Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data

Levent Ertöz
Department of Computer Science
University of Minnesota
Minneapolis, MN USA
ertoz@cs.umn.edu

Michael Steinbach
Department of Computer Science
University of Minnesota
Minneapolis, MN USA
steinbac@cs.umn.edu

Vipin Kumar
Department of Computer Science
University of Minnesota
Minneapolis, MN USA
kumar@cs.umn.edu

ABSTRACT

The problem of finding clusters in data is challenging when clusters are of widely differing sizes, densities and shapes, and when the data contains large amounts of noise and outliers. Many of these issues become even more significant when the data is of very high dimensionality, such as text or time series data. In this paper we present a novel clustering technique that addresses these issues. Our algorithm first finds the nearest neighbors of each data point and then redefines the similarity between pairs of points in terms of how many nearest neighbors the two points share. Using this new definition of similarity, we eliminate noise and outliers, identify core points, and then build clusters around the core points. The use of a shared nearest neighbor definition of similarity removes problems with varying density, while the use of core points handles problems with shape and size. We experimentally show that our algorithm performs better than traditional methods (e.g., K-means) on a variety of data sets: KDD Cup '99 network intrusion data, NASA Earth science time series data, and two dimensional point sets. While our algorithm can find the "dense" clusters that other clustering algorithms find, it also finds clusters that these approaches overlook, i.e., clusters of low or medium density which are of interest because they represent relatively uniform regions "surrounded" by non-uniform or higher density areas. The run-time complexity of our technique is $O(n^2)$ since the similarity matrix has to be constructed. However, we discuss a number of optimizations that allow the algorithm to handle large datasets efficiently. For example, 100,000 documents from the TREC collection can be clustered within an hour on a desktop computer.

Categories and Subject Descriptors

I.5.3 [**Pattern Recognition**]: Clustering

General Terms

Algorithms, Experimentation

Keywords

cluster analysis, shared nearest neighbor, time series, network intrusion, spatial data

1. INTRODUCTION

Cluster analysis [8,11] divides data into groups (clusters) for the purposes of summarization or improved understanding. For example, cluster analysis has been used to group related documents for browsing, to find genes and proteins that have similar functionality, or as a means of data compression. While clustering has a long history and a large number of clustering techniques have been developed in statistics, pattern recognition, data mining, and other fields, significant challenges still remain. In part, this is because large, as well as high dimensional, data sets and the computational power to deal with them are relatively recent. However, most of the clustering challenges, particularly those related to "quality," rather than computational resources, are the same challenges that existed decades ago: how to handle to noise and outliers, how to determine the number of clusters, and how to find clusters with differing sizes, shapes and densities. For this reason, although scalability has been the main focus of much of the clustering work in data mining, clustering research has also focused on these other issues.

1.1. The Challenges of Cluster Analysis and Related Work

Finding clusters of different shapes and sizes, especially in the presence of noise is a problem that many recent clustering algorithms, have addressed. For low dimensional data DBSCAN [3], CURE [5], and Chameleon [10] have shown good performance. Chameleon first builds a list of the nearest neighbors of each point, constructs a weighted similarity graph using this nearest neighbor list, and then partitions the graph to obtain cluster fragments which are merged into clusters with a hierarchical agglomerative clustering technique.

The notion of a representative point is key to DBSCAN, although the term "core point" is used. In DBSCAN, the density associated with a point is obtained by counting the number of points in a region of specified radius around the point. Points with a density above a specified threshold are classified as core points, while noise points are defined as non-core points that don't have a core points within the specified radius. Noise points are discarded, while clusters are formed around the core points. If two core points are neighbors of each other, then their clusters are joined. Non-noise, non-core points, which are called border points, are assigned to the clusters associated with any core point within their radius. Thus, core points form the skeleton of the clusters, while border points flesh out this skeleton.

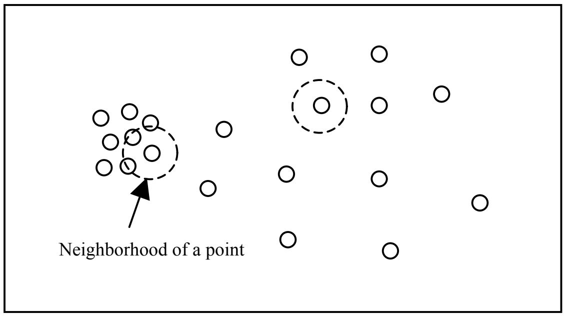


Figure 1. Density based neighborhoods.

While DBSCAN can find clusters of arbitrary shapes, it cannot handle data containing clusters of differing densities, since its density based definition of core points cannot identify the core points of varying density clusters. Consider Figure 1. If the user defines the neighborhood of a point by specifying a particular radius and looks for core points that have a pre-defined number of points within that radius, then either the tight left cluster will be picked up as one cluster and the rest will be marked as noise, or else every point will belong to one cluster.

In CURE, the concept of representative points is also employed to find non-globular clusters. The use of representative points allows CURE to find many types of non-globular clusters. However, there are still many types of globular shapes that CURE cannot handle. This is a result of the way the CURE algorithm finds representative points, i.e., it finds points along the boundary, and then shrinks those points towards the center of the cluster.

While Chameleon does not explicitly use the notion of core points, all three approaches share the idea that the challenge of finding clusters of different shapes and sizes can be handled by finding points or small subsets of points and then building clusters around them. This approach is especially important for spatial data, since non-globular clusters are not represented by their centroid, and thus, cannot be handled by centroid based schemes. Single link agglomerative clustering methods are most suitable for capturing clusters with non-globular shapes, but these methods are very brittle and cannot handle noise properly.

Cure, DBSCAN and Chameleon also gave considerable attention to dealing with noise and outliers. As mentioned, in DBSCAN noise points are not clustered. Cure eliminates noise points by periodically eliminating small groups of points that are not growing very fast. Chameleon does not eliminate any points, but by building the similarity graph from the nearest neighbor list, it does eliminate most of the influence of noise points on the clustering process. Note that outliers are also eliminated by these approaches.

However, both CURE and DBSCAN have problems with clusters of different density. Chameleon can handle clusters of varying density, partly because of its nearest neighbor approach, but does not work well for high dimensional data, e.g., documents.

While DBSCAN, CURE, and Chameleon focused on solving clustering problems for low dimensional data, data of high dimensionality brings new challenges. In particular, the dimensionality of the data typically makes its influence felt through its affect on the similarity function. For example, in high dimensional data sets, distances or similarities between points become more uniform, making clustering more difficult. Also, sometimes the similarity between individual data points can be misleading, i.e., a point can be more similar to a point that "actually" belongs to a different cluster than to points in its own cluster. A shared nearest neighbor approach to similarity, as proposed in ROCK [5] and earlier by Jarvis and Patrick in [9], is a promising way to deal with this issue. Specifically, the nearest neighbors of each point are found, and then a new similarity between points is defined in terms of the number of neighbors they share. Given the new similarity, ROCK performs agglomerative hierarchical clustering , while the Jarvis-Patrick approach simply groups all points with non-zero similarity, i.e., finds connected components.

