that is four of the six algorithms presented, are reverse nearest neighbor (RNN) variants of, or competitors to, DBSCAN. The k nearest neighborhood (kNN) of a node n in a graph is the set of k nodes that are closer to n than any of the other nodes in the graph. Where closer is determined by a distance measure, which is typically symmetric (a metric). The reverse nearest neighborhood is the k nearest neighborhood in the transpose of the kNN graph, and can be thought of as the set of nodes S such that node n is in the k nearest neighborhood of every node in S. These definitions will become more clear when we discuss the algorithms that use them.

Before we begin we need to properly introduce what clustering is, and why DBSCAN and it's associates are such a popular research topic. According to the RNN-DBSCAN paper clustering is "an unsupervised pattern recognition problem", that can be "defined generically as grouping data such that observations within a group are similar to each other while being dissimilar to observations within other groups".[1] That is any algorithm that can automatically group data such that similar data appear in the same cluster, and clusters are distinct. Furthermore, in many techniques a model point in the data set can be selected or generated, for each cluster, that can be used to demonstrate the relationship captured by that cluster. Some example types of clustering algorithms include, but are not limited to, grid-based, hierarchical, density, model, and partitioning. From this author's experience, K-means clustering is the archetypal form of clustering, by which we mean that it is the algorithm most commonly associated with clustering in spoken conversation and is featured in most tutorials and articles on the subject. DBSCAN comes in at a close second, and is without a doubt the archetypal density based clustering algorithm. However, despite K-means vaulted status DBSCAN addresses two flaws within K-means, namely DBSCAN can identify clusters of irregular shape, and can determine the number of clusters automatically.[1] K-means in its basic form will attempt to discover K clusters, where K is a user supplied parameter. If the dataset actually has more or less classes (canonical clusters) than that then classes will be incorrectly joined or partitioned respectively. DBSCAN on the other hand is much more flexible than K-means clustering, especially with unusual and/or noisy data distributions, it does have its own flaws that the various RNN algorithms discussed in the original paper attempt to address.

The two main flaws are as follows: Firstly determining the correct values of the two parameters, min_{pts} and eps (epsilon), can be a difficult process. The search space of two parameters is intrinsically larger than the search space of a single parameter k, and there are no good heuristics other than trial and error for determining the best values for clustering a particular dataset. Thus the RNN algorithms discussed shortly are intentionally designed to operate similarly to DBSCAN but only accept a single parameter k so as to restore the convenience of a single parameter algorithm without losing the power of DBSCAN. The RNN algorithms use k to define the threshold for a dense, aka “core” observation in terms of the number of nearest neighbors. OPTICS is especially interesting as it is very similar to DBSCAN, and has similar definitions of “density-reachability” and related concepts as those employed in RNN-DBSCAN. However, OPTICS is not an RNN algorithm, it is an algorithm that can be thought of as running DBSCAN with multiple values of eps simultaneously. The data is stored in an intermediate format and the clustering that would have been created by DBSCAN for a given eps value can be reconstructed from that intermediate data. Thus a user only needs to supply one parameter, the min_{pts}, to the algorithm and the eps parameter can be investigated afterwards without the overhead of running the full algorithm multiple times. [2] Secondly, vanilla DBSCAN assumes a global density and can fail when the density between two classes is smaller than the current eps, leading to the two classes being clustered together. This is easiest to visualize if you consider classes in a two-dimensional data set. There, a class is the canonical clustering that can be visual identified on a graph of the dataset. Attempts to reduce the eps in subsequent runs of the algorithm may lead to one or more of the previously incorrectly joined classes to be identified as noise.[1] RNN methods attempt to address this with definitions of density that are defined locally (that is within each cluster itself).

A more minor flaw that will also be seen is non-determinism of border/outlier points, which are points that are not core observations (in the neighborhood of a core observation but don’t exceed the threshold themselves) or noise.[1] They may belong to more than one cluster in soft-clustering, and in hard-clustering the cluster to which they are assigned depends on data traversal order. This will become clearer after describing a few DBSCAN variants.

This paper uses a graph based interpretation of DBSCAN and associated algorithms for consistency. The data, aka “observations”, form an n-dimensional fully connected graph G(V,E), that the algorithms analyze subgraphs of and consider directionality as necessary for each algorithm’s methodology.

