SUBCLU: Implementation and Analysis

Micah Nacht and Nick Spinale
Data Mining
Carleton College

May 31, 2017

1 Introduction

We chose to implement SUBCLU, an algorithm for density-based clustering in subspaces of high-dimensional data. It is built upon an older density-based clustering algorithm called DBSCAN, and is effective in clustering high-dimensional data.

We implemented SUBCLU, and analyzed it using an interactive visualizer we created and a data set containing soccer players' statistics and preferred positions. This paper details our implementation, exploration, analysis, and what we learned along the way.

1.1 DBSCAN

SUBCLU relies heavily on a widely-used clustering algorithm originally invented in 1996 called DBSCAN (Ester et al. 1996). DBSCAN groups points in densely-packed regions together. More specifically, it classifies points as core points, border points, or noise according to two parameters, ϵ and `minPts`. A point is a core point if its ϵ-neighborhood contains at least `minPts` points. Points ϵ-close to core points that are not themselves core points are called border points. Points that are neither core points nor border points are called noise.

Two points are density-connected if there is a chain of ϵ-close points between them where all interior links are core points. Intuitively, one can travel between density-connected points by taking short hops in a dense region. A density-connected set is a set of points that are mutually density-connected, and a cluster is a maximal density-connected set. DBSCAN finds clusters by expanding core points into the maximal density-connected sets that contain them until the core points have been exhausted

1.2 SUBCLU

As we show later in this paper, DBSCAN is effective in low-dimensional spaces. However, the phenomenon known as the 'Curse of Dimensionality' causes it to be less effective in higher-dimensional spaces. In high-dimensional spaces, almost all points are equally far apart, so distinguishing clusters from noise becomes very hard.

By considering only certain dimensions of a high-dimensional space, meaningful clusters can be identified. A naive approach to identifying clusters in subspaces (subsets of the entire set of dimensions) would be exponential with respect to the number of dimensions, which is unfeasible. However, in their 2004 paper, Kailing et al. prove the monotonicity of density connectivity (Kailing, Kriegel, and Kröger 2004). They show that if a set of points is density connected (i.e. every pair of points in the set are density-connected) in some space, then that set is density connected in every subspace of that space. This observation allows for an *A-priori*-like bottom up approach for discovering clusters using DBSCAN in each subspace of a high-dimensional space.

The algorithm they named SUBCLU uses DBSCAN to find clusters in each 1-dimensional subspace, and then recursively searches for clusters in k-dimensional subspaces for each k from 2 by only clustering in sets of points that are clusters in all direct subspaces of the subspace being searched. This algorithm is very similar to the *A-priori* approach to finding frequent item sets and association rules, replacing the monotonicity of frequency and confidence with the monotonicity of density-connectivity.

2 Implementation

We implemented SUBCLU in Java. In doing so, we encountered both the strengths and limitations of the algorithm. Here we describe some of the implementation decisions we made, and what these decisions revealed about DBSCAN and SUBCLU.

The efficiency of DBSCAN is based in part on the efficiency of the system used to make geometric queries on the database. Specifically, DBSCAN requires a data structure called a spatial index, which efficiently finds all points within a given ϵ-neighborhood of a given point. Data structures such as R-Trees and $R*$-Trees are commonly used for this purpose (Ester et al. 1996).

However, the construction of these structures is expensive, and a particular instance can only perform queries in the entire space, not arbitrary subspaces. Thus, SUBCLU cannot take full advantage of these specialized structures (Kailing, Kriegel, and Kröger 2004). Our implementation constructs an index structure that can find ϵ-neighborhoods in each dimension (implemented as a Red-Black Tree, in this case called a Range Tree), and then aggregates the results to filter further. The fact that SUBCLU cannot take full advantage of specialized index data structures is certainly a limitation.

In their 2004 paper, Kailing et al. suggest a way of finding candidate subsets of points in a k-space which are to be searched for clusters which is similar to that presented in class for finding candidate frequent item sets. Roughly, for finding candidate subsets of points to be searched for clusters in all $k+1$-subspaces, they suggest finding candidate $k+1$-subspaces by filtering unions of k-subspaces matching on the first $k-1$ dimensions, and then, for each candidate $k+1$-subspace, clustering only the intersection of all the clusters found in each k-subspace of that $k+1$-subspace.

