Unsupervised learning coursework: Generation of Random Clusters with Specified Degree of Separation

Dmitry Gnatyshak

04-06-18

1 Introduction

Synthetic dataset generation is a rather important problem in data analysis during the development of new algorithms. Though it is crucial to test such algorithms on well-studied real-world datasets, it may be better to test some properties on specifically generated synthetic data. Furthermore, though there is a lot of available datasets for research, new algorithms may require some very specific structures of data for testing that may not be readily available. Finally, generating synthetic data of predefined structure with set properties is an interesting problem itself, helping to understand the behaviour of data.

This paper introduces an algorithm for generating clustered dataset with an additional control over separation between clusters [1].

2 Algorithm description

The proposed algorithm is a modification of the algorithm in [2]. The main purpose of this algorithm is to generate clusters with specified separation degree rates between nearest pairs of clusters [3][4]. In addition to this main feature the algorithm also allows to have noisy variables and noise examples. Also it performs additional transformations of the data to ensure that it not very easy to distinguish clusters by the means of mere pairwise visual study.

The steps of the algorithm are provided below.

Step 1. First of all we need to specify the parameters. It has 10 main parameters:

Number of features, p_1: number of normal features, each of which represents a random variable from a mixture distribution of normal distributions, one per cluster.

Number of clusters, K: number of clusters to be generated.

Number of noisy features, p_2: number of noisy variables that do not depend on cluster and are independent of non-noisy variables. Drawn from normal distribution.

Degree of separation, J_0: degree of separation between each cluster and its closest neighbor as described below.

Separation quantile, $q_{\frac{\alpha}{2}}$: quantile used for the computation of separation index. In case of normally distributed non-noisy variables are upper $z_{\frac{\alpha}{2}}$ quantiles, 1.96 in case of $\alpha = 0.5$.

Number or ratio of outliers : either the total number of outlier examples or their ration with respect to the number of normal examples. Generation mechanism is described below.

Minimal eigenvalues value, λ_{min}: minimal value for the range from which eigenvalues for feature covariance matrices are uniformly sampled.

Maximal to minimal eigenvalue ratio, r_λ: ratio of the highest value of the range for eigenvalues to the lowest one. Together they define that range.

Minimal size of clusters, n_L: minimal possible size of clusters.

Maximal size of clusters, n_U: maximal possible size of clusters.

Step 2. Generate cluster centers and covariance matrices with respect to non-noisy variables so that the separation condition is satisfied. This is the main and the most computationally hard part of the algorithm.

First of all generate K random covariance matrices, on per cluster. This can be done firstly by sampling eigenvalues from uniform distribution $\mathcal{U}(\lambda_{min}, \lambda_{min} \cdot r_\lambda)$, sorting them and forming the diagonal matrix Λ. Then we need to generate random orthonormal matrix Q. Then the covariance matrix can be obtained using the eigenvalue decomposition formula: $C = Q\Lambda Q^T$.

Then we find a p_1-dimensional simplex (with $p_1 + 1$ vertices) with edges of size 2. If the number of clusters is less than $p_1 + 1$, we just take first K vertices as cluster centers. If it is greater than $p_1 + 1$, we take also shifted vertices.

After that we need to ensure the separation index constraint. Separation is defined by the ration of minimal to maximal differences between fixed percentiles of 2 clusters. The direction of projection for this is chosen to maximize the index and is found by solving an optimization problem. Suppose the optimal direction is a, the lower and upper percentiles of the clusters i and j are, respectively, L_i, U_i, L_j, U_j. Then the separation index is equal to:

$$J(a) = \frac{L_j - U_i}{U_j - L_i}$$

In our case of normal distributions the formula changes to:

$$J(a) = \frac{a^T(\mu_2 - \mu_1) - q_{\frac{\alpha}{2}}\left(\sqrt{a^T\Sigma_1 a} + \sqrt{a^T\Sigma_2 a}\right)}{a^T(\mu_2 - \mu_1) + q_{\frac{\alpha}{2}}\left(\sqrt{a^T\Sigma_1 a} + \sqrt{a^T\Sigma_2 a}\right)}$$

We find the separation values for all pairs of variable and scale the length of the edge of the simplex by a scalar until the minimal non-diagonal separation is (approximately) equal to the desired J_0. This is a simple optimization problem.