DBSCAN is concerned with the subgraph G_{eps}=(V,E), the directed eps neighborhood graph. V is the set of observations and $\forall$(u,v) $\in$ E, v is in the neighborhood (within eps distance) of u. [1] An observation is core if its neighborhood size is greater than min_{pts}. A cluster is defined by taking an unclustered core observation and adding to the new cluster all nodes reachable by some path such that every node is within the eps neighborhood of the last, and each node in the path (except the last one) must be a core observation. For this process to be deterministic the distance measure must be symmetric. This provides the symmetry necessary to reconstruct the entire cluster from any of the core observations. Border observations are those non-core observations that are placed in a cluster, namely the ones at the ends of the aforementioned paths. The clusters they are added to are determined by the order the nodes are visited. Unclustered nodes are identified as noise.[1]

This algorithm and several of those that follow can be broken down into two key stages: 1) Identify core observations and 2) growing clusters from the core observations. The details of how these tasks are accomplished varies depending on the algorithm used, that is their methods for handling both tasks are what make DBSCAN-esque algorithms distinct from one another.

RECORD is a far more restrictive, and simplistic algorithm. Core observations in a DBSCAN cluster can be viewed as forming strongly connected components in the subgraph G/(V/Core), where Core is the set of all core observations. However, in RECORD that is the definition of a cluster. A cluster is a strongly connected component of that subgraph. There is no notion of border points, they are grouped together with noise under the title “outliers”. Here the graph is G_{kNN}=(V,E) where V is the set of nodes as before, and (u,v)$\in$E if v is a k-nearest neighbor of u. G_{RkNN} is the transpose of G_{kNN} and is the reverse k nearest neighbor graph as explained in the introduction. A node is core if its out-degree in G_{RkNN} is at least k. After clusters of strongly connected components are identified, outliers are processed. An outlier is added to the cluster that makes up a majority of its reverse nearest neighbors, where majority is defined as being greater than k/d where d is the dimensionality of the data.[1] Any remaining outliers are noise, as to be expected, and the cluster assigned to is again non-deterministic.

IS-DBSCAN and ISB-DBSCAN are the final algorithms before we present RNN-DBSCAN. ISB-DBSCAN is a simplification of IS-DBSCAN that removes a pre-processing step known as STRATIFY which removes noise, but is known to be overly aggressive and falsely identify valid data as noise when the level of noise is very low. Here the “influence space” is used for identifying core observations, and is defined as the intersection of the kNN and RkNN graphs (in terms of edges). A node is core if is influence space contains 2k/3 or greater edges. The intersection in the definition of influence space means a symmetric distance measure is not necessary, as intersection is symmetric. However, there is a second problem in both algorithms, the authors argue the ⅔ in 2k/3 acts as an implicit and fixed second parameter that doesn’t work well on all datasets. [1]

3. RNN-DBSCAN

This algorithm has the following benefits: it doesn’t require s symmetric distance measure, it has no hidden parameters, and it automatically handles differential densities between classes correctly. As seen in the literature review the current algorithms each have a flaw in one or more of these regards. It is yet another reverse nearest neighbor algorithm, however, unlike RECORD and IS/ISB-DBSCAN, the core algorithm is nearly identical to vanilla DBSCAN.

In order to understand the algorithm and its features we need to first define a few ideas and notations. We will be brief, for full details and mathematical formulas consult the original paper. Let X represent the set of n data points, each is d-dimensional and real valued. dist(x,y) is the distance between points x and y, this need not be a symmetric measure but in the original paper euclidean distance was used.[1] The nearest neighbor and reverse nearest neighbor algorithms are used, they must be provided with an integer k such that $0 \leq k \leq n-1$. The k-nearest neighborhood of observation x, $N_k(x)$=N where N is a subset of X/{x} of size k, where all the elements in N are closer to x than any other points in X. The reverse k nearest neighborhood is similar, $R_k(x)$=R where R is a subset of X/{x} and $\forall y \in R : x \in N_k(y)$. Like DBSCAN there are three ways to classify data: core, boundary (aka border points), and noise. Here an observation x is core iff $|R_k(x)| \geq k$. [1]