Our implementation fuses these discrete steps together using a Trie (implemented as a Hash Tree) and an interesting traversal. Each step from k-subspaces to $k+1$-subspaces corresponds to extending the Trie to another level, only attempting to add $k+1$-subspaces which could possibly have clusters by the *A-priori* principle, and only actually adding candidate $k+1$-subspaces which have clusters.

Unfortunately, space does not permit a detailed description of this approach, but the relevant parts of the source attempt to explain it. It is not crucial to do so, however, because, while our approach here is more efficient than the one presented in the paper, the running time of this part of the algorithm is dominated by other parts, such as ϵ-neighborhood queries, so our change does not effect the running time of the algorithm as a whole. Nevertheless, spending more time on the implementation of this *A-priori* procedure showed us the power of a monotonicity principle in making what would be an exponential operation feasible.

3 Analysis

We used our implementation to explore SUBCLU's uses, strengths, and limitations. Here we present how we experimented with SUBCLU, and what we learned as a result.

3.1 Visualization

We built a desktop application that visualizes the results of SUBCLU to aid in our exploration. Its usage and features are described in detail in the README distributed with this paper. In the simples case, the user can draw points in 2-space, and see the result of running SUBCLU on the points. The application shows the clustering in 2-space, and each single dimension, along with the ϵ-neighborhoods of each clustered point.

This application allowed us to visually test our implementation, and see for ourselves some of the strengths and limitations of DBSCAN and SUBCLU. We now present a representative sample of these informative visualizations. Note that cluster colors are local to subspaces only. That is, a green cluster in the center pane does not necessarily correspond to a green cluster in the left pane. Colors are only used to distinguish clusters within each subspace.

Figure 1 (found at the end of the paper) shows DBSCAN successfully identifying nested clusters. k-Means would not be able to identify these clusters, as partitioning space using centroids does not allow for such nested structures.

Figure 2 demonstrates the principle of the monotonicity of density-connectivity. The set of points that are a cluster in 2-space are also all part of the same clusters in each singleton subspace (the left and bottom of the figure). However, there is a cluster (and thus a density connected set) in the x singleton dimension (green) that is not a cluster (and thus not part of a density connected set) in 2-space, showing that the implication only goes in one direction.

Figure 3 is an example of SUBCLU identifying a cluster hidden in a subspace. 2-space looks like noise, but its projection onto the x-axis reveals a tight cluster.

Figures 4 and 5 show a limitation of DBSCAN (and thus of SUBCLU also). This database is generally more dense in the bottom-right region than the top-left region. So, the cluster in the top-left is about as dense as the noise in the bottom-right. Certain density parameters ϵ and $minPts$ (e.g. those used in Figure 4) correctly identify the two clusters in the more dense bottom right, but categorize the top-left cluster as noise. Other parameters (e.g. those used in Figure 5) capture the cluster in the top-left, but lump the two in the bottom-right as one. Our visualizer doesn't have a k-means implementation, but k-means with $k = 3$ would likely have partitioned 2-space around those clusters correctly.

3.2 Runtime comparison to k-Means

We have already discussed some differences between k-means and DBSCAN concerning the sorts of layouts of data in space for which each fails to identify meaningful clusters. However, we were also curious about the difference in runtime between the two algorithms. Making this comparison is complicated, as each algorithm takes different parameters and has output with different meaning. Nevertheless, an idea of which algorithm would generally take longer to complete with reasonable parameters could be useful in situations where one seeks to do some preliminary analysis of some data, and just wants some information quickly.

Given the fuzziness of this runtime comparison and the fact that this was not the focus of our project, we did not dive to deeply looking for an answer, but rather just ran both on the same reasonably-sized data set with reasonable parameters. The data was 6500 rows draw randomly from 8 Gaussian clusters (Rezaei and Fränti 2016).

The k-means implementation one of us (Nick) submitted for Homework 6 took 15.421 seconds to run with $k = 8$. Our implementation of DBSCAN, however took only 1.795 seconds to cluster the same data with sane values of ϵ and *minPts* (5000 and 10 respectively). Of course, these values mean nothing without the data, but they yielded 8 clusters, which is equal to the ground truth. A more in-depth parameter analysis would be a valid subject for further research.