Finally, for each variable we compute the minimal separation value (and thus get the nearest neighbor clusters). If the maximal of these minimal separation is not greater than J_0, we got the right scaling. If not, we scale the corresponding covariance matrix by another scalar until the corresponding degree of separation is equal to J_0, than repeat this step. This is another optimization problem.

Step 3 We generate the sizes of each cluster uniformly from discrete uniform distribution $\mathcal{U}(n_L, n_U)$. This way we also get the total number of non-outlier examples. Thus, we can generate the membership vector.

Step 4 We generate the mean vector μ_0 and the covariance matrix $\Sigma_)$ common for all the noisy variables. If we consider each variable as a variable from a mixture distribution of K normal distributions weighted by the size of the corresponding cluster, we can represent their total mixture mean vector as $\mu^* = \sum_{k=1}^{K} \pi_k \mu_k$. Than the p_2 noisy variable mean components can be sampled from $\mathcal{U}(\mu^*_{min}, \mu^*_{max})$.

The mixture covariance matrix can be represented as

$$\Sigma^* = \sum_{k=1}^{K} \pi_k \Sigma_k + \sum_{k<k'} \pi_k \pi_{k'} (\mu_k - \mu_{k'}) (\mu_k - \mu_{k'})^T$$

The eigenvalues of the noisy variables covariance matrix are sampled from a uniform distribution with range from the lowest to the largest eigenvalues of Σ^*, eigenvectors are generated as before as a random orthonormal matrix.

Step 5 Now we can generate the random vectors X from the corresponding distributions. Firstly we do it for the non-noisy variables. After that we can rotate this datapoints by left-multiplying them by some fixed orthonormal matrix. This will make it harder to visually distinguish clusters by direct pairwise feature examination, for instance, by using scatter plots.

Step 6 Now we can add noisy variables to the vectors. After that we may shuffle the variables to further complicate visual analysis, making the dataset more similar to the real-world datasets.

Step 7 Now we can also compute the population (and sample) separation values (though if we do this while including noisy variables tis may fail, as they may break properties of the carefully generated data).

Step 8 Finally, we can add outliers. First we compute population/sample mean and standard deviation values $\hat{\mu}$ and $\hat{\sigma}$. Then we generate outliers as a vectors which components are drawn from the uniform distribution $\mathcal{U}(\hat{\mu}_j - 4\hat{\sigma}_j, \hat{\mu}_j + 4\hat{\sigma}_j)$

3 Implementation

The implementation is rather straightforward. The main and only function to be used by the user is `make_sepclusters` fashioned after the `make_classification` function in `sklearn` package. It accepts as the arguments all the necessary input parameters and also a few additional. The full definition is: `make_sepclusters(n_features, n_clusters, n_noisy=0, sep_degree=0.2, sep_quantile=1.96, sep_error = 0.001, outliers=0, lambda_min=1, lambda_ratio=10, min_cluster_size=10, max_cluster_size=100)`. Here `sep_error` is a parameter that defines the precision the optimization functions would try to achieve when implicitly tuning the separation value by scaling simplex and covariance matrices by scalars; this also heavily influences computation times. `outliers` can be either a float between 0 and 1 representing the ratio, or an integer, representing the exact number of outliers.

The implementation follows the structure of the algorithm. The most computationally hard part is finding the centers and covariance matrices, but this can be helped by reducing the `sep_error` variable.

First of all, this function invokes the function `cluster_center_allocation`. That function returns the covariance matrices and cluster centers. `generate_covariance` function is used to generate a random covariance matrix. The orthonormal matrices are not generated by the method provided as an example in the paper as the author explicitly stated that any method will do. For that purpose the simplest solution was to use the `scipy`'s built-in generator of orthonormal matrices.

The optimal projection direction for the pairwise separation index is calculated using `scipy`'s `minimize` optimization function. The scalars to scale the edge of the simplex and the covariance matrices are found by, respectively, binary approximations and `minimize` function.

The simplex is generated by the provided straightforward procedure with addition of the edge length parameter for better reusability during the further optimization.

When the centers and covariance matrices are calculated and adjusted, the noisy variables parameters are also generated and the datapoints are generated from these distributions.

Also, before the return of the output, the data is shuffled in both dimensions to make it less clear where the noisy variables, and what are the groupings of the examples.

The final output of `make_sepclusters` has the same format as the `make_classification`

function: the two-dimensional array `X` containing the observations and one-dimensional array `y` containing the membership functions.

4 Other cluster generation algorithms

One the the obvious choices for generating the dtaset consisting of clusters is to use `sklearn`'s `make_classification` function [5]. Though its main purpose is to generate datasets for classification tasks, such datasets may also be used without their labels for clustering problems.