Our discussion of how several recent clustering approaches handle some of the important challenges of cluster analysis is far from complete. For example, DENCLUE [6] and OptiGrid [7] are more recent density based schemes that are likely to outperform DBSCAN. However, we believe that we have identified the key issues: using representative points to deal with differing shapes and sizes, the difficulty of dealing with clusters of differing densities, the importance of eliminating outliers and noise, and the problems with similarity that can arise, particularly in higher dimensions.

1.2. Our Contribution

Here we present a clustering approach that can simultaneously address several important clustering challenges for a wide variety of data sets. In particular, our algorithm first finds the nearest neighbors of each data point and then redefines the similarity between pairs of points in terms of how many nearest neighbors the two points share. Using this new definition of similarity, we eliminate noise and outliers, identify core points, and then build clusters around the core points. These clusters do not contain all the points, but rather represent relatively uniform groups of points. The use of a shared nearest neighbor definition of similarity removes problems with varying density and the unreliability of distance measures in high dimensions, while the use of core points handles problems with shape and size. Furthermore, the number of clusters is automatically determined, although there are parameters that allow for the adjustment of the algorithm.

A novel aspect of our algorithm is that it finds clusters that other approaches would overlook. In particular, many clustering algorithms only find "dense" clusters. However, this approach ignores sets of points that represent relatively uniform regions with respect to their surroundings. Another novel aspect of our approach is that a cluster consisting of a single data point can be significant, since this data point may be representative of a large number of other data points. (Unfortunately , there is not room in this paper to illustrate this idea and we refer the reader to [4].)

Much of the strength of our approach comes from ideas (core or representative points, defining similarity in terms of near neighbors, noise removal) that are found in several recent clustering algorithms, i.e., CURE, Chameleon, and DBSCAN, although our basic inspiration derives from the Jarvis-Patrick clustering technique, which was proposed in 1973. Our contributions include extending the Jarvis-Patrick clustering technique to encompass the notion of representative points, creating a complete clustering algorithm which incorporates a variety of recent and old ideas, relating this approach to the approaches of other researchers, and importantly, showing that our approach works better than current algorithms for a variety of different types of data.

While our algorithm has many good characteristics, it has a few characteristics that may be liabilities in some situations. First, the algorithm does not cluster all the points. Generally, this is a good thing, as often much of the data is noise and should be eliminated. However, if a complete clustering is desired, the unclustered data can be added to the core clusters found by our algorithm by assigning them to the cluster containing the closest representative point. Secondly, the algorithm is basically partitional, although we have experimented some with producing a hierarchy of clusters. Finally, The run-time complexity is O(n^2), where n is the number of points, since the similarity matrix has to be constructed. However, we discuss a number of optimizations that allow the algorithm to handle large datasets efficiently.

1.3. Outline of the Paper

The rest of the paper is organized as follows. Sections 2 and 3, respectively, describe our approaches to the definition of similarity and density (or connectedness), which are key to our clustering algorithm. The actual clustering algorithm itself is described in Section 4. Section 5 follows up with three case studies: two dimensional point data, NASA Earth Science time series data, and the KDD cup '99 network intrusion data. Section 8 discusses the complexity of our clustering algorithm and strategies for improving the run-time, while Section 7 presents a short conclusion and directions for future work.

2. A BETTER DEFINITION OF SIMILARITY

The most common distance metric used in low dimensional datasets is Euclidean distance, or the L_2 norm. While Euclidean distance is useful in low dimensions, it doesn't work as well in high dimensions. Consider the pair of ten-dimensional data points, 1 and 2, shown below, which have binary attributes.

Point	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
1	1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1

If we calculate the Euclidean distance between these two points, we get √2. Now, consider the next pair of ten-dimensional points, 3 and 4.

Point	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
3	1	1	1	1	1	1	1	1	1	0
4	0	1	1	1	1	1	1	1	1	1

If we calculate the distance between points 3 and 4, we again find out that it's √2. Notice that points 1 and 2 do not share any common attributes, while points 3 and 4 are almost identical. Clearly Euclidean distance does not capture the similarity of points with binary attributes. The problem with Euclidean distance is that missing attributes are as important as the present attributes. However, in high dimensions, *the presence of an attribute is typically a lot more important than the absence of an attribute*, provided that most of the data points are sparse vectors, and in high dimensions, it is often the case that the data points will be sparse vectors, i.e., they will only have a handful of non-zero attributes (binary or otherwise).

Different measures, such as the cosine measure and Jaccard coefficient, have been suggested to address this problem. The cosine similarity between two data points is equal to the dot product of the two vectors divided by the individual norms of the vectors. (If the vectors are already normalized the cosine similarity simply becomes the dot product of the vectors.) The Jaccard coefficient between two points is equal to the number of common attributes divided by the number of attributes in either vector (if attributes are binary). (There is also an extension of the Jaccard coefficient to handle non-binary attributes.) If we calculate the cosine similarity or Jaccard coefficient between data points 1 and 2, and 3 and 4, we see that the similarity between 1 and 2 is equal to zero, but is almost 1 between 3 and 4.

Nonetheless, even though both of these measures give more importance to the presence of a term than to its absence, there are cases where using such similarity measures still does not eliminate all problems with similarity in high dimensions. For example, for several TREC datasets which had class labels, we found that 15-20% of the time a document's most similar document (according to the cosine measure) is of a different class [15]. The "unreliability" of direct similarity is also illustrated in [5] using a synthetic market basket data set. Note that this problem is ***not*** due to the lack of a good similarity measure. Instead, the problem is that direct similarity in high dimensions cannot be trusted when the similarity between pairs of points are low. In general, data in high dimensions is sparse and the similarity between data points, on the average, is low.

Another very important problem with similarity measures in high dimensions is that, *the triangle inequality doesn't hold*. Here's an example:

Point	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
A	1	1	1	1	1	0	0	0	0	0
B	0	0	1	1	1	1	1	1	0	0
C	0	0	0	0	0	1	1	1	1	1

Point A is close to point B, point B is close to point C, and yet, points A and C are infinitely far apart. The similarity between points A and B, and points C and B come from different sets of attributes.

An alternative to direct similarity is to define the similarity between a pair of points in terms of their shared nearest neighbors. That is, the similarity between two points is "confirmed" by their common (shared) nearest neighbors. If point A is close to point B and if they are both close to a set of points C then we can say that A and B are close with greater confidence since their similarity is "confirmed" by the points in set C. The shared nearest neighbor approach was first introduced by Jarvis and Patrick [9]. A similar idea was later presented in ROCK [5].