As mentioned before, comparing these algorithms is hard, but this single test suggests that, for tame databases (such as one with $N = 6500$), DBSCAN is far faster than k-means. This is consistent with intuition, as k-means may have to make many passes through the data, while DBSCAN (when supported by a good indexing data structure) only need to make one pass through the entire database.

3.3 FIFA Dataset

To see how well SUBCLU works, we wanted to run it on a real dataset with some notion of ground truth. We chose a set of players from FIFA 17, with the idea that players in the same position would be clustered together in an effective clustering.

3.3.1 Questions and Procedure

Given a dataset of soccer player attributes, described below, we wanted to run the SUBCLU algorithm and see how well the clusters correlated to real positions of said players. We ran the SUBCLU algorithm, printing to a log each space in which a cluster was found, and the contents of each cluster. From this, we were able to find a single-position similarity measure for each cluster:

$$\max_{\text{positions}} \frac{\text{number of players that play position in cluster}}{\text{size of cluster}}$$

We also examined a more general single-type similarity measure, defined as

$$\max_{\text{types}} \frac{\text{number of players that play type of position in cluster}}{\text{size of cluster}}$$

where a type is simply a set of positions. For example, the defensive players type includes RB, LB, CDM, and CB. High similarities imply good clusterings, because we expect that players who play similar positions are similar to each other.

3.3.2 Dataset

The dataset includes over 17,000 players, with 53 different attributes. We narrowed this down to approximately 40 soccer-relevant attributes. We chose to ignore redundant, non-numerical categories such as nationality, club, and a few others. We also removed position for national team, and position for club team, because a third category, preferred position, captured the actual information about where the player is the most comfortable.

In order to run the analysis in a reasonable amount of time, we selected a random 10,000-player subset of players who had a single preferred position. Some players had multiple preferred positions– Christiano Ronaldo, for example, is listed as “LW/ST.” To make our similarity measures easier to calculate, we eliminated these players from our consideration. This could be something to look at in the future.

3.3.3 Results

Clustering the data took a while to run, despite the fact that we were working with about half of the full dataset. The results were greatly dependent on the parameters we used. As the ϵ-neighborhood increased, the largest space in which there were clusters found also increased, but so did the number of clusters found in lower spaces (and therefore the runtime of the entire algorithm). The rest of this analysis focuses on the results with parameters $\epsilon = 10, \text{minPts} = 200$. It is important to note that DBSCAN does not have helpful metrics, unlike k-Means, in order to create quantitative results. The original DBSCAN paper used pictures of the clusters, which is possible in 2-D cases but not in higher dimensions. With this dataset, we can use the similarity metrics discussed above as a proxy for clustering quality. This is obviously not ideal, and research into accuracy metrics would be a helpful next step.

The first interesting result was the reliable clustering of GK and CB into reliably consistent clusters in logical spaces. For example, goalkeepers were clustered together in the space {Marking, Heading}, which makes sense because goalies never need to do either of those things. At the same time, under the lens of single-position similarity, there were no consistent striker clusters, midfield clusters, or outside defender clusters. This is likely because these positions are a little more fluid, with skills that are relevant to a CM also being relevant to a CAM.

Thus, we also looked at the results under the lens of the single-type similarity measure. Using the same clustering as above, 51% of the clusters had a single-type similarity measure of $\geq .8$. We selected 0.8 pretty arbitrarily, but the trend we are about to describe also appears with cutoff values ranging from 0.4 to 0.9. In figure 6, we show the increase in this percentage as we truncate the subspaces of lower dimensions. Similarly, figure 7 shows that this is due to increased accuracy in higher dimensions. This implies that SUBCLU searches the feature space effectively, since more features leads to a more expected clustering of the data.

Overall, it seems as if this algorithm worked well on this dataset. Unfortunately, SUBCLU takes a while to run and we would like to do a larger parameter sweep in order to find the best parameters for finding clusters of players. Additionally, it could be nice to examine the similarity metrics themselves and see if they are generalizable.

4 Next steps

There is certainly room for more analysis on SUBCLU. For one, some kind of metric of DBSCAN clustering efficacy would be incredibly helpful in determining how effective SUBCLU is. Measures such as SSE are not particularly applicable to DBSCAN clusters, as they are not “centered” around a centroid like the circular k-means clusters are. Further analysis of the efficacy of single-type similarity and single-position similarity may be a good path to begin down.