This algorithm uses similar approach to ours to put cluster centers in the space, but lacks the sophistication and explicit precision in control of the clusters separation rate. Unlike the studied algorithms, `sklearn`'s method places the cluster centers on the vertices of a hypercube in a subspace of the feature space, while the separation is controlled indirectly by setting the edge length of this hypercube.

Still, it provides quite a number of options to control the dimensionality of hypercube, data redundancy, and various customizations to utilize, making it a great choice if there is no need for sophisticated separation control or no additional properties requirements.

There is also even simple option in `sklearn` libraby — `make_blob` function, that just randomly generates a number of points within a certain bounding box, and equally distributes the datapoints among them. These datapoints are normally distributed.

Still, this option is the most trivial of the three, lacking control over many properties of the generated clusters, but simple enough to easily generate data for basic tests.

5 Experimental results

Several sets of experiments were conducted to verify that the implementation and the algorithm itself work as expected. Three most interesting parameters were tested: presence of noisy variables, presence of outliers, and degree of separation.

Parameter values were set as follows, with all possible combinations tested:

Number of features, p_1: 4, similar to Iris dataset.

Number of clusters, K: 3, similar to Iris dataset.

Number of noisy features, p_2: either 0 or 2; if the number is set too high, they will quickly overrule real variables.

Degree of separation, J_0: $[0.01, 0.21, 0.342]$; these values were computed by the authors of the papers and aproximately correspond to the cases, where clusters are very "close", "separated", and "well-separated". It is expected to have better results for higher degree of separation.

Separation quantile, $q_{\frac{\alpha}{2}}$**:** 1.96.

Number or ratio of outliers : either 0 or 20.

Minimal eigenvalues value, λ_{min}**:** 1, empirical recommendation by the authors.

Maximal to minimal eigenvalue ratio, r_λ**:** 1, empirical recommendation by the authors.

Minimal size of clusters, n_L**:** 30

Maximal size of clusters, n_U**:** 100

The twelve datasets were built for the following sets of parameters:

1. ($p_2 = 2$, $J_0 = 0.01$, outliers $= 20$)
2. ($p_2 = 0$, $J_0 = 0.01$, outliers $= 20$)
3. ($p_2 = 2$, $J_0 = 0.21$, outliers $= 20$)
4. ($p_2 = 0$, $J_0 = 0.21$, outliers $= 20$)
5. ($p_2 = 2$, $J_0 = 0.342$, outliers $= 20$)
6. ($p_2 = 0$, $J_0 = 0.342$, outliers $= 20$)
7. ($p_2 = 2$, $J_0 = 0.01$, outliers $= 0$)
8. ($p_2 = 0$, $J_0 = 0.01$, outliers $= 0$)
9. ($p_2 = 2$, $J_0 = 0.21$, outliers $= 0$)
10. ($p_2 = 0$, $J_0 = 0.21$, outliers $= 0$)
11. ($p_2 = 2$, $J_0 = 0.342$, outliers $= 0$)
12. ($p_2 = 0$, $J_0 = 0.342$, outliers $= 0$)

These datasets are sorted in one of the possible ordering of descending clustering complexity, so we may expect the increase of the score along with the increase of dataset number.

Four algorithms were tested in these experiments:

1. LOF to detect outliers, tested on the datasets with actual outliers
2. k-Means as the simplest and one of the most common clustering algorithm

3. DBSCAN as it does not need the information about the number of clusters and as it also quite good at outlier detection