In the Jarvis–Patrick scheme, a shared nearest neighbor graph is constructed from the similarity matrix as follows. A link is created between a pair of points, p and q, if and only if p and q have each other in their closest k nearest neighbor lists. This process is called k-nearest neighbor sparsification. The weights of the links between two points in the SNN graph can either be simply the number of nearest neighbors the two points share, or can take the ordering of the nearest neighbors into account. Let i

and *j* be two points. The strength of the link between *i* and *j* is now defined as:

$$\textit{strength}(i, j) = \sum (k+1-m)(k+1-n), \text{ where } i_m = j_m$$

In the equation above, *k* is the nearest neighbor list size, *m* and *n* are the positions of a shared nearest neighbor in *i* and *j*'s lists. At this point, clusters can be obtained by removing all edges with weights less than a user specified threshold and taking all the connected components as clusters [9].

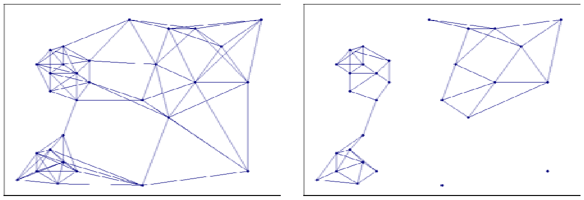


Figure 2. Near Neighbor Graph (left) and Unweighted Shared Nearest Neighbor Graph

Figures 2 illustrates two key properties of the shared nearest neighbor graph in the context of a 2-D point data set. In Figure 2, (left) links to the five most similar neighbors are drawn for each point. Figure 2 (right) shows the unweighted shared nearest neighbor graph. In the graph, there is a link between points A and B, only if A and B have each other in their nearest neighbor lists.

There are two important issues to note in this 2-D point set example. First, noise points and outliers end up having most, if not all, of their links broken. The point on the lower right corner ended up losing all its links, because it wasn't in the nearest neighbor lists of its own nearest neighbors. Thus, by looking at the number of surviving links after constructing the SNN graph, we can remove a lot of noise. Second, the shared nearest neighbor graph is "density" invariant, i.e. it keeps the links in uniform regions and breaks the ones in the transition regions. This is an important property, since widely varying density (tightness) of clusters is one of the hardest problems in cluster analysis.

3. A BETTER DEFINITION OF DENSITY

In high dimensional datasets, the traditional Euclidean notion of density, which is the number of points per unit volume, is meaningless. To see this, consider that as the number of dimensions increases, the volume increases rapidly, and unless the number of points grows exponentially with the number of dimensions, the density tends to 0. Thus, in high dimensions, it is not possible to use a (traditional) density based method such as DBSCAN which identifies core points as points in high density regions and noise points as points in low density regions. (To be fair, it is possible to use DBSCAN if you abandon Euclidean density, i.e., use a similarity measure such as the cosine measure, instead of Euclidean distance.)

However, there is another notion of density that does not have the same problem, i.e., the notion of the probability density of a point. In the k-nearest neighbor approach to multivariate density estimation [2], if a point that has a lot of close near neighbors, then it is more likely to be in a region which has a relatively high probability density. Thus, when we look at the nearest neighbors of a point, points with a large number of close (highly similar) neighbors are in more "dense" regions than are points with distant (weakly similar) neighbors.

In practice, we take the sum of the similarities of a point's nearest neighbors as a measure of this density. The higher this density, the more likely it is that a point is a core or representative points. The lower the density, the more likely, the point is a noise point or an outlier. Note that while our motivation is to identify those points which have the highest probability density, from a graph point of view we are identifying the points that have the strongest connectivity. Also note that since we are using an SNN graph as our starting point, our densities do not correspond to an absolute probability density or connectivity strength, but rather correspond to values that are "normalized" to a local neighborhood.

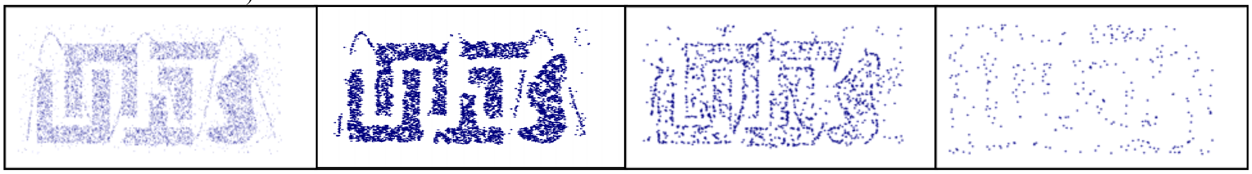
The importance of this approach is that it gives us a density invariant method for identifying core or representative points, which, in turn, are the key to handling clusters of differing sizes and shapes.

4. A SHARED NEAREST NEIGHBOR CLUSTERING ALGORITHM

We now present an algorithm that builds on the Jarvis–Patrick method. This algorithm uses a density based approach to find core or representative points. However, this approach is based on the notion of density introduced in Section 3, which is based on the idea of probability density (or alternatively, the graph connectivity in a similarity graph). However, since we are using similarity based on a shared nearest neighbor approach, which automatically compensates for different densities (see Section 2), this density approach is not be subject to the problem illustrated in Figure 1.

4.1. Identifying Core Points and Removing Noise

Figure 3 illustrates how we can find representative points and effectively remove noise using the SNN graph. In this 2D point dataset, there are 8000 points and a nearest neighbor list of size of 20 is used. Figure 3b shows all the points that have 15 or more links remaining in the SNN graph. In Figure 3c, all points have 10-14 links surviving and Figure 3d shows the remaining points. As we can see in these figures, the points that have high connectivity in the SNN graph are candidates for representative or



a) All points **b) Core Points** **c) Border Points** **d) Noise Points**

Figure 3. Two Dimensional Point Set Example

core points since they tend to be located well inside the natural cluster, and the points that have low connectivity are candidates for noise points and outliers as they are mostly in the regions surrounding the clusters. Note that all the links in the SNN graph are counted to get the number of links that a point has, regardless of the strength of the links. An alternative way of finding representative points is to consider only the strong links in the count. Similarly we can find the representative points and noise points by looking at the sum of link strengths for every point in the SNN graph. The points that have high total link strength then become candidates for representative points, while the points that have very low total link strength become candidates for noise points.

4.2. The SNN Clustering Algorithm

The steps of the SNN clustering algorithms are as follows:

1. Compute the similarity matrix. (This corresponds to a similarity graph with data points for nodes and edges whose weights are the similarities between data points.)
2. Sparsify the similarity matrix by keeping only the *k* most similar neighbors. (This corresponds to only keeping the *k* strongest links of the similarity graph.)
3. Construct the shared nearest neighbor graph from the sparsified similarity matrix. (Each link is assigned a strength according to the formula in Section 3.)
4. For every node (data point) in the graph, calculate the total strength of links coming out of the point. (Steps 1-4 are identical to the Jarvis – Patrick scheme.)
5. Identify representative points by choosing the points that have high density, i.e., high total link strength.
6. Identify noise points by choosing the points that have low density (total link strength) and remove them.
7. Remove all links between points that have weight smaller than a threshold.
8. Take connected components of points to form clusters, where every point in a cluster is either a representative point or is connected to a representative point.

The number of clusters is not given to the algorithm as a parameter. Depending on the nature of the data, the algorithm finds the natural number of clusters for the given set of parameters. Also note that not all the points are clustered using out algorithm. Depending on the application, we might actually want to discard many of the points.

