UNSUPERVISED AND REINFORCEMENT LEARNING

MASTER IN ARTIFICIAL INTELLIGENCE

Universitat Politècnica de Catalunya

IMPLEMENTATION AND EVALUATION OF THE SHARED NEAREST NEIGHBOR CLUSTERING ALGORITHM

Albert Espín

April 28th, 2019

Table Of Contents

1. Introduction

Clustering is one of the most common tasks in unsupervised machine learning, consisting in dividing the data points in groups (clusters), in such way that the elements of each group share some features that make them similar (according to a similarity metric, e.g. euclidean or cosine), and different from instances in other clusters. A wide set of algorithms can perform the clustering task, each of them leading to different results, due to their differing criteria on how to associate or link the points.

The first clustering algorithm that became widely used in both research and industrial applications is K-Means (Lloyd, 1957). This algorithm finds a user-specified number of clusters, *k*, each of them represented by a fictional data point called centroid (or prototype), defined as the mean point of all the real data points of the cluster. The algorithm initializes centroids randomly or with a specific algorithm (such as K-Means++, that promotes initial centroids to be as far as possible from one another). After that, two consecutive update steps are repeated: each data point is assigned to its closest cluster (the one minimizing a distance metric), and afterwards all centroids are recalculated as the new mean point of their cluster. The algorithm stops after a maximum number of iterations *t*, or performs an early stop in case of convergence (that can be defined in different ways, such as no point changing cluster from one iteration to another, or the distance between the previous and new centroids being below a threshold value). Therefore, the time complexity of the algorithm is $O(n\,k\,t)$, if *n* is the number of data points. The space complexity is $O(n^2)$, which makes it advisable to use batch-based approaches for large samples of data, so that instances are progressively loaded into memory when needed, and not all at once. K-Means obtains a set of clusters representing a local minima of the optimal combination, which is not normally attempted to find, since it would be much more computationally expensive (it is an NP-Hard optimization problem). To improve the quality of the results, different strategies exist, such as the aforementioned K-Means++ initialization, or the repetition of the algorithm multiple times and selection of the results of the one considered more suitable according to a certain cluster evaluation metric (e.g. the one with the highest adjusted mutual information score, if the true class labels are known).

K-Means is very fast, but it has some limitations. Firstly, it will always find *k* clusters in the data, no matter how it is structured, and this fixed number of clusters must be specified by the user, i.e. known or estimated in some way. Secondly, the centroid-based definition of K-Means makes it biased towards finding globular (or hyper-spherical) data clusters. For this reason, it does not produce acceptable clusters when the shape of the data is very different from globular, such as representations of concentric rings. Many different alternatives have been designed to tackle these problems. Spectral clustering (Cheeger, 1969) is one of them, finding clusters not in the original data, but in a normalized Laplacian projection of it. It is far more resource-consuming than K-Means, with a time complexity of $O(n^3)$, since it has to compute the similarity matrix and find the eigenvalues, to reduce the dimensionality of the data and perform the actual clustering in a lower-dimension space. The advantage of spectral clustering is, as mentioned, the capacity to find non-globular clusters, even though it fails on some complex, noisy scenarios.

Both K-Means and spectral clustering algorithms require to specify the number of clusters. An innovative method that does not have this requirement is the Density-Based Spatial Clustering of Applications with Noise, or DBSCAN (Ester et al., 1996). In this algorithm, points are clustered together when they are considered to be part of the same mass of points (density area). Points are classified in three categories: core points, when at least a certain user-specified number of points are within a user-specified radius of reachability for the point; normal points, when they are in reach of at least one core point; and outliers (if not reachable). Clusters are formed starting at each core point and adding to the cluster all the reachable points, progressively merging the clusters of core points that can be reached through points in both clusters. DBSCAN has a worst-case complexity of $O(n^2)$, but some implementations achieve an average of $O(n \log n)$ using efficient indexing of points. Despite not having to specify the number of clusters, it forces the user to specify two other parameters, the radius and the minimum number of reachable points to have a core point. The notion of density that powers this algorithm allows it to find clusters of arbitrary shapes, but only if they have very similar density.