A cluster is defined in terms of its reachability, here we introduce how this is defined. An observation x is directly density-reachable from y if $x \in N_k(y)$ and y is a core observation. In other words points in the nearest neighborhood of a core observation are directly density-reachable from that observation. Density reachability corresponds to the paths of nodes within eps distance of each other in DBSCAN. Here density-reachable means chains of points directly reachable from each other where each point in the chain is a core observation with the possible exception of the final node. Since the nearest neighborhood relationship is non-symmetric then density reachability is not guaranteed to be symmetric, indeed for non-core points it will never be symmetric. To allow symmetric relationships a third definition is need: density connected. Observations x and y are density connected if there is a third point z such that x and y are density-reachable from z.[1]

A cluster C in RNN-DBSCAN is a non-empty subset of X such that all pairs of points in C are density-connected to each other and if x ∈ C and y is density-reachable from x, then y ∈ C. Calculating the density of a cluster is the secret to the effectiveness of this algorithm. Density is calculated with the function den(C)=max$_{(x,y)}$ dist(x,y) where for all x,y ∈ C, x and y are core observations and y is directly density-reachable from x. With this unclustered points are added to clusters yielding "extended" clusters. Unclustered points "within den(C) distance to a core k nearest neighbor" of a cluster are the points added to that cluster, provided there is no point in another cluster that such a point is closer to.[1]

Algorithm 1. $RNN - DBSCAN(X, k)$

```
assign[∀x ∈ X] = UNCLASSIFIED
cluster = 1
for all x ∈ X do
  if assign[x] = UNCLASSIFIED then
    if ExpandCluster(x, cluster, assign, k) then
      cluster = cluster + 1
    end if
  end if
end for
ExpandClusters(X, k, assign)
return assign
```

The algorithm itself is rather simple provided all the definitions above, especially if one is familiar with DBSCAN. The only difference in the base part of the algorithm is the call to ExpandClusters(X,k,assign) after the main loop. This represents a post processing step that expands clusters into extended clusters as described above.

Algorithm 2. $ExpandCluster(\mathbf{x}, cluster, assign, k)$

```
if $|R_k(\mathbf{x})| < k$ then
  assign[x] = NOISE
  return FALSE
else
  initialize empty queue seeds
  seeds.enqueue(Neighborhood(x, k))
  assign[x + seeds] = cluster
  while $seeds \neq \emptyset$ do
    y = seeds.dequeue()
    if $R_k(\mathbf{y}) \geq k$ then
      neighbors = Neighborhood(y, k)
      for all z ∈ neighbors do
        if assign[z] = UNCLASSIFIED then
          seeds.enqueue(z)
          assign[z] = cluster
        else if assign[z] = NOISE then
          assign[z] = cluster
        end if
      end for
    end if
  end while
  return TRUE
end if
```

ExpandCluster is also very similar to the equivalent in DBSCAN. It uses a queue to traverse each neighbors transitively, that is from the initial point it visits neighbors of neighbors and so on. Points are only added to the queue if the visited node is a core observation, thus ending the paths explored at boundary points.

Also note that neighbors in this algorithm include nearest neighbors and core reverse nearest neighbors.

For more details and the ExpandClusters pseudocode consult the original paper.

Algorithm 3. $Neighborhood(\mathbf{x}, k)$

```
$neighbors = N_k(\mathbf{x}) + \{\mathbf{y} \in R_k(\mathbf{x}) : |R_k(\mathbf{y})| \geq k\}$
```

4. Implementation

For this project we choose to use the Java programming language due to its large collection of available libraries and because it is slightly higher level than C/C++. In particular we wanted to use the Apache Commons math library with we have used in previous projects, is somewhat easy to use, and includes a clustering sub-library with DBSCAN presupplied. However the Apache Commons library is out of date and it's documentation doesn't match the current api, instead it is for an a beta of the 4.0 version of the api. So instead the Hipparchus library was used which is a fork of Apache Commons that uses an api similar to a previous version of the beta.