5. EXPERIMENTAL STUDIES

5.1. 2D Data

We first illustrate the superiority of our SNN clustering algorithm with respect to the Jarvis-Patrick approach. A major drawback of the Jarvis–Patrick scheme is that the threshold needs to be set high since otherwise two distinct set of points can be merged into same cluster even if there is only one link between them. On the other hand, if the threshold is too high, then a natural cluster may be split into small clusters due to natural variations in the similarity within the cluster. Indeed, there may be no right threshold for some data sets. This problem is illustrated in the following example that contains clusters of points sampled from Gaussian distributions of two different means and variances.

In Figure 4, there are two Gaussian samples. (Note that these clusters cannot be correctly separated by K-means due to their different sizes and densities). Figure 5 shows the clusters obtained by Jarvis–Patrick method using the smallest possible threshold. (Any threshold smaller than this puts all points in the same cluster.) Even this smallest possible threshold breaks the data into many different clusters. In Figure 5, different clusters are represented with different shapes and colors, where the discarded points / background points are shown as tiny squares. Even with a better similarity measure, it is hard to obtain the two apparent clusters. Figure 6 shows the connected components that result from SNN clustering.

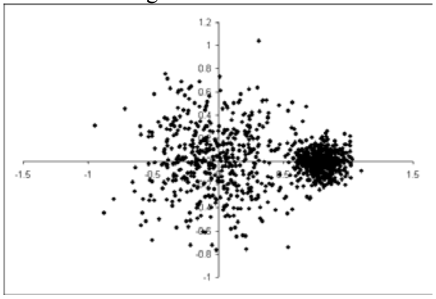


Figure 4. Gaussian Data Set.

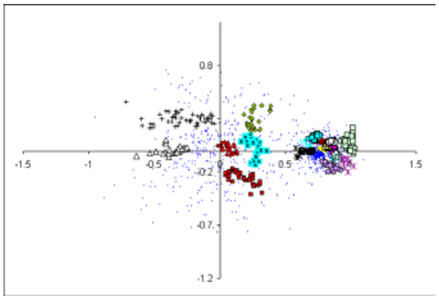


Figure 5. Connected components, JP Clustering.

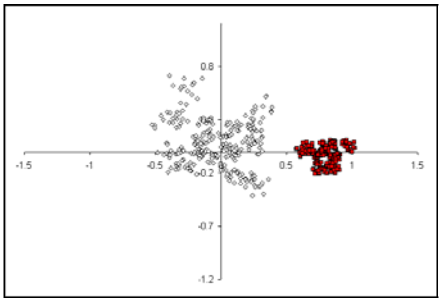


Figure 6. Connected components SNN Clustering.

In order to visualize how the SNN clustering algorithm described in this paper compares to other clustering algorithms, we created a 2-D point dataset and ran our algorithm, as well as CURE, DBSCAN, and K-means, on this dataset. The dataset consists of 6 globular clusters of varying densities. Figure 6 shows the clusters obtained from the Gaussian dataset shown in Figure 4 using the SNN clustering method described here. We can see that by using noise removal and the representative points, we can obtain two clusters. The points that do not belong to any of the two clusters can be brought in by assigning them to the cluster that has the closest core point.

In Figure 7, we see that CURE failed to detect the clusters correctly; it almost identified one cluster correctly (the middle one), but merged the four rightmost clusters into one, and split the leftmost cluster into several parts.

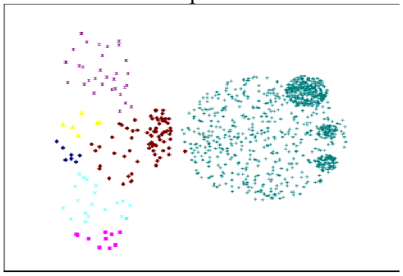


Figure 7. Clusters Produced by Cure.

For one parameterization (shown in Figure 8), DBSCAN found three different clusters, two that were genuine, and one that combined four "real" clusters. For a different choice of parameters (Figure 9), DBSCAN was able to separate the three tightest clusters, but classified everything else as noise. This example clearly illustrates DBSCAN's problems with clusters of varying density.

K-means' performance (Figure 10) was very poor even though the clusters were globular and the right value of *K* was specified. This is due to the fact that the clusters are of very different sizes and densities. Clusters found by K-means were either a mixture of two real clusters or a portion of one real cluster.

Figure 11 shows that our SNN clustering algorithm was able to identify each of the clusters in the dataset, although 15% of the points were not assigned to any cluster.

The performance of our SNN clustering algorithm on the datasets used in CHAMELEON can be found in the appendix.

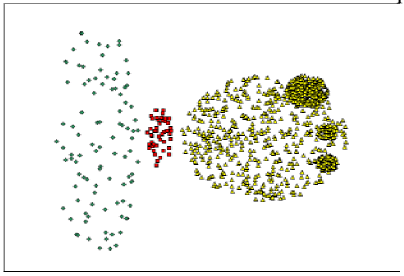


Figure 8. DBSCAN (MinPts=4, Eps=9.75).

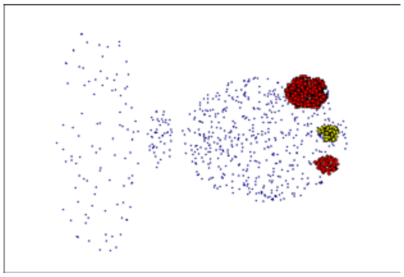


Figure 9. DBSCAN (MinPts=4, Eps=9.92).

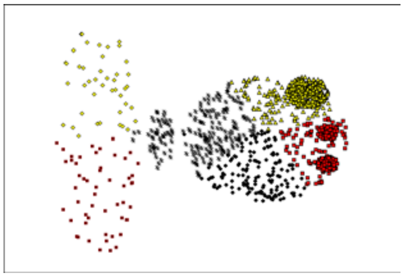


Figure 10. K-means Clustering.

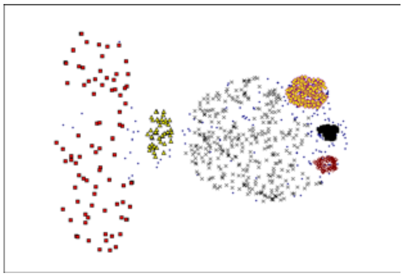


Figure 11. SNN Clustering.

5.2. NASA Earth Science Data

In this section, we consider an application of our SNN clustering technique to Earth science data. In particular, our data consists of monthly measurements of sea level pressure for grid points on a 2.5° longitude-latitude grid (144 horizontal divisions by 72 vertical divisions) from 1950 to 1994, i.e., each time series is a 540 dimensional vector. These time series were preprocessed to remove seasonal variation. For a more complete description of this data and the clustering analysis that we have performed on it, please see [13] and [14].