The next Chapters of this document explain a more recent clustering algorithm, Shared Nearest Neighbor, that extends the functionality of DBSCAN to achieve robustness to inter-cluster density. While Chapter 2 describes the theoretical background and practical steps used to implement the algorithm, Chapter 3 compares this new implementation with reputed ones of the other algorithms, to empirically determine which scenarios are the best suited for each technique.

2. The Shared Nearest Neighbor clustering algorithm

2.1. Theoretical formulation

The Shared Nearest Neighbor clustering algorithm (Ertöz et al., 2003), also known as SNN, is an extension of DBSCAN that aims to overcome its limitation of not being able to correctly create clusters of different densities. This problem is grounded in the distance-based definition of density used by DBSCAN. In other words, DBSCAN considers that points are more important (eventually becoming core points) the higher the number of points they have nearby, so that the more dense regions will tend to be clustered, while sparse areas are considered as noise, outliers. This view ignores that some groups of sparse data actually form a conceptual unity, and therefore should be clustered.

To achieve independence to density, SNN uses the idea of the shared neighbors (Jarvis & Patrick, 1973), by considering that the distance between two points is not directly a spatial measurement, but related to how many of the top spatially-closest points, or neighbors, they share. With this notion, the more nearest neighbors they share, the more similar two points are; the less, the more distant. In sparse regions, the top nearest neighbors of a point are spatially far, while in dense regions the neighbors are close to each other. Effectively, this is a way to normalize distance by density, and allows to detect clusters of different densities, even when they are quite close to each other.

The original SNN proposal (Ertöz et al., 2003) lists a series of steps to perform the algorithm, taking into account that the last 4 correspond to the DBSCAN algorithm:

1. Construct the similarity graph for the data, where data points are the nodes and all pairs are connected, with the edge weight being their spatial similarity (the complementary of the spatial distance).
2. In the similarity graph, remove all connections but those corresponding to the *k* nearest neighbors of each data point, i.e. the most similar ones.
3. Construct the shared nearest neighbor graph using the pruned similarity graph, by associating each data point to each other in such way that the edge weight is a number proportional to the quantity of *k* nearest neighbors

that they share. This can be computed by checking in the pruned similarity graph which nodes are directly connected (in 1-step paths) to the points.

4. Calculate the SNN density of each data point, using a reachability radius measured in terms of SNN similarity among points.
5. Find the core points, those with a number of reachable points equal or higher a minimum amount, that can be expressed as a fraction of the number of neighbors *k*, which defines the maximum number of neighbors that two points can have in common.
6. Assign each core point to a different cluster, and merge clusters of mutually-reachable core points.
7. Identify noise points, those that are not reachable from any core point, so that they are considered to be outside of all clusters (outliers).
8. Assign all points that are neither core points or noise points to the cluster of their closest core point, considering shared-neighbor similarity.

It can be observed that the computational complexity of the SNN algorithm is lower-bounded by the $O(n^2)$ cost of computing the spatial similarity between all pairs of points. The actual DBSCAN-alike proceeding is, in the best case, $O(n \log n)$, and should be also taken into account. Therefore, this algorithm is clearly slower than DBSCAN.

2.2. Implementation

The algorithm has been implemented in Python 3, following the conventions of the *scikit-learn* library (Pedregosa et al., 2011). That is, a class for a handler of the SNN algorithm has been defined, inheriting from *BaseEstimator* and *ClusterMixin*, like other clustering handlers such as that of DBSCAN, to implement a *fit* method to divide the data in clusters.

The actual logic of the SNN is located in a separate function called by the handler, *snn*, just like done for DBSCAN in *scikit-learn*. The function expects a feature matrix *X*, and two additional parameters essential for SNN to work: the number of top *k* nearest neighbors to take into account in the shared neighbor similarity calculus, and the proportion of *k* neighbors that should be reachable for a point for it to be considered a core point.