The clustering sub-library was used as a basis for implementing three of the clustering algorithms described in the paper: RNN-DBSCAN, RECORD, ISB-DBSCAN. Each is implemented in a corresponding Clusterer class and is operated analogously to the native Clustering classes, in particular DBSCAN. Code that demonstrates the use of a clusterer can be found in BasicTest and ParameterScan.

Thibault Debatty's java-graphs library we used to implement k nearest neighbor search. The library has methods for constructing nearest neighbor graphs and fetching of the k nearest neighborhood from the graph of a given node. There is considerable overhead, but the library allows the use of several different algorithms for constructing the graph, including BruteForce and NNDescent. We used the threaded version of BruteForce in order to have deterministic results during testing, however NNDescent when used gave decent results (confirmed visually).

BasicTest is a simple visual test that allows a user to compare the DBSCAN performance vs RNN-DBSCAN performance visually on 2d data sets. It is implemented using a JFrame subclass, xyScatterPlot, using JFreeChart to graph the data. It was used to test the efficacy visually while implementing the clustering algorithms. Debugging continued until the code worked on all data sets used and the results were visually similar to the examples provided in the original paper and/or the algorithm's associated paper when such images were available. This class expects input data to have a first line describing the number of data items and dimensionality. All data must be floating point after this.

The main class in this project is ParameterScan. It is used to implement all the experiments attempted. It performs a parameter sweep, k is swept from 1 to 100, and min_{pts} is swept over the set {1, 5, 10, 20} while the eps value is tried for every possible distance between nodes in the k-nearest neighbor graph. For each set of parameter values in the sweep the custerig is performed and analyzed. If the -v flag is provided then each evaluation is printed. Without that flag, just the first value of those parameters that produced the highest ARI or NMI values is printed before exiting. The input file must be in the form of one point per line, tab separated floating point values, one for each dimension of the data, and at the end of the line an integer label of the class or 'noise' as appropriate.

ARI is the adjusted rand index, and it had to be implemented by hand. Formula thanks to Dave Tang and can be found here https://davetang.org/muse/2017/09/21/adjusted-rand-index. Explanation of n parameter was found here: http://faculty.washington.edu/kayee/pca/supp.pdf. It is a measure of similarity between two clusterings, it was used to compare a given clustering in the parameter sweep against the canonical clustering. Testing was performed by comparing results of RNN and RECORD against those in Table 3. Debugging continued until the results became visibly similar. When using NNDescent the values are the same up to two decimal places, however with BruteForce RECORD actually becomes superior to RNN on the aggregate data set. We learned during the implementation of ARI that clusters must be sorted such that the first cluster in the list of clusters is the most similar to the first class in the list of classes and so on. Otherwise the score becomes wildly incorrect, with scores ridiculously outside the -1.0 to 1.0 range that a correct implementation is supposed to output.

NMI is normalized mutual information, it is used in a similar manner to ARI. However the implementation was never successfully completed. NMItest is a small class created to test it. The examples in it are from https://course.ccs.neu.edu/cs6140sp15/7_locality_cluster/Assignment-6/NMI.pdf and https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html was also used to make sense of the algorithm and is the basis of a second incomplete implementation locatable in the ParameterScan source code. Tests from the first source pass but the output values are nowhere near the expected values from the original paper. The other implementation after numerous tweaks gave values either in the correct range but with incorrect results (namely results just keep going up instead of having a proper bell curve), or it gave values close to what is expected near the maximum k value but values wildly outside the correct range elsewhere (both above 1.0 and below -1.0).

Overall when it comes to the implementation I regret several of the decisions made, namely using Hipparchus instead of the Smile mathematics library I discovered later on. Smile comes with a larger set of built-in clustering algorithms and has its own implementation of ARI. However the api is not as friendly for subclassing as Hipparchus/Commons Math is. Alternatively I should have implemented the algorithms in short C/C++ programs and perform the experiments with python, bash or some other scripting language that provides an implementation of ARI, NMI, and other statistics measures. The existing method is very hard to package into a standalone executable and can't be properly scripted as it can only run from within our chosen IDE (IntelliJ).

Also some parts were implemented via bash scripting, namely a short script entitled "convert.sh" which converts files from the format expected by ParameterScan into the format expected by BasicTest, and generate_grid.sh which creates a data set similar to the grid data set in the original paper. The convert.sh script is not perfect and the dimensions must be checked/correct before use.