Briefly, Earth scientists are interested in discovering areas of the ocean, whose behavior correlates well to climate events on the Earth's land surface. In terms of pressure, Earth scientists have discovered that the difference in pressure between two points on the Earth's surface often yields a time series that correlates well with certain weather phenomena on the land. Such time series are called Ocean Climate Indices (OCIs). For example, the Southern Oscillation Index (SOI) measures the sea level pressure (SLP) anomalies between Darwin, Australia and Tahiti and is associated with El Nino, the anomalous warming of the eastern tropical region of the Pacific that has been linked to climate phenomena such as droughts in Australia and heavy rainfall along the Eastern coast of South America [16]. Our goal in clustering SLP is to see if the difference of cluster centroids can yield a time series that reproduces known OCIs and to perhaps discover new indices.

Our SNN clustering approach yields the clusters shown in Figure 12. These clusters have been labeled for easy reference. (Note that cluster "1" is the background or "junk" cluster, and that while we cluster pressure over the entire globe, we focus on the ocean.) Although we cluster the time series independently of any spatial information, the resulting clusters are typically geographically contiguous because of the underlying spatial autocorrelation of the data.

Using these clusters, we have been able to reproduce SOI as the difference of the centroids of clusters 15 and 20. We have also been able to reproduce another well-known OCI, i.e., NAO, which are the normalized SLP differences between Ponta Delgada, Azores and Stykkisholmur, Iceland. NAO corresponds to the differences of the centroids of clusters 25 and 13. For more details, see [14]. This success gives us confidence that the clusters discovered by SNN have real physical significance.

The reader might wonder how we know that these are the "real" set of clusters, i.e., is it possible that the clusters we find might change significantly if we change the parameters for the SNN clustering algorithm. However, that is not the case, and basically the same clusters are produced for different parameterizations. We illustrate why this is so by plotting the SNN density (as discussed in Section 4) at each point – see Figure 13. The densest (reddest, darkest) points correspond to time series that will be the core points, although exactly which ones are chosen will depend on a threshold. The time series merged with any given core point will typically be the time series of the surrounding points since the spatial autocorrelation makes the time series of physically close points be highly similar. Thus, for this data, the SNN clusters can be picked out visually and are in good agreement with those produced by our clustering algorithm.

It is also reasonable to wonder whether other clustering techniques could also discover these clusters. To answer that question, we clustered the same data using DBSCAN and K-means. The best results for DBSCAN clustering are shown in Figure 14, while Figure 15 shows the results for a less optimal parameter choice for DBSCAN clustering. Note that the colors have no meaning except to distinguish the different clusters and are not coordinated between any of the different figures. While there are similarities between the two figures, it seems that to find the less dense clusters, e.g.., 15 and 23, which are the clusters not on the equators or at the poles, it is necessary to relax the parameterization. These less dense clusters are present in Figure 15, but it is also clear that we have "blurred" the other clusters in our attempt to find the "weaker" clusters.

We can see this blurring effect even better if we try to reproduce the SOI and NAO indices using the DBSCAN clusters. For the first set of DBSCAN clusters (Figure 14) we can find a difference of two cluster centroids that correlates with SOI at a level of 0.81, which is essentially the same correlation as the difference of the centroids 15 and 20 from Figure 12. However, no pairs of clusters from Figure 16 yield NAO. If we use the DBSCAN clusters in Figure 15, we can find a pair of clusters that correlates to NAO at a level of 0.77, which is almost as strong a correlation as for the difference of centroids 13 and 25 from Figure 12. However, the correlation to SOI has drops to 0.73. In contrast, SNN clustering can find the less dense clusters without compromising the tighter clusters.

For K-means, we attempted to find roughly the same number of clusters as with our SNN approach, i.e., ~30. (For K-means approach, we used the CLUTO package [1] with the bisecting K-means algorithm and the refine parameter.) However, since K-means naturally clusters all the data, we generated 100 clusters and then threw out all but the 30 most cohesive clusters (as measured by their average cluster correlation). Still, the clusters shown in Figure 16 are not a very good match for clusters of Figure 12, although they do show the same pattern of clusters around the poles and along the equator. Note that cluster 13, which is key to reproducing the NAO index, is missing from the K-means clusters that we kept. Interestingly, one of the discarded K-means clusters did correlate highly with cluster 13 (corr = 0.99), but it is the 57th least cohesive cluster with respect to average cluster correlation.

For both DBSCAN and K-means the main problem is that the "best" clusters are those which are the densest, i.e., tightest. In this Earth science domain, clusters which are relatively dense are also important. (We speculate that regions of the ocean that are relatively uniform have more influence on the land, even if their tightness is not as high as that of other regions.) For this Earth science domain, it is well known that the equatorial SLP clusters correlate highly to the El Nino phenomenon, which is the strongest OCI. However, for K-means and DBSCAN, these "strong" clusters tend to mask the weaker clusters.

5.3. KDD Cup '99 Network Intrusion Data

In this section, we present experimental results from KDD Cup '99 dataset which is widely used for benchmarking network intrusion detection systems. The dataset consists of TCP sessions and the attributes consist of TCP connection properties, network traffic statistics using a 2 second window and several attributes extracted from TCP packets. We also have access to the labels for these sessions which take one of the 5 possible values: normal, u2r, dos, r2l, probe. The original data is very large, therefore we sampled the data using a simple scheme. We picked up all the attack sessions from the training and the test data and put a cap of 10,000 records for each sub-attack type (there are 36 actual attack types in the data that are grouped into 4 categories), and we picked 10,000 normal sessions from both test and training sets. We ended up with a dataset of approximately 97,000 records. We then removed the duplicate sessions and the dataset size reduced to 45,000 records. We clustered the data using our algorithm and k-means (k=100, 300, 1000). The following tables show the purity of the clusters for our algorithm and k-means (k=300). Since our algorithm removes noise, for k-means results we present a similar table constructed from the tightest clusters whose size add up to total size of the shared nearest neighbor clusters. For all of the following comparisons, we used CLUTO to obtain k-way partitioning (using default values), which does a better job than standard implementations of k-means. For comparison, we used Clementine data mining software to cluster the data using k-means and the comparison can be found in appendix.

In the following tables we present results for our algorithm and the results from k-means for 300-way partitioning. The results for the other k-means runs can be found in the appendix. We can clearly see that the level of impurity for k-means clusters is considerably higher than the shared nearest neighbor clusters even when we only consider the tightest k-means clusters. In particular, for the rare class (user to root attack), k-means picks up only 15 out of 267 correctly (using the majority rule) and our algorithm picks up 101 out of 267. When we look at the tightest k-means clusters, k-means doesn't pick up anything from the rare class. The clusters that our algorithm found were superior to k-means clusters almost exclusively for different values of k. For example, when we look at the tightest clusters of the 1000-way partition, k-means was better only in identifying probe class; the impurity of probe class was 5.63% as opposed to 5.95% for our algorithm.

The size of the largest k-means clusters for k=300 was 300 and the largest cluster we obtained using our algorithm was 479 where we had 521 clusters. Even though our algorithm had many more clusters in the output, the largest cluster was much bigger than the largest k-means cluster. This shows that our algorithm is able to find more unbalanced clusters if they're present in the data.