The implementation of the *snn* function relies heavily on *scikit-learn* functions and *numpy* arrays, to guarantee efficiency. In particular, the first two steps of the algorithm, consisting in obtaining the spatial similarity graph of the data limited to the *k* nearest neighbors is obtained with the *scikit-learn* function *kneighbors_graph*. Then, the third step of the algorithm is computed by storing in a *numpy* matrix the shared-nearest-neighbor distances between each pair of data points. The distance is the complementary of the similarity, computed as the number of neighbors in common (calculated as the intersection of the two sets of neighbors). For the remaining steps, 4 to 8, the DBSCAN *scikit-learn* is used, passing it the already computed distance matrix (specifying that it does not need to be recalculated, setting parameter *metric='precomputed'*), and using the *fit* method to assign clusters to each data point. In other words, performing SNN is the same as performing DBSCAN on a set of data points that are not the original feature-based data, but a derived set of shared-neighbor distances to the rest of points.

3. Experimental Analysis

3.1. Experimental Methodology

The basic objective of the experimental analysis is to empirically determine the suitability of each of the mentioned clustering algorithms (K-Means, Spectral, DBSCAN and SNN) by comparing their performance in a wide range of data sets of different characteristics. To eliminate biases based on different magnitudes of the data features, the data is scaled to have mean 0 and standard deviation 1 as a pre-processing step. The SNN implementation is measured against reputed implementations of the rest of algorithms, namely those provided by *scikit-learn*. Other interesting, yet less well-known algorithms such as alternative DBSCAN extensions have been omitted due to their lack of highly reputed Python implementations available: these include Cure (Guha et al., 1998) and Chameleon (Karypis et al., 1999); which can not rival the density-robustness of SNN, according to the original proposal experiments (Ertöz et al., 2003). Therefore, this analysis is focused in comparing SNN with more well-established algorithms; for a more fair comparison, different combinations of the parameters of DBSCAN are tested, and the best ones are used to show the final results. The real number of clusters is specified for K-Means and Spectral clustering.

A plot showing the clusters assigned by each method is shown, which should give an idea of how well each algorithm has clustered the data. Nevertheless, three additional measures are computed for more formal evaluation of the results. Namely, they are:

- Adjusted mutual information score (MI in plots), which checks the intersection of points in every pair of clusters between the real and predicted sets. It estimates the degree of dependence between the real and the predicted clusters, adjusted to neutralize the possibility of the clusters being similar just by change. It ranges from 0 (no similarity) to 1 (equal clusters).
- Adjusted Rand index (RS in plots), which measures the similarity of clusters based on the pairs assigned to same or different clusters in the real and predicted assignments. As in the previous case, it is robust to similar-by-chance assumptions. It ranges from 0 (no similarity) to 1 (equal clusters).
- Calinski-Harabaz score (CHS in plots): it is a ratio between the within-cluster and inter-cluster variance. A low value indicates that the variance between the elements of a cluster is smaller than that of elements of different clusters, which is the preferred scenario.

The experiments have been performed in data sets of different sizes and properties:

- A set of simple-haped synthetic data sets shown in a *scikit-learn* example (also used as base code for scatter-plotting the results of the competing algorithms). These data sets include concentric rings, half circles, blobs (of globular and non-globular shape) and square shapes.
- Two extensions of the previous synthetic data sets combining multiple clusters of a wide range of densities, with some challenging cases of very near clusters.
- Three real data sets. One of them is a small one: iris, with 3 classes, 4 features and 150 instances. The other two can be considered high-dimensional challenging data sets, normally used in supervised learning to classify instances, but it was wanted to push clustering to its limits to see if it is able to find the true classes of the data with no labels. Namely, one of them is breast cancer, a binary classification data set with 30 features for only 569 instances, a clear case of curse of dimensionality, in which the data points are sparse in a high-dimensional space so that spatial distances tend

to become similar and lose meaningfulness; the other data set is even more challenging: it is a face classification data set called "Labeled Faces in the Wild" (yet labels are not used to cluster), with 5828 pixel features for 13233 instances of 5749, making much more obvious the dimensionality problems just described.