We might also want to perform a larger parameter sweep and analysis on the FIFA data. The data we gathered shows promise for future results with clusterings, but getting an optimal clustering was outside the scope of this paper due to time constraints. On the topic of parameters, we also would like to do a more in-depth analysis of the relationship between parameters and runtime. The structure of the data has a large effect on how quickly the algorithm runs, so analyzing different datasets could also prove helpful.

References

Ester, Martin et al. (1996). “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *Kdd*. Vol. 96. 34, pp. 226–231.

Kailing, Karin, Hans-Peter Kriegel, and Peer Kröger (2004). “Density-connected subspace clustering for high-dimensional data”. In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, pp. 246–256.

Rezaei, M. and P. Fränti (2016). “Set-matching methods for external cluster validity”. In: *IEEE Trans. on Knowledge and Data Engineering* 28.8, pp. 2173–2186.

List of Figures

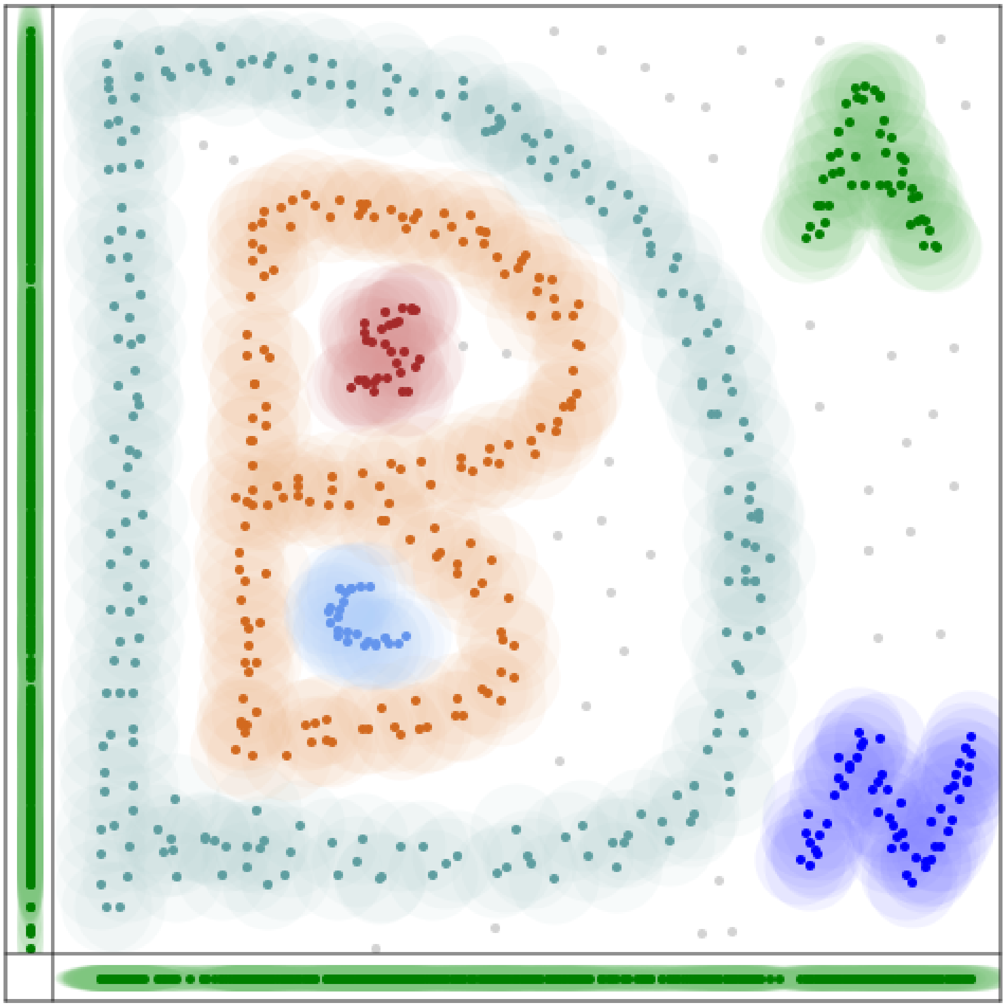

Figure 1: Nested clusters