Table 1 Purity across classes - K-means (k=300)

	total	correct	incorrect	Impurity
normal	18183	17458	725	3.99%
u2r	267	15	252	94.38%
dos	17408	17035	373	2.14%
r2l	3894	3000	894	22.96%
probe	4672	4293	379	8.11%

Table 2 Purity across classes - Tightest K-means Clusters (k=300)

	total	correct	incorrect	missing	impurity
normal	18183	9472	520	8191	5.20%
u2r	267	0	113	154	100.00%
dos	17408	16221	186	1001	1.13%
r2l	3894	2569	471	854	15.49%
probe	4672	3610	302	760	7.72%

Table 3 Purity across classes - SNN Clusters

	total	correct	incorrect	missing	impurity
normal	18183	12708	327	5148	2.51%
u2r	267	101	67	99	39.88%
dos	17408	13537	53	3818	0.39%
r2l	3894	2654	257	983	8.83%
probe	4672	3431	217	1024	5.95%

6. COMPLEXITY ANALYSIS

Our SNN clustering algorithm requires O(n^2) time (n is the number of data points), although since it only requires the K nearest neighbors, the space complexity is O(Kn). The K nearest neighbor list can be computed once and used repeatedly, for different runs of the algorithm with different parameter values, but the initial computation of the nearest neighbor list can become a bottleneck. For low dimensional data, the complexity of computing the nearest neighbor list can be reduced to n log(n) by the use of a data structure such as a k-d tree or an R* tree.

While there is no completely general technique for finding nearest neighbors efficiently (in time less than O(n^2)) in high dimensions, the technique of canopies [12] is applicable in many cases. This approach first cheaply partitions a set of data points by repeatedly selecting a point and then forming a group around that point of all other points that are within a certain similarity threshold. The closest of these points are removed from further consideration, and the process is repeated until the set of points is exhausted. (If the actual similarity measure is expensive, an approximate similarity measure can be used to further increase efficiency.) The result of this process is a set of (possibly overlapping) groups of points which are much smaller than the original set and which can then be processed individually. In our case, canopies would reduce the computation required for finding the nearest neighbors of each point since each point's nearest neighbors will only be selected from it's canopy. The rest of the SNN algorithm, which is linear in the number of points, would proceed as before.

While canopies represent a general approach for speeding up nearest neighbor list computation, other, more domain specific approaches are also possible. In the remainder of this section we present a couple of approaches that, respectively, substantially reduce the amount of computation required for documents and for NASA Earth science time series. For documents efficiency is enhanced by using an inverted index and the fact that documents vectors are rather sparse. For times series associated with physical points, efficiency can be enhanced by realizing that the most similar time series are almost always associated with physical points that are geographically close.

For document clustering, every document contains a fraction of the words from the vocabulary and the document vectors are quite sparse. One of the most widely used similarity measures in document clustering is the cosine measure. According to the cosine measure, for two documents to have a non-zero similarity, they have to share at least one word. Thus, for a given document i, we can find the list of documents that have a non-zero similarity to i by keeping a list of all the documents that contain at least one of the terms in document i. The transpose of the document-term matrix (inverted index) consists of word vectors that are lists of documents that contain each word. If we maintain the inverted index for the data we're trying to cluster, then we can construct the list of similar documents for a given document by taking the union of the word vectors for each word in that particular document. After this list is constructed, we only need to calculate the similarity of the given document to the documents in this list since all other documents will have a zero similarity. This avoids unnecessary computation.

A second optimization can be performed by exploiting the fact that for a given document, only the top few words contribute substantially to the norm. If we only keep the high-weighted terms that make up most of the norm (90% or so), we can reduce the number of similar documents considerably, but not lose much information. Using this optimization resulted in a speedup of 30 over the case where only the first optimization was used. We clustered 3200 documents in 2 seconds on a Pentium 3, 600MHz.

For our Earth science data the O(n^2) complexity has not been a problem yet since the current data sets, which consist of data sets of < 100,000 records with dimensionality of roughly 500 can be handled in less than a day on a single processor. However, Earth scientists are collecting data using smaller and smaller grid sizes. For example, if the grid size at which data is collected is halved, the number of data points, n, increases by a factor of 4, and the SNN computation increases by a factor of 16. Thus, in the future computational issues may be more important.

Fortunately it is possible to take advantage of the spatial autocorrelation of the data to significantly reduce the time complexity of SNN to O(n). The basic idea is as follows: because of spatial autocorrelation, time series from two different physical points on the globe that are geographically close tend to be more similar than those from points that are geographically more distant. (Please note the distinction between physical points on the globe and the data points, which are time series associated with those points.) Consequently, if we look at the nearest neighbors (most similar time series) of a particular time series, then they are quite likely to be time series corresponding to points that are spatially very close, and we can compute the nearest neighbor list of a time series – at least to a high accuracy - by looking only at the time series associated with a relatively small number of surrounding locations, e.g., a few hundred. This reduces the complexity of the computation to O(n). Of course, this improvement in complexity is not useful, unless the resulting clusters are about the same. However, in preliminary experiments this seems to be the case. Figure 17 shows the clusters resulting for this "local" SNN clustering approach, which are quite similar to those in Figure 12. For this experiment, we built the nearest neighbor list (of size 100) of each time series by looking only at the time series for the 200 closest physical locations.

7. CONCLUSIONS AND FUTURE WORK

In this paper we described a novel shared nearest neighbor clustering algorithm that can find clusters of varying shapes, sizes, and densities, even in the presence of noise and outliers. The algorithm can simultaneously handle data of varying densities, and automatically determines the number of clusters. Thus, we believe that our algorithm provides a robust alternative to many other clustering approaches that are more limited in the types of data and clusters that they can handle.

In particular, our SNN clustering algorithm can find clusters that represent relatively uniform regions with respect to their "surroundings," even if these clusters are of low or medium density. We presented an example of this in the context of NASA Earth science time series data, where our SNN clustering approach was able to simultaneously find clusters of different densities that were important for explaining how the ocean influences the land. DBSCAN could only find the "weaker" cluster by sacrificing the quality of the "stronger" clusters. K-means could find the weaker cluster, but only as one of a large number of clusters. Furthermore, since the quality of this cluster was low in K-means terms, there was no way to identify this cluster as being anything special.

In addition to the NASA time series data example, we also presented an example of the superior performance of our algorithm on two dimensional point data sets with respect to Jarvis-Patrick, K-means, and DBSCAN. In particular, SNN clustering is consistently able to overcome problems with differing cluster densities that cause other techniques (even Jarvis-Patrick) to fail. For the KDD Cup '99 network intrusion data SNN clustering was able to produce purer clusters than the clusters produced by K-means.

While our clustering algorithm has a basic time complexity of O(n^2), there are a number of possible optimizations that provide reasonable run-time for many domains. The basic idea in all cases is to find a cheaper way of computing the nearest neighbors of a points by restricting the number of points considered. The most general approach to accomplish this is to use method of canopies to split the data into smaller sets and find a points nearest neighbors only among the points in its canopy. However, for documents and Earth science data, several domain specific approaches are possible that can reduce the required computation by one or two orders of magnitude.