5. Experiments

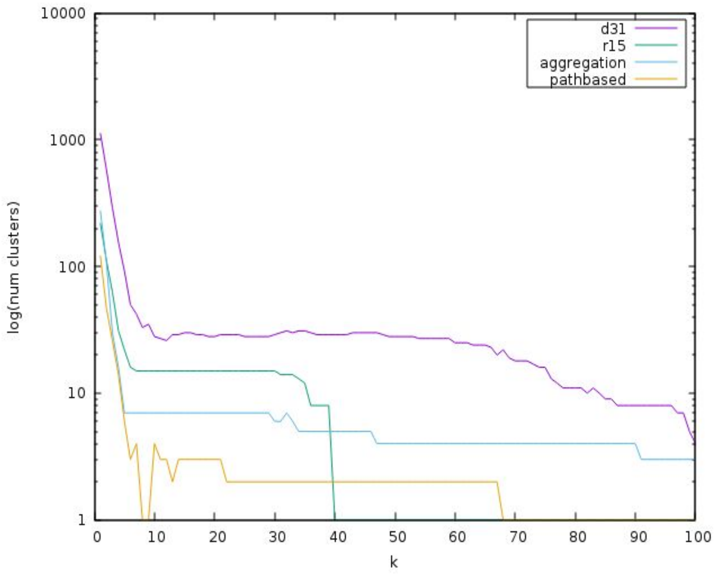
There are five main experiments in the original paper. Firstly a parameter sweep of k from 1 to 100 is performed and the resulting number of clusters is charted for several artificial datasets. The second experiment is much the same, but it charts number of clusters versus frequency of occurrences and ARI performance. The third main experiment is another parameter sweep this time charting k vs ARI and DBCV. DBCV is a complicated statistical measure that measures the "validity index" of a clustering, which is roughly how "tightly connected" a clustering is. A good clustering should have observations that have a high degree of connections to other observations in the same cluster, and a low amount to observations in other clusters. [1] Experiments four and five are parameter sweeps trying to locate the highest possible ARI or NMI value for most artificial and natural data sets used in the experiments.

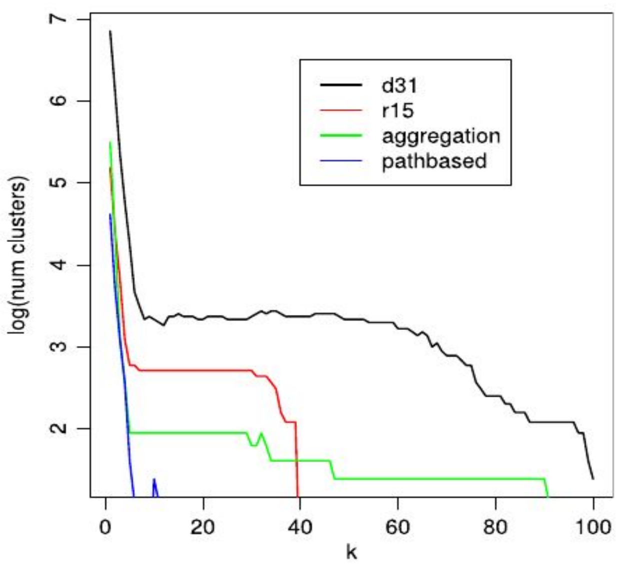
6. Comparison

We performed two of the five experiments, the k vs number of clusters experiment, and the ARI table for artificial data sets. Below we present graphs/tables of our findings next to the original paper's graphs/tables.

We hypothesize that the differences in results are due to five possible causes: 1) We used the brute force implementation of the nearest neighbor graph generation algorithm instead of NNdescent, which produces higher quality results in all RNN based clusterers due to more accurate graphs, and RECORD/ISB-DBSCAN performing better with correct graphs. 2) Errors in our implementations. 3) Misrepresentation of results in the original paper. 4) Unanalyzed impacts due to our and the original paper's choice of implementation language and libraries used. Also we believe that ISB-DBSCAN is most likely implemented incorrectly as it gives visibly poor results when used in BasicTest. 5) Incorrect implementation or use of ARI calculation in our code.