3.2. Results and Discussion

The inability of K-Means to detect non-globular clusters becomes obvious when observing the cluster assignments depicted in Figure 1. Namely, it is not able to detect concentric rings, nor the correct boundaries of half circles, nor those of elongated blobs. K-Means only performs as expected for the globular blobs, due to its centroid-based definition, assigning points to the closest centroid, that attracts them with a hyper-spherical radius. This globular bias is also obvious, in a negative way, in the different-variance blobs, that are close to each other, so that points in the border between real clusters are captured by the globular radius of the nearest centroid, particularly one of a smaller cluster that just happens to be closer to the border points of another cluster that is much larger, so that the centroid is further. For the square structure the number of clusters is specified to be 3 despite of being just 1 in reality (it would be trivial to cluster), to make it obvious how, independently of the data, K-Means will always find the number of clusters that it is given. This circumstance also happens when using Spectral clustering in the square data. However, the projection of Spectral clustering to a eigenvalue space allows it to correctly detect clusters in the concentric rings data, but not in its highly noisy version, due to the difficulty to differentiate between the groups of points even in the normalized Laplacian space used by this algorithm. It also succeeds in the half circles and performs much better in the elongated blobs cases, again due to the eigenvalue representation of the data, a mapping that is less affected by globular bias.

DBSCAN outperforms the previous algorithms in the noisy circles, due to its focus on following shapes by its notion of density, but it fails in regions where the two clusters are very close, which makes it perform worse than Spectral clustering in the different-variance blobs case. It succeeds to detect that the square problem contains a single cluster. Unlike the previous algorithms, DBSCAN assigns no cluster to the points it considers to be noisy, outliers (shown in black). This feature is inherited by SNN, which is the only of the four algorithms that completely succeeds in the detection of clusters in the noisy concentric rings problem: thanks to its shared-nearest-neighbor criteria, it finds the gap between the two concentric circles, even in regions where it is very narrow (that make DBSCAN fail), seeing them as different clusters. The same phenomenon can be observed in the different-variance problem. Both the adjusted mutual information and the adjusted Rand index of the SNN algorithm is superior to that of the rest of algorithms, for all these simple data sets. The Calinski-Harabaz measure, that is better the lower it is, is smaller for SNN in all the problems but for the difference-variance and elongated-blobs cases, where DBSCAN is somewhat lower, but this should be seen as caused due to the high proximity of the clusters, that SNN still detects more accurately.