The most important goal of our future research will be to investigate the behavior of our SNN clustering approach on other types of data, e.g., transaction data and genomic data. We feel that looking at a wide variety of data sets and comparing the performance of our algorithm to that of other clustering algorithms is the best way to better understand our algorithm and to discover its strengths and weaknesses. Finally, we have made our SNN clustering algorithm publicly available so that others can try it for themselves. It can be download from http://www.cs.umn.edu/~ertoz/snn/

8. ACKNOWLEDGMENTS

This work was partially supported by Army High Performance Computing Research Center cooperative agreement number DAAD19-01-2-0014. The content of this work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by the AHPCRC and the Minnesota Supercomputing Institute.

9. REFERENCES

[1] CLUstering Toolkit. http://www-users.cs.umn.edu/~karypis/

[2] Richard Duda, Peter Hart, and David Stork, Pattern Classification, Wiley-Interscience (2001).

[3] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," KDD 96, Portland, OR, pp. 226-231 (1996).

[4] L. Ertöz, M. Steinbach, and V. Kumar, " A New Shared Nearest Neighbor Clustering Algorithm and its Applications," Workshop on Clustering High Dimensional Data and its Applications," Second SIAM International Conference on Data Mining, Arlington, VA, (2002).

[5] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes," In Proceedings of the 15th International Conference on Data Engineering (1998).

[6] Alexander Hinneburg and Daniel. A. Keim, "An Efficient Approach to Clustering in Large Multimedia Databases with Noise," Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, AAAI Press (1998).

[7] Hinneburg and D. Keim. "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering." VLDB '99, Edinburgh, Scotland (1999).

[8] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall (1988).

[9] R. A. Jarvis and E. A. Patrick, "Clustering Using a Similarity Measure Based on Shared Nearest Neighbors," IEEE Transactions on Computers, Vol. C-22, No. 11, November (1973)

[10] George Karypis, Eui-Hong Han, and Vipin Kumar, **"**CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling," IEEE Computer, Vol. 32, No. 8,. pp. 68-75, August (1999).

[11] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley and Sons (1990).

[12] A, McCallum, K. Nigam, L. Ungar, "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching," KDD 2000, pp. 169-178 (2000).

[13] M. Steinbach, P. N. Tan, V. Kumar, C. Potter, S. Klooster, A. Torregrosa, "Clustering Earth Science Data: Goals, Issues and Results", In *Proc. of the Fourth KDD Workshop on Mining Scientific Datasets* (2001).

[14] M. Steinbach, P. N. Tan, V. Kumar, C. Potter, S. Klooster, A. Torregrosa, "Data Mining for the Discovery of Ocean Climate Indices", *Mining Scientific Datasets Workshop, 2nd Annual SIAM International Conference on Data Mining, April* (2002).

[15] Michael Steinbach, George Karypis, and Vipin Kumar, "A Comparison of Document Clustering Algorithms," KDD-2000 Text Mining Workshop, (2000).

[16] G. H. Taylor, "Impacts of the El Niño/Southern Oscillation on the Pacific Northwest" (1998) http://www.ocs.orst.edu/reports/enso_pnw.html

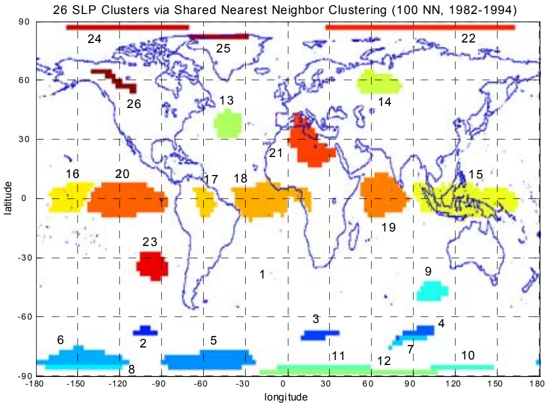


Figure 12. SNN Clusters of SLP.

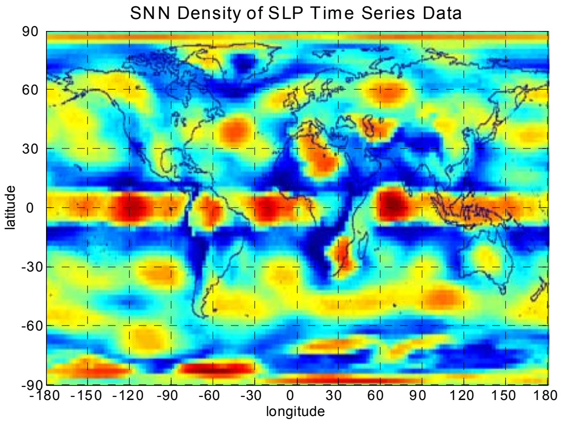


Figure 13. SNN Density of Points on the Globe.

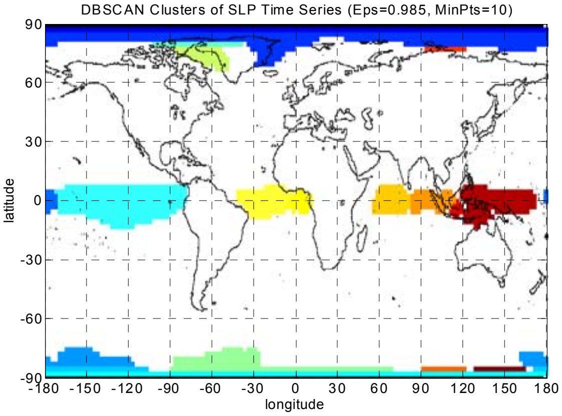


Figure 14. DBSCAN Clusters of SLP.

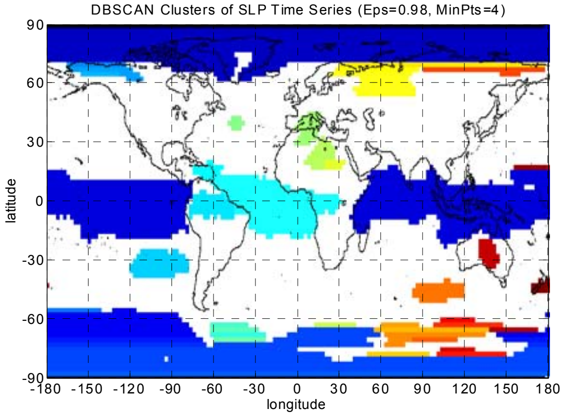


Figure 15. DBSCAN Clusters of SLP.