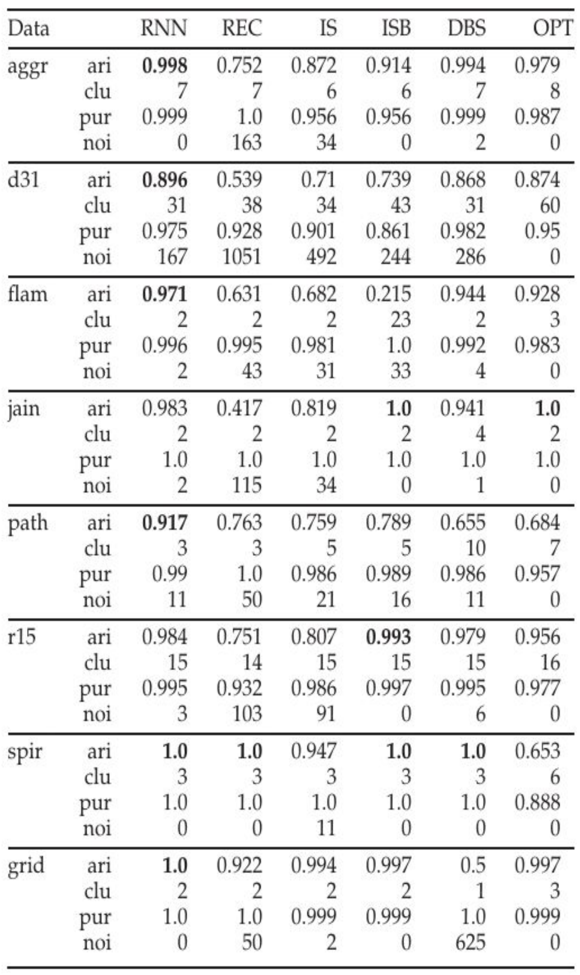
Output from Parameter scan was manually copied from the IDE output into files in /src/main/resources/results where each title describes the corresponding configuration in the IDE and the files with the extension "best" store the final line of output. The table on the left was generated with GNUplot from these files, while the table on the right is from the original paper. Notice the similar curves.





The following table is a reproduction of Table 3, best ARI value and clusters for k between 1 to 100, in the original paper using our implementations for RNN-DBSCAN, ISB-DBSCAN, and RECORD, and Hipparchus library's implementation of DBSCAN.