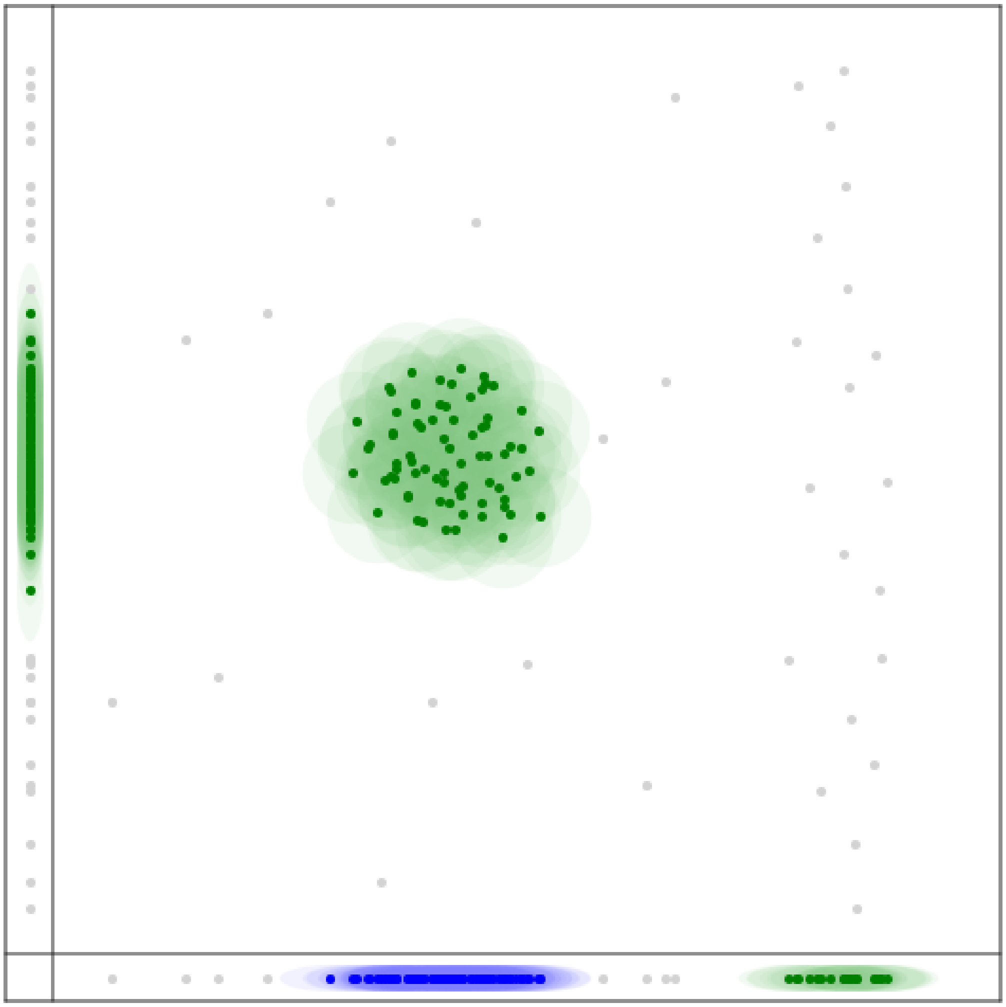

Figure 2: Monotonicity of density-connectivity

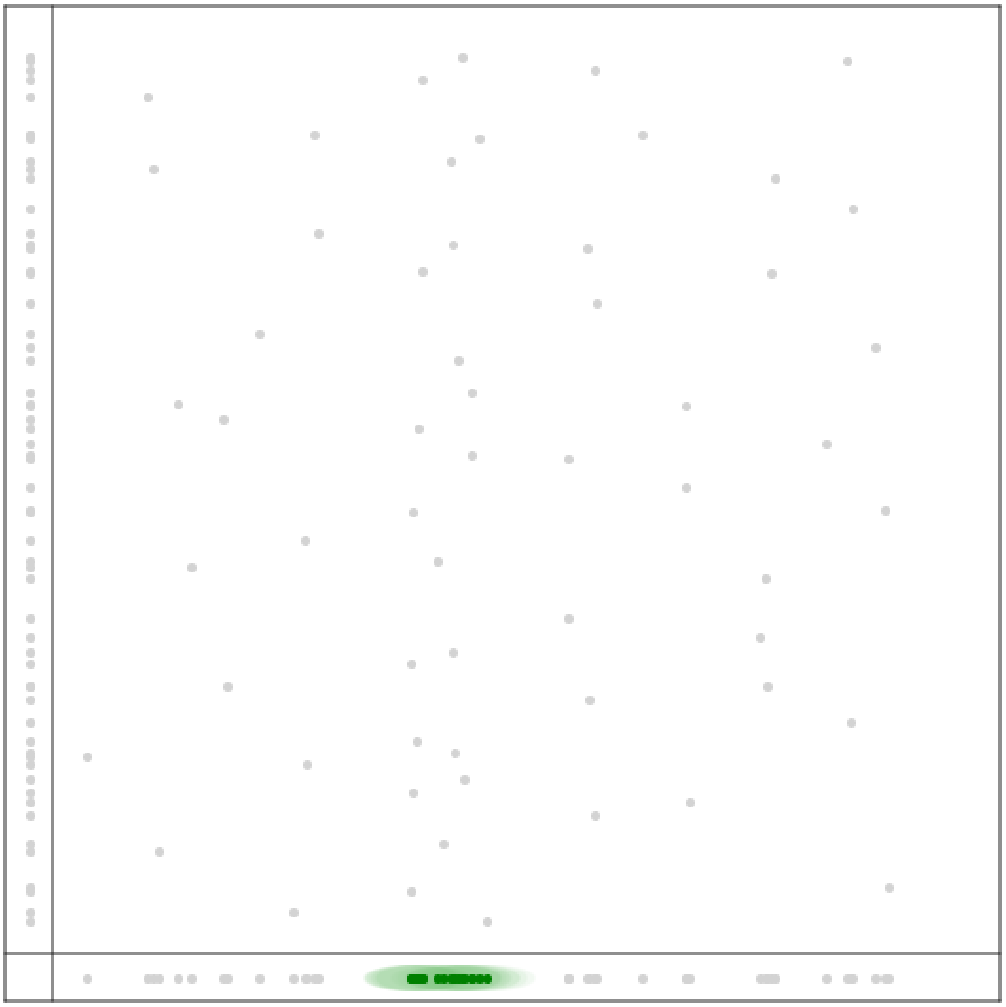

Figure 3: Cluster hidden in a subspace

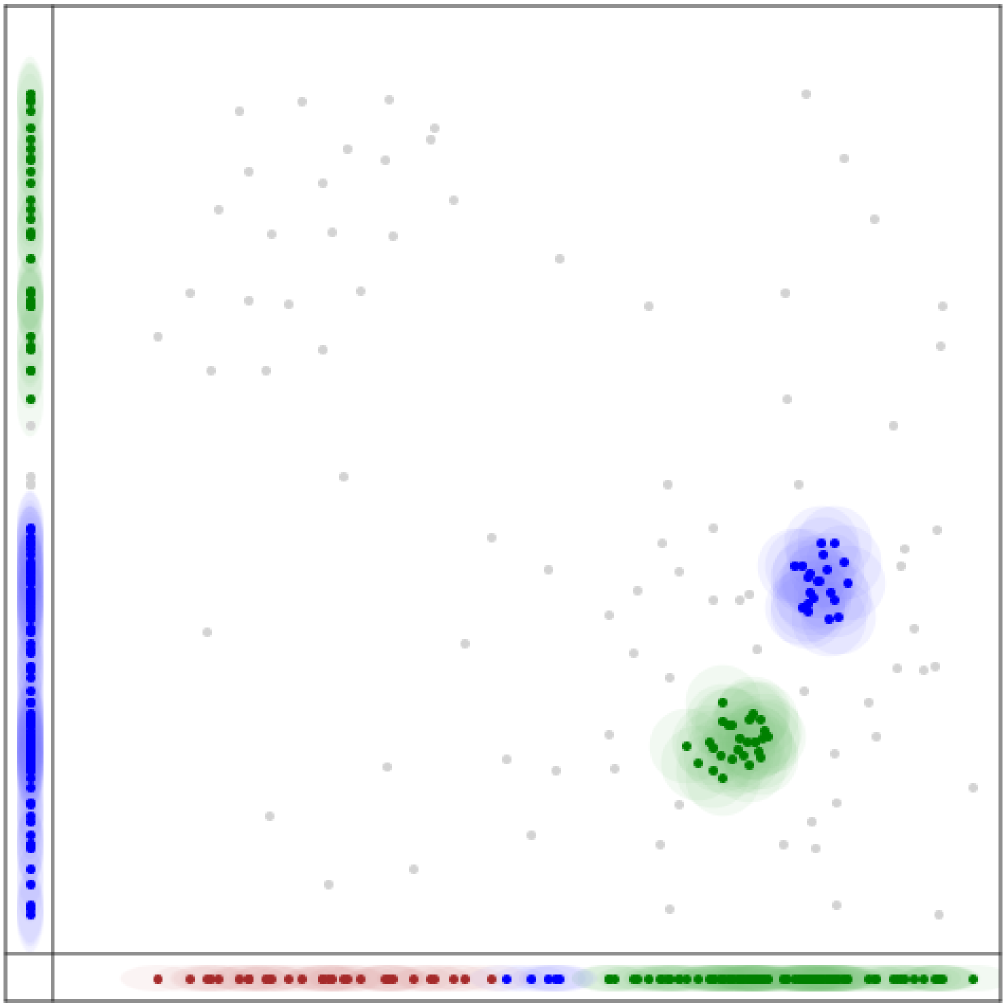

Figure 4: SUBCLU failure due to irregular density (a)

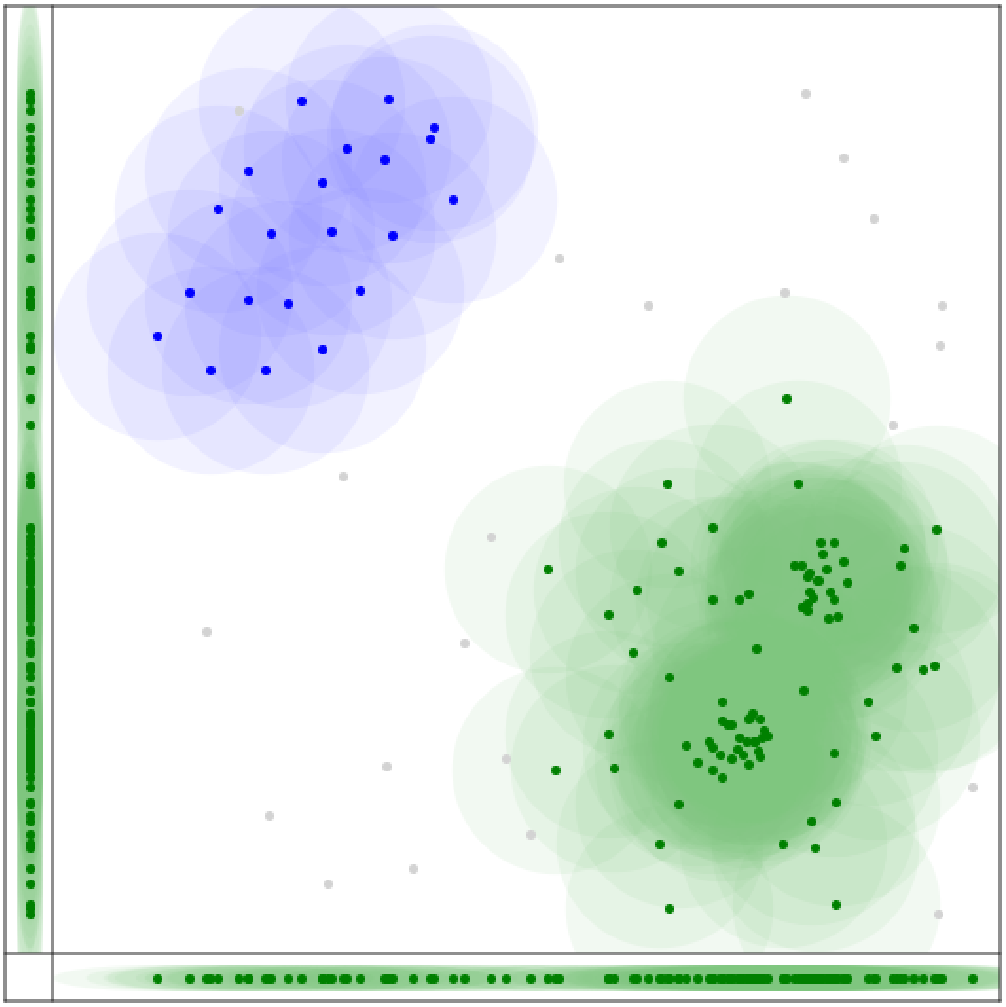

Figure 5: SUBCLU failure due to irregular density (b)