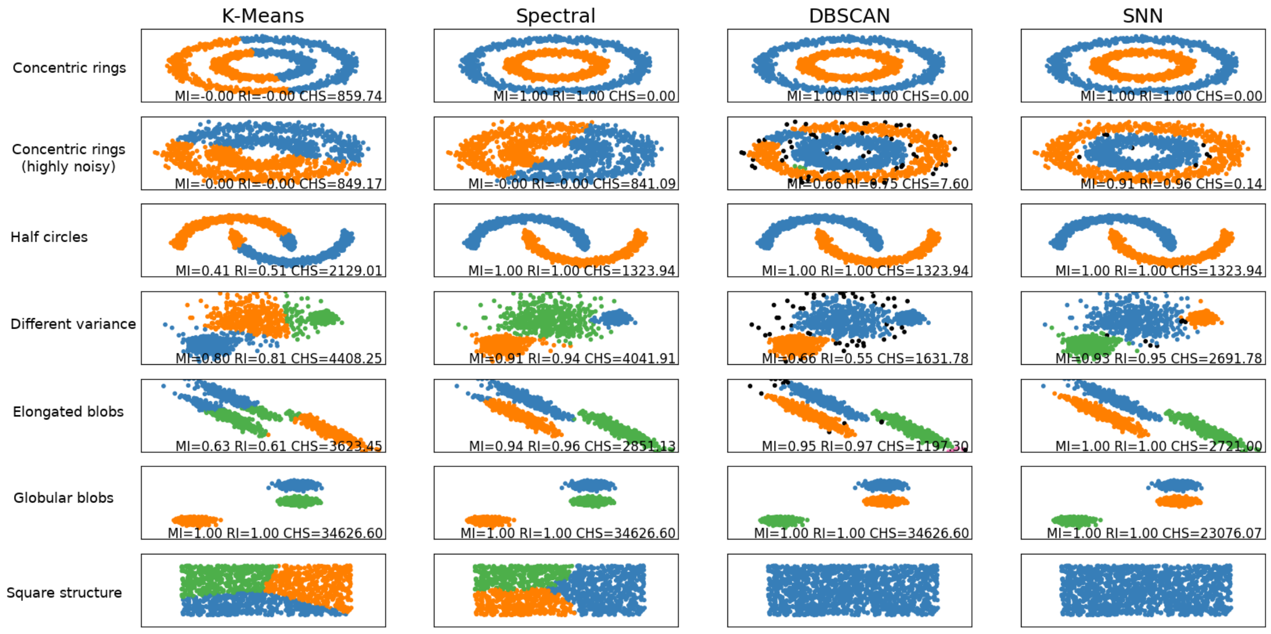

Figure 1: Comparison of different clustering algorithms in simple synthetic data sets (based on a *scikit-learn* example).

Figure 2 shows two more complex synthetic data sets, especially designed for these experiments, where many clusters are present, featuring sets of points of very different densities. Some of the real clusters contain a large number of points in a small space, while other sparse clusters exist through a large part of the space, with a high degree of separation between each of their points: often even larger than the distance between the pair of points furthest from each other in one of the small clusters. To make the problem even more challenging some clusters are placed very close to each other, almost overlapping. It can be seen how K-Means and Spectral clustering are biased to separate high-density points, that weight more in the centroid calculations, attracting other surrounding points of more sparse clusters that simply are closer to some high-density cluster than to any other.

The results of DBSCAN are also quite poor: despite of the fact that it is able to detect some clusters with complex shapes, it finds many wrong micro-clusters in regions where a few surrounding points happen to have a density somewhat similar among them and different to the rest of their neighbors. It also considers that many points of the more sparse clusters are outliers, since the sparseness make those points be less reachable by core ones (in highly dense areas). SNN, however, proves to be highly robust to density differences, by detecting clusters in a more reliable way, with higher mutual information and rand index results. It is especially significant how sparse clusters are identified to surround most of the dense clusters in the second data set, where also a very difficult case of touching clusters is correctly separated as two different groups, examining the globular shape that they form, leaving a small opening in the borders that alters the shared neighbor similarity among points of each group, leading to the successful discrimination. Here, the sparsity of the clusters, some of which surround each other, makes Calinski-Harabaz not suitable for the evaluation of clustering quality.