4. Simple consensus clustering, as the advanced version of k-Means, able to find non-convex shapes.

5.1 Local outlier factor (LOF)

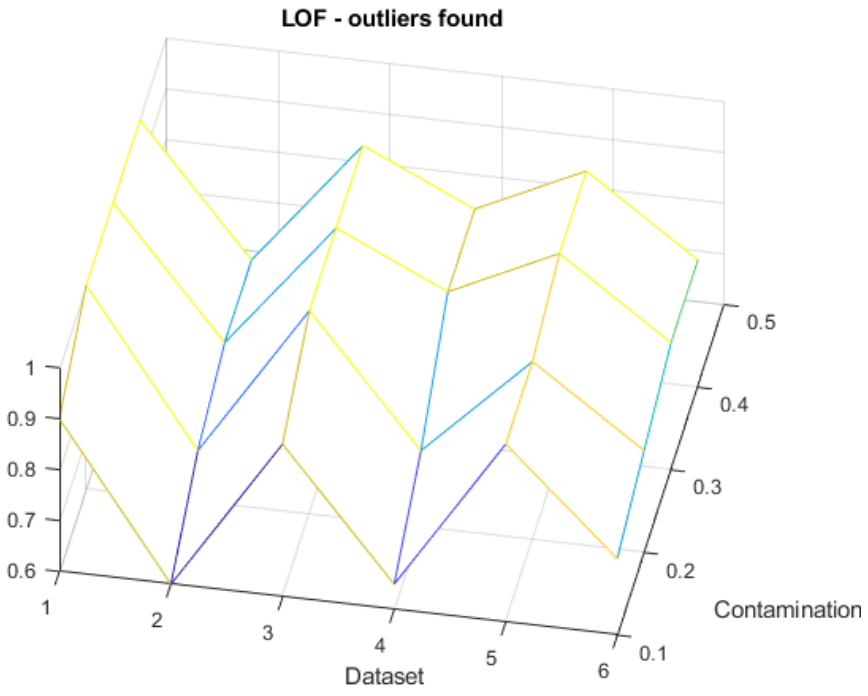

Figure 1: Ratio of found outliers

On Figure 1 and Figure 2 we can see the results of running the outlier search for different datasets for different value of contamination (predicted or expected ratio of outliers in the data). The first thing that catches the eye is that the presence of noisy variables actually helps to detect outliers. The reason is that the these variables are more "stable" and, besides everything else, they are normally distributed; thus when a significant number of features behaves the same way for all examples except the small few, it makes if easier to detect outliers with extreme values of features. The higher false positive rate for the other 3 datasets also confirms this.

Another interesting though obvious observation is that increasing the contamination value above the actual one still increases the number of correctly identified outliers, but at cost of huge false positive rates. Thus, it is not the ability of the algorithm to detect outliers

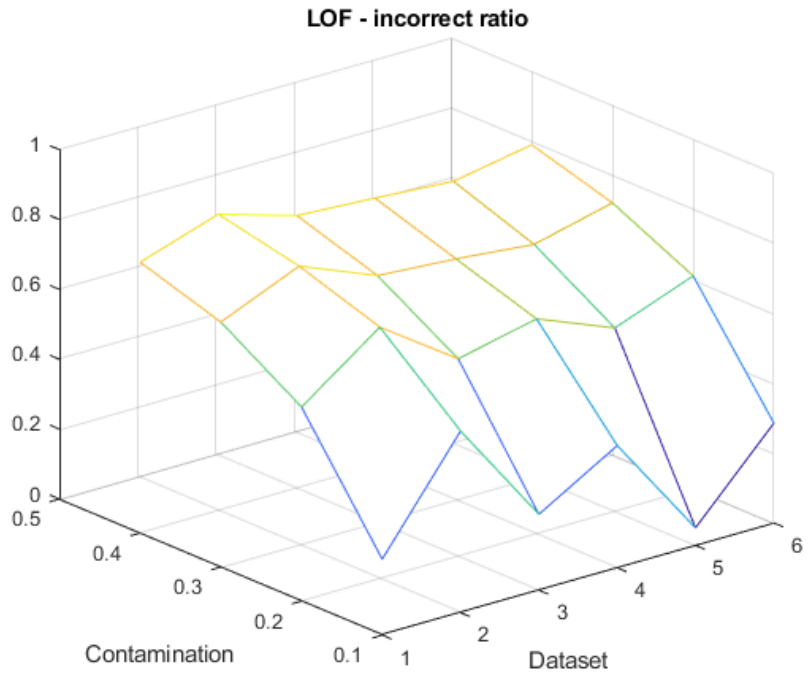

Figure 2: Ratio of false positives

for these cases, but rather high chances of real outliers to accidentally get into 70%-80% of discarded examples.

But the main conclusion is that the fact that these obvious observations are present confirms that the algorithm can correctly spawn outliers in its generated datasets.

5.2 k-Means

The results of k-Means algorithm is exactly the same as expected from it for such datasets. As the variables are normally distributed, it is trivial for the algorithm to correctly find the clusters given the correct guess of the cluster number and not extremely unlucky initialization. Thus, we can see that the adjusted mutual information score is equal to 1 for the case without outliers.

As k-Means can be very sensitive to outliers, we can see that their presence significantly worsens the results. If in addition to that the clusters are also very close and data contains irrelevant features, we can see that the score deteriorate quickly getting to the minimal value for the worst case dataset.