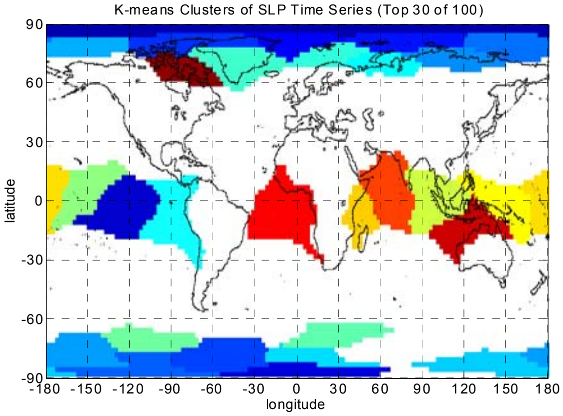


Figure 16. K-means Clusters of SLP.

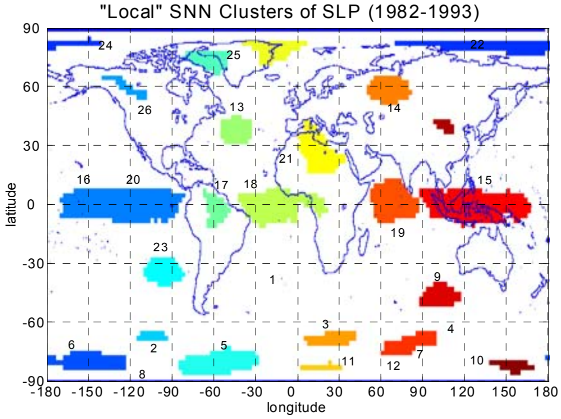


Figure 17. "Local" SNN Clusters of SLP.

APPENDIX

A.1 KDD Cup '99 Data

Following tables show the performance of K-means clustering on the network intrusion data. The results for k=300 were presented in the paper. The results for k=100, 1000 are presented here in the appendix as well as the results for a standard implementation of K-means for k=300.

Purity across classes - K-means (k=100)

	total	correct	incorrect	impurity
normal	18183	17014	1169	6.43%
u2r	267	0	267	100.00%
dos	17408	16620	1188	6.67%
r2l	3894	2840	1054	27.07%
probe	4672	4309	363	7.77%

Purity across classes - Tightest K-means Clusters (k=100)

	total	correct	incorrect	missing	impurity
normal	18183	12817	512	4854	3.84%
u2r	267	0	102	165	100.00%
dos	17408	14346	305	2757	2.08%
r2l	3894	1179	390	2325	24.86%
probe	4672	2948	192	1532	6.11%

Purity across classes - K-means (k=1000)

	total	correct	incorrect	impurity
normal	18183	17504	679	3.73%
u2r	267	91	176	65.92%
dos	17408	17231	177	1.02%
r2l	3894	3238	656	16.85%
probe	4672	4348	324	6.93%

Purity across classes - Tightest K-means Clusters (k=1000)

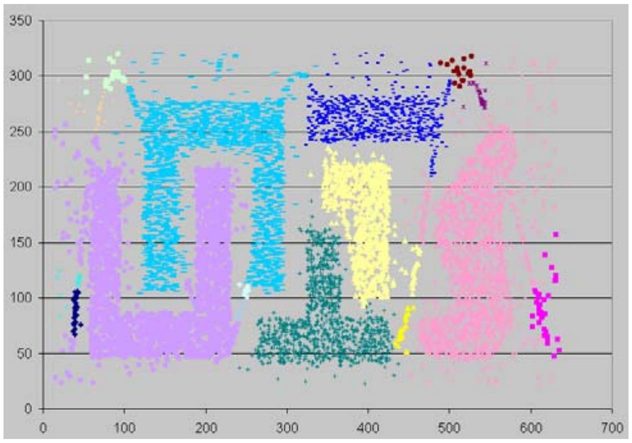
	total	correct	incorrect	missing	impurity
normal	18183	8293	448	9442	5.13%
u2r	267	68	132	67	66.00%
dos	17408	16664	122	622	0.73%
r2l	3894	2533	511	850	16.79%
probe	4672	4291	256	125	5.63%

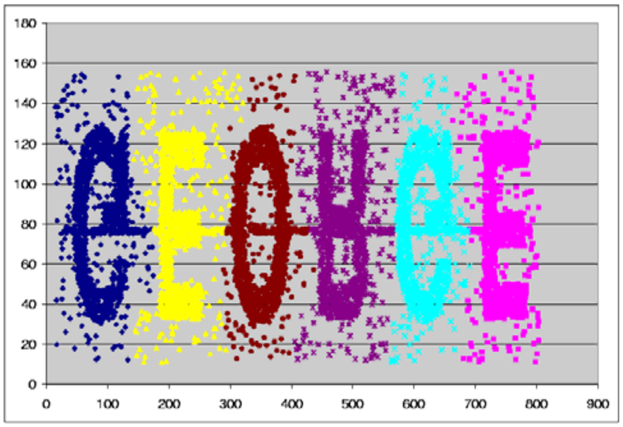
Purity across classes - Clementine K-means (k=300)

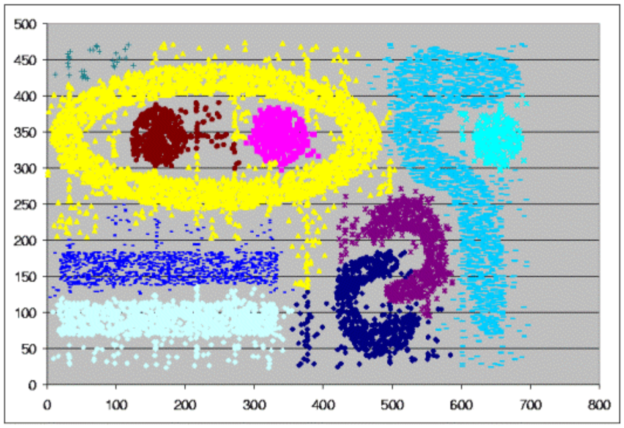
	total	correct	incorrect	impurity
normal	18183	17222	960	5.28%
u2r	267	81	186	69.66%
dos	17408	16373	1035	5.95%
r2l	3894	1694	2200	56.50%
probe	4672	4198	474	10.15%

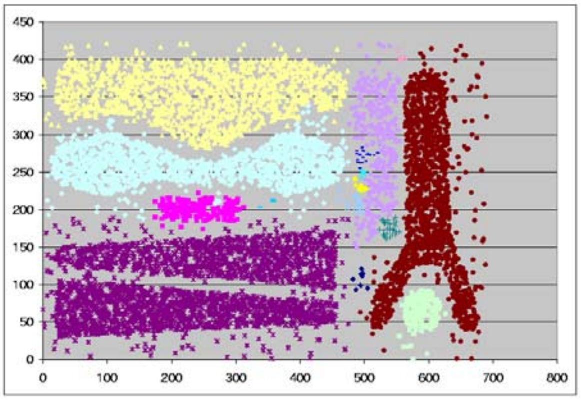
A.2 Chameleon Datasets

Following figures show the performance of our algorithm on the datasets used in Chameleon






500
450
400
350
300
250
200
150
100
50
0
0
100
200
300
400
500
600
700
800


450
400
350
300
250
200
150
100
50
0
0
100
200
300
400
500
600
700
800