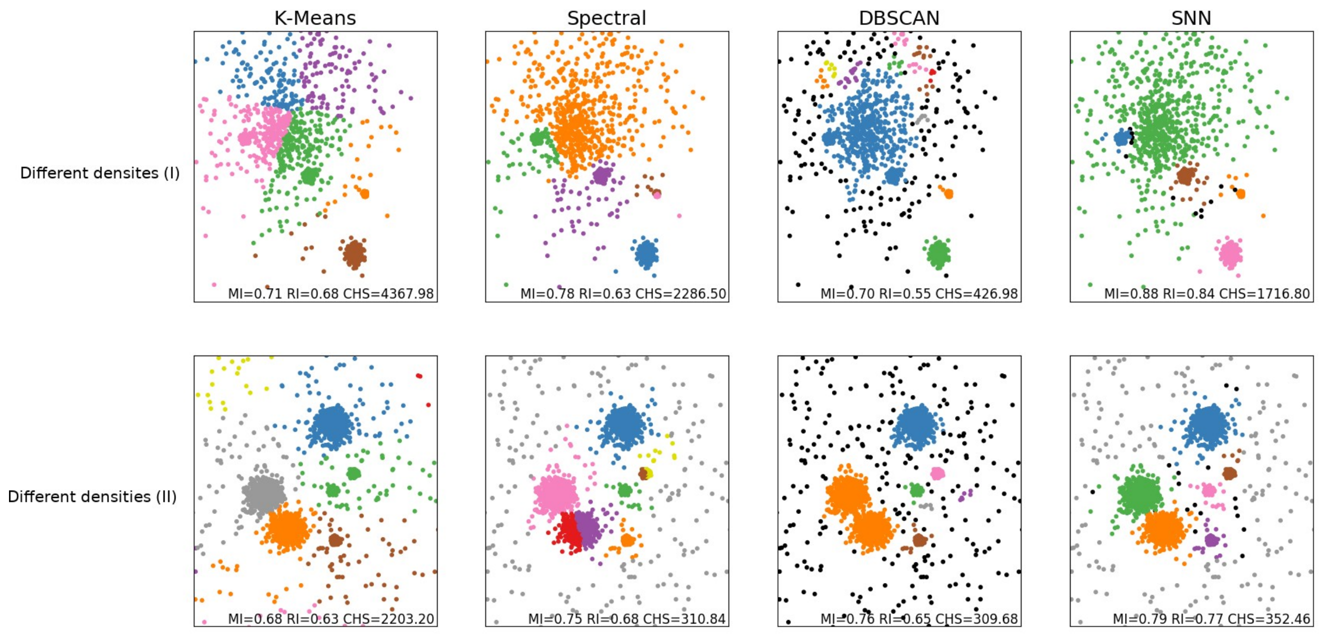

Figure 2: Comparison of clustering algorithms in challenging synthetic data sets.

Figure 3 shows the application of the same clustering algorithms to real data sets. All of them have more than 2 dimensions, so the two-dimensional plot shows only two of them, selected as those showing a reasonably meaningful visualization, when possible. The first data set is Iris, where K-Means and Spectral clustering have the advantage of being given the real number of clusters, while DBSCAN and SNN find two regions so well-connected, and sharing a similar density, that they detect them as a single cluster, finding a total of two clusters instead of 3. Nevertheless, SNN still scores the best mutual information and rand index, finding divisions that make more sense than those of K-Means for instance, which are based on its globular bias. Iris is typically used as a classification data set, and without knowing the classes and training a model the clustering algorithms are unable to draw a hyper-plane separating the two classes that have points with spatial continuity. Taking into account that no label information is used by SNN, its adjusted mutual information of 0.73 and rand index of 0.57 shows that the classification is not very imprecise. DBSCAN, however, obtains a better value of Calinski-Harabaz score, by discarding some dubious points, separated from the main density areas. Nevertheless, such decision of DBSCAN is not a suitable for this data set, which contains not outliers (DBSCAN, however, ignores such information).

The breast cancer data is much more complex for a clustering task. Nevertheless, it is remarkable to see how SNN, without knowing the true number of clusters in this high-dimensional data set, is able to conclude that there are 2 unique classes, and achieve values of mutual information and rand index not very lower than those obtained by K-Means and Spectral clustering, that know the number of clusters from the very beginning. It should be considered, however, what would happen if that number was not known: these two algorithms would require many iterations to test different values of k to find the best one, while a single SNN execution obtains results of quite similar quality to the best iteration of the other two. All these three algorithms are slightly better in the classification task than a random choice (the metric results are clearly greater than 0.5, for a problem with two equally-distributed classes), which is considerably good knowing that the label information is not used. The curse of dimensionality leads to having very sparse elements, making the notion of distance less reliable and many different densities are present in the hyper-space, a circumstance that makes DBSCAN collapse if using a neighborhood radius of the magnitude of the previous examples, concluding that all points are noise, without forming reasonable density areas that the algorithm can detect; if the radius is enlarged (as done in Figure 3), it can eventually detect that there are two clusters, but still many examples are considered noise due to curse of dimensionality; with even higher values for the radius, just one cluster is detected. At any case, the mutual information and Rand index of DBSCAN is very poor compared to the rest of algorithms. Thanks to the shared-nearest-neighbor definition of distance, SNN is able to at least detect some density areas, even if they are not especially accurate, which is normal due to the high dimensionality.