Such behavior is expected and also confirms the correctness of the algorithm.

Table 1: k-Means results

Dataset	Score
($p_2 = 2$, $J_0 = 0.01$, outliers = 20)	0.43403
($p_2 = 0$, $J_0 = 0.01$, outliers = 20)	0.56794
($p_2 = 2$, $J_0 = 0.21$, outliers = 20)	0.64046
($p_2 = 0$, $J_0 = 0.21$, outliers = 20)	0.57278
($p_2 = 2$, $J_0 = 0.342$, outliers = 20)	0.64270
($p_2 = 0$, $J_0 = 0.342$, outliers = 20)	0.62883
($p_2 = 2$, $J_0 = 0.01$, outliers = 0)	0.90623
($p_2 = 0$, $J_0 = 0.01$, outliers = 0)	0.78026
($p_2 = 2$, $J_0 = 0.21$, outliers = 0)	0.94275
($p_2 = 0$, $J_0 = 0.21$, outliers = 0)	1.0
($p_2 = 2$, $J_0 = 0.342$, outliers = 0)	1.0
($p_2 = 0$, $J_0 = 0.342$, outliers = 0)	1.0

5.3 DBSCAN

Table 2: DBSCAN

Dataset	(ϵ,min sample)	Score
($p_2 = 2$, $J_0 = 0.01$, outliers = 20)	(5,15)	0.25502
($p_2 = 0$, $J_0 = 0.01$, outliers = 20)	(3,5)	0.49960
($p_2 = 2$, $J_0 = 0.21$, outliers = 20)	(7,35)	0.75936
($p_2 = 0$, $J_0 = 0.21$, outliers = 20)	(5,5)	0.64515
($p_2 = 2$, $J_0 = 0.342$, outliers = 20)	(7,5)	0.78300
($p_2 = 0$, $J_0 = 0.342$, outliers = 20)	(7,30)	0.78722
($p_2 = 2$, $J_0 = 0.01$, outliers = 0)	(5,10)	0.60886
($p_2 = 0$, $J_0 = 0.01$, outliers = 0)	(3,5)	0.61420
($p_2 = 2$, $J_0 = 0.21$, outliers = 0)	(7,40)	0.92191
($p_2 = 0$, $J_0 = 0.21$, outliers = 0)	(5,30)	0.92623
($p_2 = 2$, $J_0 = 0.342$, outliers = 0)	(7,20)	1.0
($p_2 = 0$, $J_0 = 0.342$, outliers = 0)	(5,5)	1.0

Table 2 contains the highest results for DBSCAN algorithm among the tested parameters values. As the DBSCAN is a density based algorithm, it does not need to know in advance the exact number of clusters, as it derives it from the data. What it is sensitive to, though, is the structure of the data: if the clusters are rather distant or at least distinguished by shape or density, DBSCAN will identify them; on contrary, in the cases of close intersected and noisy clusters it would be almost impossible for it to get good results. That is exactly what we see in the table. We can notice severe drop of score for the separation degree of 0.01 when the clusters are almost united in one. The same drop is present for the cases where

the outliers are present, as they make the whole dataset more uniform. Noisy variable do not significantly affect the algorithm in most of the cases, at least with the current number of them.

Another thing about DBSCAN is its ability to efficiently identify and discard the outliers in most normal cases, as we can see on Figure 3.

