Ben-Gurion University
of the Negev

Electrical and Computer Engineering Department

Clustering and Unsupervised Computational Learning

Report

Ofir Yaish, 313254716

1. Task 0 – Dataset Generation

1. Dataset type 1 - Data generated using random multivariate normal distributions:

The first datasets are based on the **multivariate normal distribution**:

$$f_{\mathbf{X}}(x_1, \ldots, x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}$$

The generator has different parameters that define the multivariate normal distribution from which the dataset is generated:

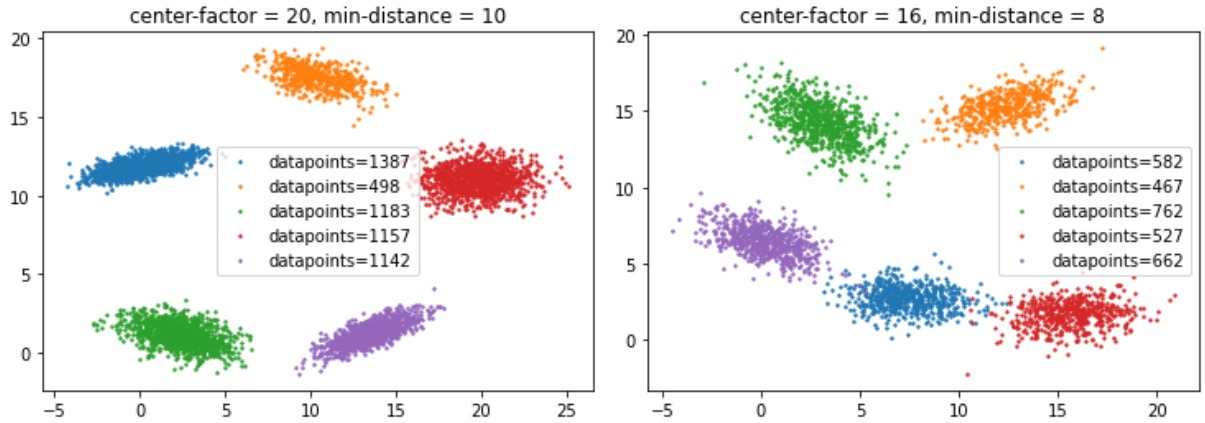
- c – the number of generated clusters.
- dimension – the dimension of the data.
- size-range – the size of each cluster is randomly chosen from the size-range. This allows achieving variability in the cluster sizes and, therefore, in the prior probabilities of the clusters.
- center-factor – the cluster centers values (μ vector values) is randomly chosen from the range $[0, center - factor)$. The higher the center-factor is, the higher probability of the clusters intersecting.
- min-distance – confirms that the cluster centers ($\mu's$) will have a minimum Euclidian distance of min-distance from one to the other. **Note** that the higher the center-factor values, the easier to fulfill the min-distance constraint.
- outliers – add outliers data points. The outliers are uniformly randomized.

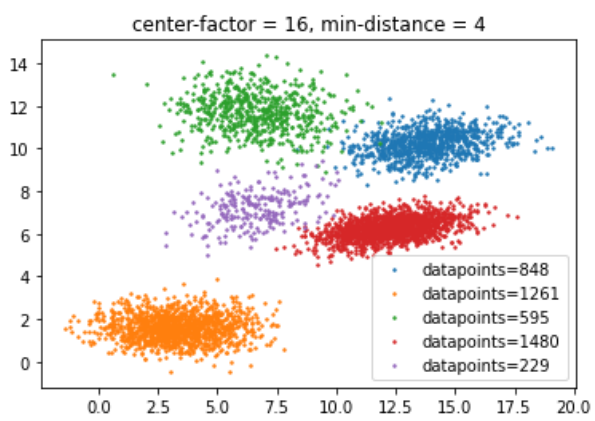
The covariance matrix (Σ) of each cluster is randomized to be a positive semi-definite matrix:

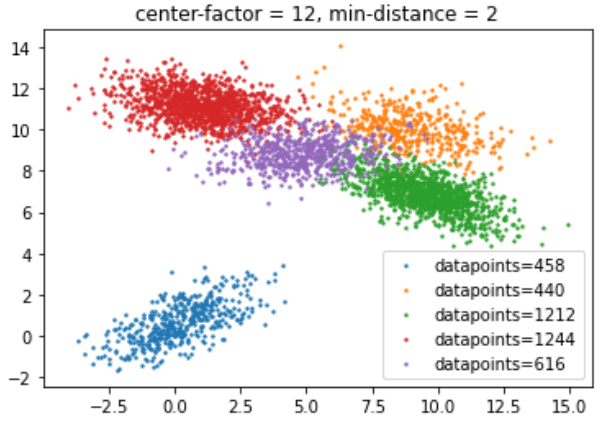
$$M \text{ positive semi-definite} \iff \mathbf{x}^{\mathrm{T}} M\mathbf{x} \geq 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n$$

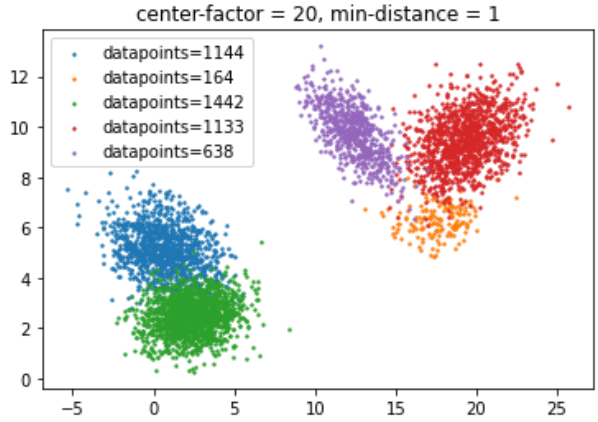
2D clusters:

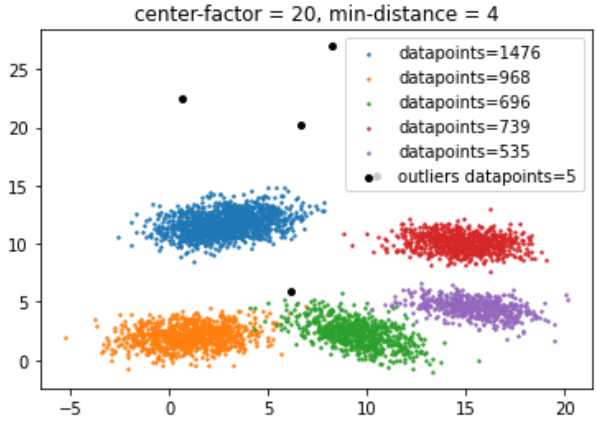
We can generate any number of clusters. For example, we show here different variations for 5 clusters:

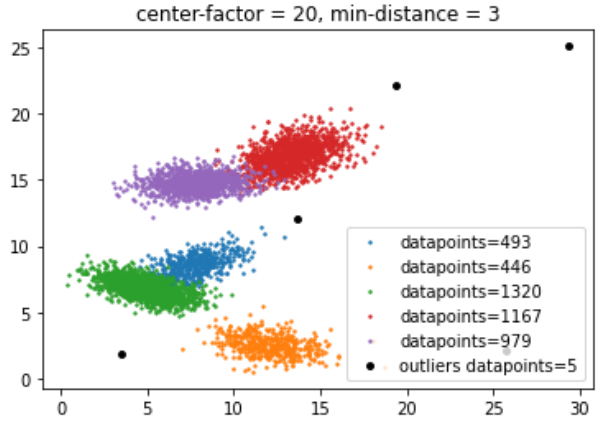


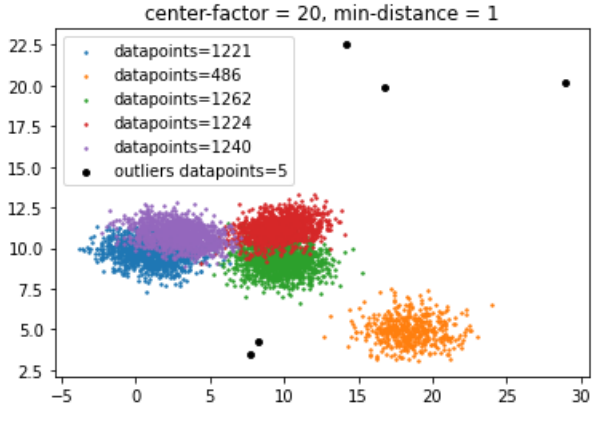

center-factor = 16, min-distance = 4
datapoints=848
datapoints=1261
datapoints=595
datapoints=1480
datapoints=229


center-factor = 12, min-distance = 2
datapoints=458
datapoints=440
datapoints=1212
datapoints=1244
datapoints=616


center-factor = 20, min-distance = 1
datapoints=1144
datapoints=164
datapoints=1442
datapoints=1133
datapoints=638