The "Faces in the Wild" data set brings the aforementioned high dimension complications to an extreme case. Obviously, no clustering algorithm by itself is capable of performing face classification with image-based features (pixels) with thousands of possible faces (with few images present for each of them) and absolutely no labels. Again, the curse of dimensionality makes DBSCAN consider that everything is noise with a reasonable neighbor radius. If the magnitude of radius is increased orders of magnitude (to be about 100 times higher than the used for the toy data sets), most points are still detected as noise, but some clusters are identified (as depicted in Figure 3); just a slightly higher value for the radius makes all points be considered to be part of the same cluster. In any case, the best DBSCAN results are a mutual information of 0.02 and a Rand index value of 0, which is extremely poor, and the worst among all the tested algorithms. DBSCAN's Calinski-Harabaz score is much lower than that of the rest of techniques, but only because most of the points are considered noise, and therefore not taken into account for the cluster variance calculations. The best mutual information result is obtained by both K-Means and SNN (0.07, which is very low, but not so bad considering the complexity of the problem), with SNN obtaining the best rand index, even if still very poor (0.03, three times higher that that of Spectral clustering, 0.01, while K-Means obtains 0). This is, nevertheless, a considerable achievement by SNN, since K-Means and Spectral clustering are given the number of clusters beforehand (which is not trivial to determine in an unsupervised way when having tens of thousands of instances belonging to thousands of classes), which makes it understandable and not a high achievement the fact that K-Means and Spectral clustering have lower Calinski-Harabaz score than SNN.

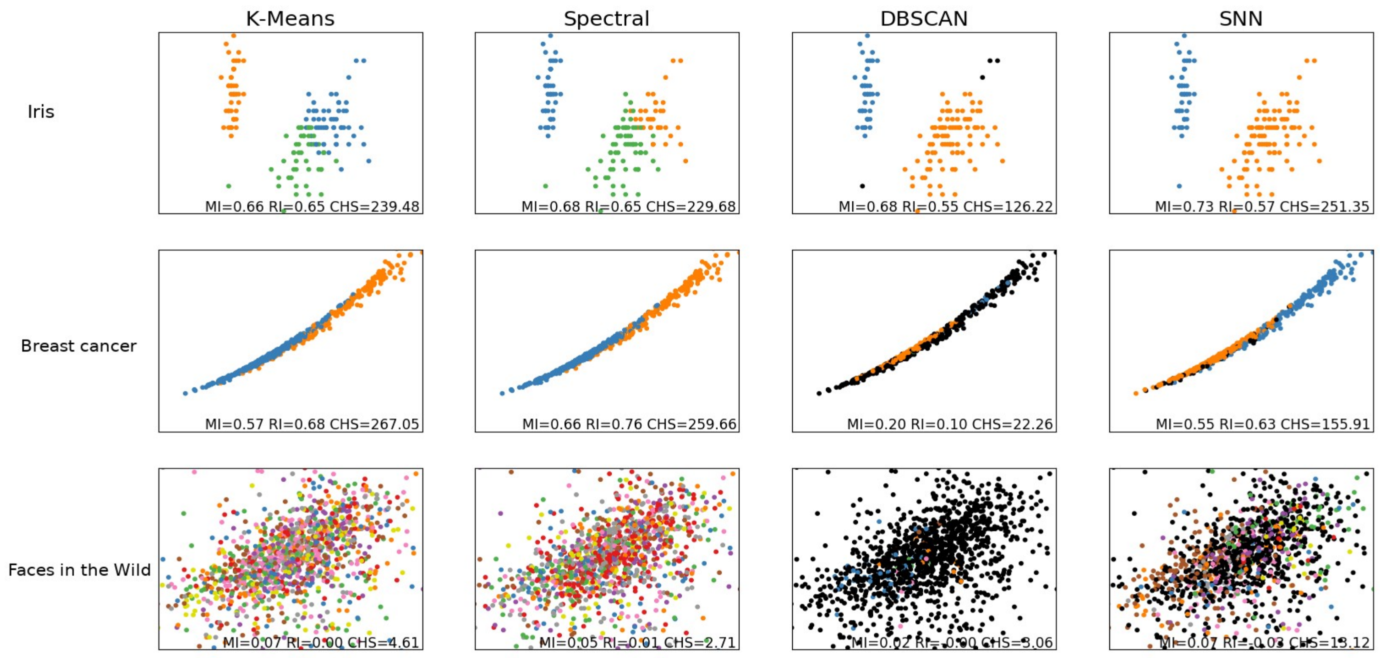

Figure 3: Comparison of clustering algorithms in real data sets.