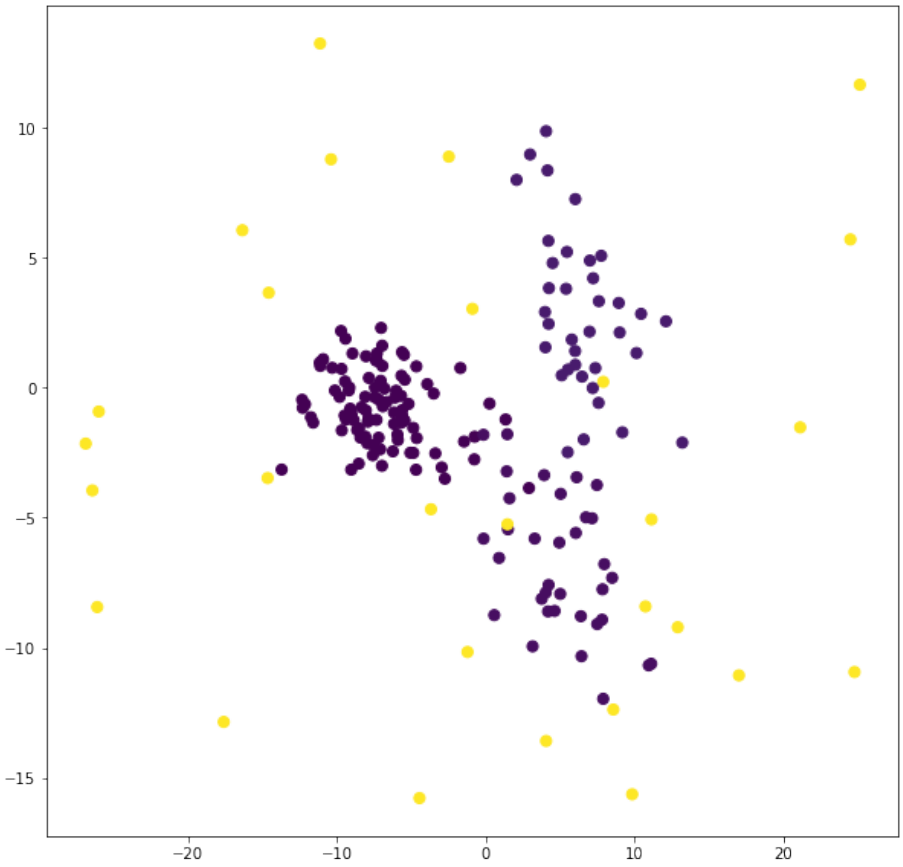

Figure 3: DBSCAN results, dataset 4, $\epsilon = 5$, ms= 5

Once again the algorithm behavior is an expected one.

5.4 Consensus clustering

As we can see, the results for the consensus clustering (number of clusters to use the base classifiers = 10, number of components = 30, non-random number of clusters in components) is approximately the same as for the k-Means. Though this may seem uninformative, it is not so: given the capabilities of this approach to adjust to complex spatial structures in the data if the cluster generation algorithm accidentally generated some unintended structures, consensus clustering method would have adjusted to them and improved the

Table 3: Consensus clustering results

Dataset	Score
($p_2 = 2$, $J_0 = 0.01$, outliers $= 20$)	0.44276
($p_2 = 0$, $J_0 = 0.01$, outliers $= 20$)	0.55503
($p_2 = 2$, $J_0 = 0.21$, outliers $= 20$)	0.64046
($p_2 = 0$, $J_0 = 0.21$, outliers $= 20$)	0.58724
($p_2 = 2$, $J_0 = 0.342$, outliers $= 20$)	0.64270
($p_2 = 0$, $J_0 = 0.342$, outliers $= 20$)	0.64280
($p_2 = 2$, $J_0 = 0.01$, outliers $= 0$)	0.90623
($p_2 = 0$, $J_0 = 0.01$, outliers $= 0$)	0.87429
($p_2 = 2$, $J_0 = 0.21$, outliers $= 0$)	0.94275
($p_2 = 0$, $J_0 = 0.21$, outliers $= 0$)	1.0
($p_2 = 2$, $J_0 = 0.342$, outliers $= 0$)	1.0
($p_2 = 0$, $J_0 = 0.342$, outliers $= 0$)	1.0

results compared to k-Means. The same results tell us our algorithm generates the clusters of the expected properties, not more, not less complex ones.

6 Conclusions

- The cluster generation algorithm was analyzed and successfully implemented.
- The behavior of the algorithm was tested and empirically shown to be the expected one.
- The algorithm was tested with different clustering algorithms that provided the expected results with respect to the "complexity" of the dataset.

References

[1] Weiliang Qiu and Harry Joe. Generation of random clusters with specified degree of separation. *Journal of Classification*, 23(2):315–334, Sep 2006.

[2] Glenn W. Milligan. An algorithm for generating artificial test clusters. *Psychometrika*, 50(1):123–127, Mar 1985.

[3] Underhill J. M. Kaiser H. A. Waller, N. G. A method for generating simulated plasmodes and artificial test clusters with user-defined shape, size, and orientation. *Multivariate Behavioral Research*, 34(2):123–142, 1999.

[4] Weiliang Qiu and Harry Joe. Separation index and partial membership for clustering. *Comput. Stat. Data Anal.*, 50(3):585–603, February 2006.

[5] Isabelle Guyon. Design of experiments for the NIPS 2003 variable selection benchmark. Technical report, July 2003.