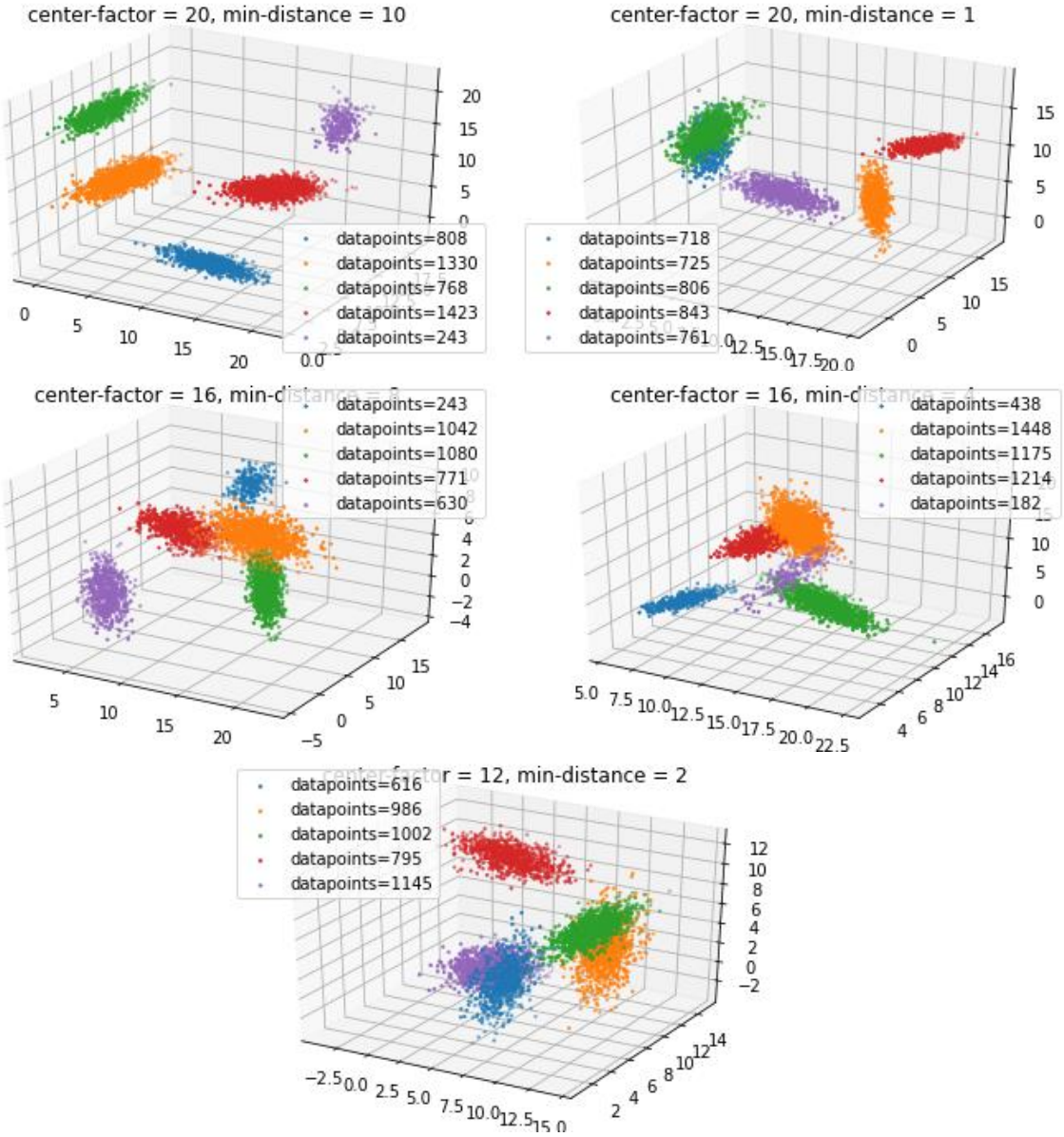

center-factor = 20, min-distance = 4
datapoints=1476
datapoints=968
datapoints=696
datapoints=739
datapoints=535
outliers datapoints=5


center-factor = 20, min-distance = 3
datapoints=493
datapoints=446
datapoints=1320
datapoints=1167
datapoints=979
outliers datapoints=5


center-factor = 20, min-distance = 1
datapoints=1221
datapoints=486
datapoints=1262
datapoints=1224
datapoints=1240
outliers datapoints=5