TABLE 3
ARI Performance on Artificial Datasets

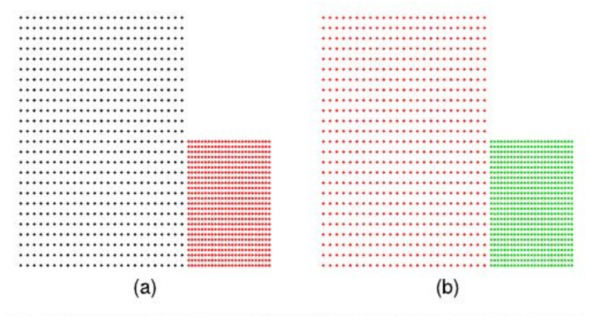


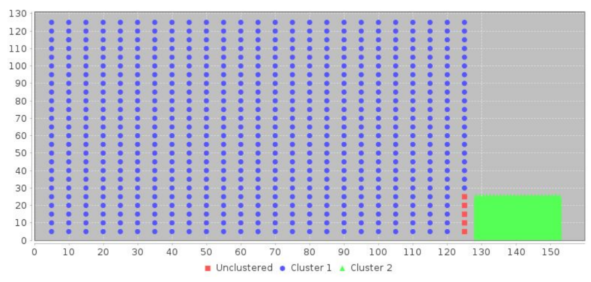
Data		RNN	REC	IS	ISB	DBS	OPT
aggr	ari	**0.998**	0.752	0.872	0.914	0.994	0.979
	clu	7	7	6	6	7	8
	pur	0.999	1.0	0.956	0.956	0.999	0.987
	noi	0	163	34	0	2	0
d31	ari	**0.896**	0.539	0.71	0.739	0.868	0.874
	clu	31	38	34	43	31	60
	pur	0.975	0.928	0.901	0.861	0.982	0.95
	noi	167	1051	492	244	286	0
flam	ari	**0.971**	0.631	0.682	0.215	0.944	0.928
	clu	2	2	2	23	2	3
	pur	0.996	0.995	0.981	1.0	0.992	0.983
	noi	2	43	31	33	4	0
jain	ari	0.983	0.417	0.819	**1.0**	0.941	**1.0**
	clu	2	2	2	2	4	2
	pur	1.0	1.0	1.0	1.0	1.0	1.0
	noi	2	115	34	0	1	0
path	ari	**0.917**	0.763	0.759	0.789	0.655	0.684
	clu	3	3	5	5	10	7
	pur	0.99	1.0	0.986	0.989	0.986	0.957
	noi	11	50	21	16	11	0
r15	ari	0.984	0.751	0.807	**0.993**	0.979	0.956
	clu	15	14	15	15	15	16
	pur	0.995	0.932	0.986	0.997	0.995	0.977
	noi	3	103	91	0	6	0
spir	ari	**1.0**	**1.0**	0.947	**1.0**	**1.0**	0.653
	clu	3	3	3	3	3	6
	pur	1.0	1.0	1.0	1.0	1.0	0.888
	noi	0	0	11	0	0	0
grid	ari	**1.0**	0.922	0.994	0.997	0.5	0.997
	clu	2	2	2	2	1	3
	pur	1.0	1.0	0.999	0.999	1.0	0.999
	noi	0	50	2	0	625	0

Data	RNN	REC	ISB	DBS
aggr ari clu	 0.995 7	 1.0 7	 1.0 7	 1.0 7
d31	0.953 31	0.995 31	0.553 30	0.995 31
flam	0.985 2	1.0 2	0.513 6	1.0 2
jain	0.974 2	1.0 2	1.0 2	1.0 2
path	0.954 3	1.0 3	0.948 2	1.0 2
r15	1.0 2	1.0 15	1.0 15	1.0 15
spir	1.0 3	1.0 3	0.580 6	1.0 3
grid	1.0 2	1.0 2	1.0 2	0.0 0

As can be clearly seen in our experiments RNN-DBSCAN did not perform better than it's competition, in fact it usually did worse than RECORD. As our implementation used brute force calculation rather than approximations of the nearest neighbor graph we believe that RNN-DBSCAN's true advancement is in its ability to degrade gracefully when approximation of nearest neighbors is used. Also we would like to note that the parameter scan for d31 took over 2 hours to complete, despite the fact that each application of DBSCAN took less than a second to complete. So even if RNN-DBSCAN is not an advancement in clustering capabilities, it is a worthy entry in the single parameter family of alternatives to DBSCAN.

The final experiment shown here on the left are the results of DBSCAN and the paper's implementation of RNN-DBSCAN applied to the grid dataset, which works as expected. On the right is RNN-DBSCAN applied to our recreation of grid. It works mostly as expected with a few outlier points that should have been clustered.





7. Conclusions

RNN-DBSCAN is a new reverse nearest neighbors based clustering algorithm meant to be an alternative to DBSCAN and its existing competition. It is an improvement in simplicity of implementation and easily handles differential density between classes correctly. Whereas, the existing algorithms must either have very specific parameters selected in order to handle data sets similar to grid, or can't handle at all (DBSCAN most notably).

RNN-DBSCAN purports to produce better overall results on artificial data sets, however when we performed some of the experiments without approximating nearest neighbors the results were notably worse than the alternatives.

This leads us to the conclusion that it is likely that RNN-DBSCAN's true contribution is not in its ability to cluster but in its ability to cluster well under approximations of nearest neighbor.

[1] A. Bryant and K. Cios. RNN-DBSCAN: A Density-Based Clustering Algorithm Using Reverse Nearest Neighbor Density Estimates. *IEEE Transactions on Knowledge and Data Engineering*, 30(6):1109-1121, June 2018

[2] Mihael Ankerst, Markus M. Breunig, Hanspeter Kriegel, and Jög Sander. Optics: Ordering Points To Identify the Clustering Structure. In , pages 49-60. ACM Press, 1999