4. Conclusions and Future work

The experiments performed in this work have empirically shown that SNN is equally good to traditional clustering algorithms such as K-Means and Spectral clustering in approximately globular data. In globular scenarios, K-Means should be preferred due to its lower time complexity (in a batch-based implementation for large data, to avoid memory requirements of quadratic complexity). Both SNN and DBSCAN clearly outperform Spectral clustering and especially K-Means in problems with arbitrary, non-globular shapes. However, there are two scenarios where SNN clearly outperforms DBSCAN: when two clusters have boundaries very close to each other and when data clusters have significantly different densities. In such cases, the shared-nearest-neighbor definition of distance makes SNN robust to different densities, and able to detect the real clusters more accurately. Therefore, SNN should be preferred for complex problems of different densities, while if the density is approximately constant and the clusters are known to be well-separated DBSCAN should be used, since it is faster: SNN has a time overload of calculating the k nearest neighbors and constructing a distance matrix based on the proportion of neighbors each pair of points share, which grows linearly with k.

Future work can be oriented in different research directions. One interesting option would be to consolidate (or try to refute under certain conditions) the results obtained in the current experimental setting. To do so, more clustering algorithms may be tested in the tested data sets, as well as in other ones. The new clustering algorithms may include Cure and Chameleon, that would first need to be implemented following the *scikit-learn* conventions, relying on *numpy* for matricial representations of data (as SNN in this work), to be comparable in fair, similar conditions. The new data sets may include new real or synthetic data sets with shapes that could challenge the properties of SNN and the rest of algorithms in innovative ways. It would be interesting to see, for instance, how they react to highly noisy data, and see whether SNN correctly distinguishes between noise or what may be considered by the algorithm to be a sparse cluster. Another possible orientation for future work would be precisely to try to refine SNN even further, such as allowing additional tuning parameters. One customization option may include a threshold for cluster density, possibly expressed not in absolute terms but as a relative measure. The threshold would be used to consider that extremely sparse clusters (those with a intra-cluster variance much higher to that of other clusters) should be discarded and considered as noise, to compensate SNN's higher tolerance to sparsity. In some problems, sparsity may be highly related to outliers, even if they are numerous enough to be considered as a group on its own: in some problems such groups may be interesting for study purposes, while in others they should be discarded outright.

5. References

[Cheeger, 1969] Cheeger, Jeff (1969). A lower bound for the smallest eigenvalue of the Laplacian. *Proceedings of the Princeton Conference in Honor of Professor S. Bochner*.

[Ertöz et al., 2003] Ertöz, L., Steinbach, M., & Kumar, V. (2003, May). Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM international conference on data mining* (pp. 47-58). Society for Industrial and Applied Mathematics.

[Ester et al., 1996] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press. pp. 226–231.

[Guha et al., 1998] Guha, S.; Rastogi, R.; Shim, K.; (1998). Cure: An efficient clustering algorithm for large databases. In L. M. Proceedings of SIGMOD '98, The ACM SIGMOD International Conference on Management of Data. pp. 73–84. ACM Press.

[Jarvis & Patrick, 1973] Jarvis, R. A.; Patrick E. A. (1973). Clustering using a similarity measure based on shared nearest neighbors. *IEEE Transactions on Computers*, C-22(11).

[Karypis et al., 1999] Karypis G.; Han E.-H.; Kumar V. (1999). Chameleon: A hierarchical clustering algorithm using dynamic modeling. IEEE Computer, 32(8):68–75.

[Lloyd, 1957] Lloyd., S. P. (1957, published 1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory.* 28 (2): 129–137.

[Pedregosa et al., 2011] Scikit-learn: Machine Learning in Python, Pedregosa F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. (2011), *JMLR 12*, pp. 2825-2830.