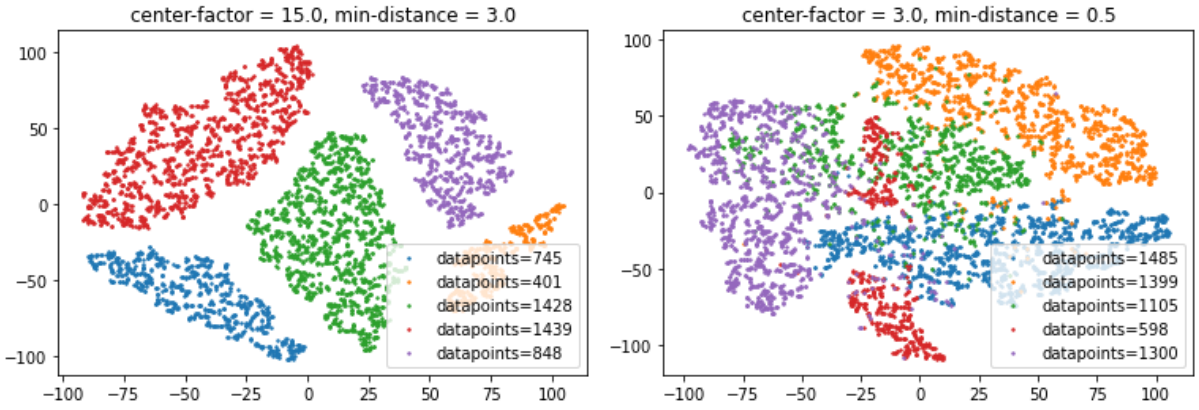
3D clusters:

Similarly, we can generate 3D datasets:



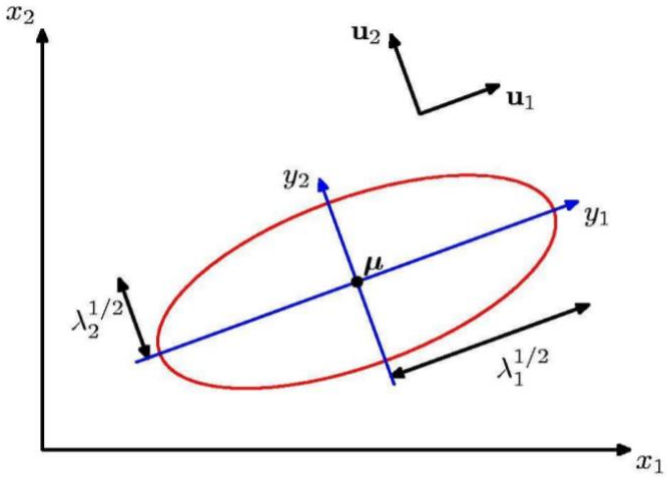
Higher dimension clusters:

As we move to higher dimensions, the visualization of the dataset is impossible. Therefore, we used a technique called **T-SNE** for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. Here are two examples of 5D datasets that were projected into the 2D space:



2. Dataset type 2 - Data generated using craft multivariate normal distributions:

The geometry of the multivariate normal distribution can be investigated by considering the orientation and shape of the prediction ellipse as depicted in the following diagram:



Where the ellipse has axes pointing in the directions of the unit eigenvectors u_1 and u_2 of Σ, and the height and wide correspond to the eigenvalues λ_1, λ_2. This way, we can generate, using the eigendecomposition, crafted multivariate normal distributions.

$$\Sigma = \begin{bmatrix} | & | \\ u_1 & u_2 \\ | & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} | & | \\ u_1 & u_2 \\ | & | \end{bmatrix}^T = U\Sigma U^T$$

Since U control the ellipse directions, we can start with a unit matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, and then rotate the matrix using a rotation matrix:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

which rotates points in the x-y plane counterclockwise through an angle θ.