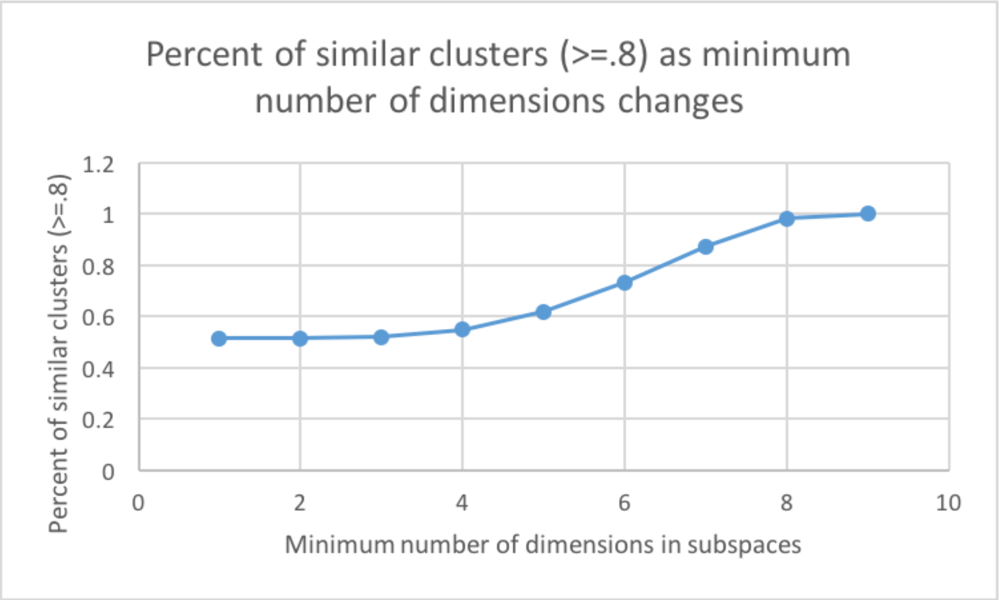

Figure 6: Raising the minimum subspace size increases similarity

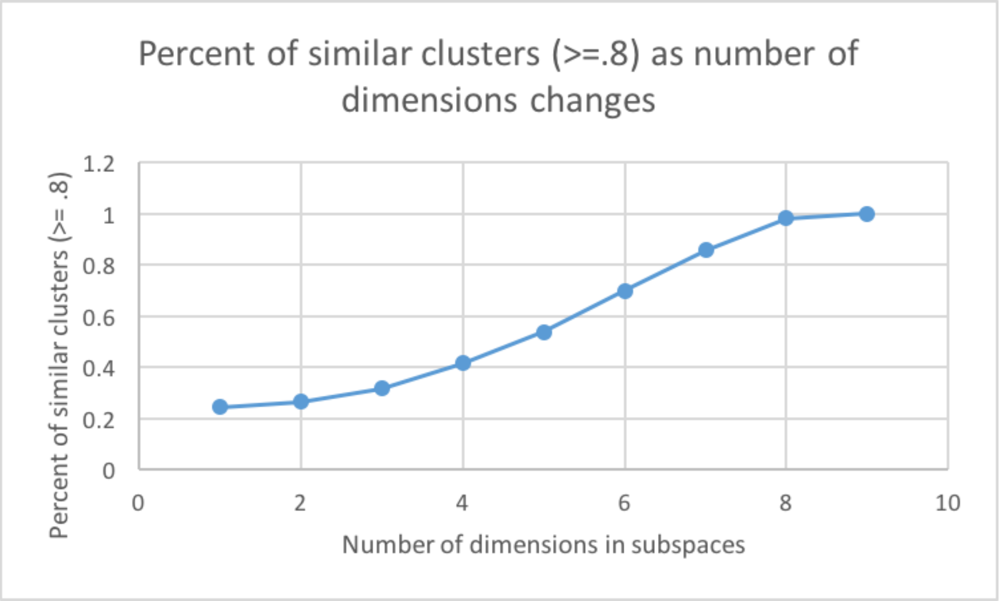

Figure 7: Higher subspace sizes are more accurate