Note that in order the rotate the shape, we also need to rotate the cluster center (μ):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

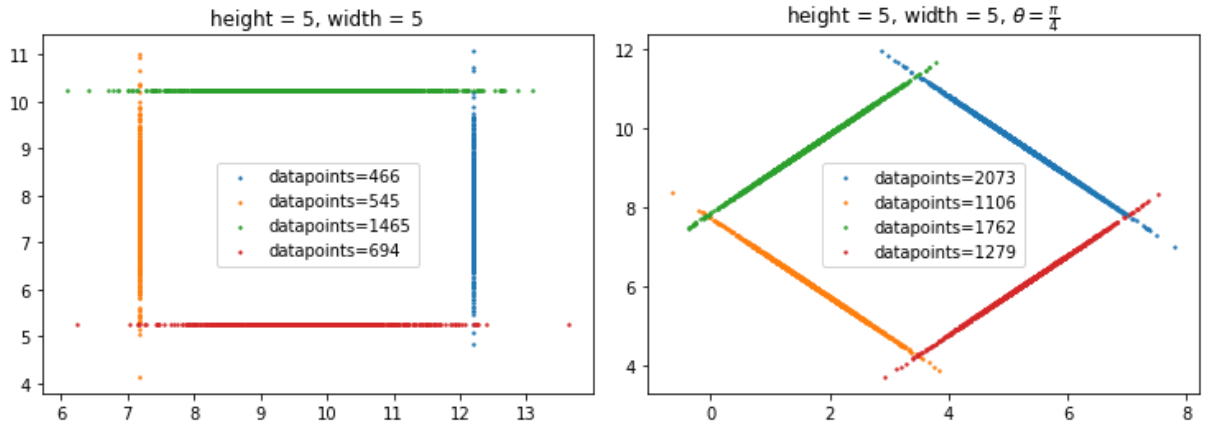
Rectangle shape:

To generate a straight-line-like shape cluster, one of the eigenvalues needs to be zero (for horizontal line - $\lambda_1 \to 0$ and vertical line - $\lambda_2 \to 0$), and the other defines its "length". In addition, we can control the following parameters:

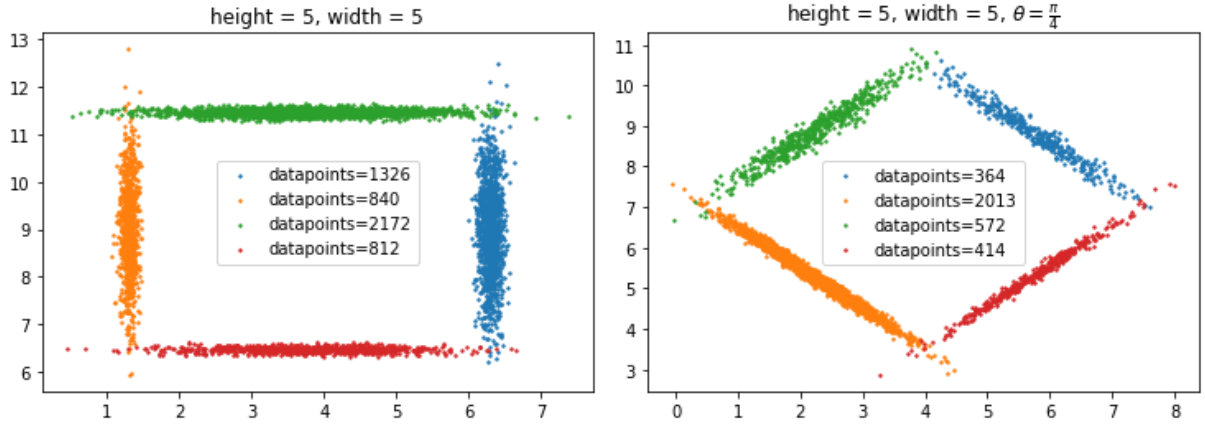
- center-factor – the center of the rectangle is randomly chosen from the range [0, $center - factor$).
- height, width – given the rectangle center, we can define the straight-line-like clusters' centers to form the height and width of the rectangle. Note that an appropriate line length needs to be formed.
- rotation-angle – define the rotation angle to rotate the entire Rectangle shape.

For example:

- A rectangle of height and width of 5, and its $\frac{\pi}{4}$ rotation:

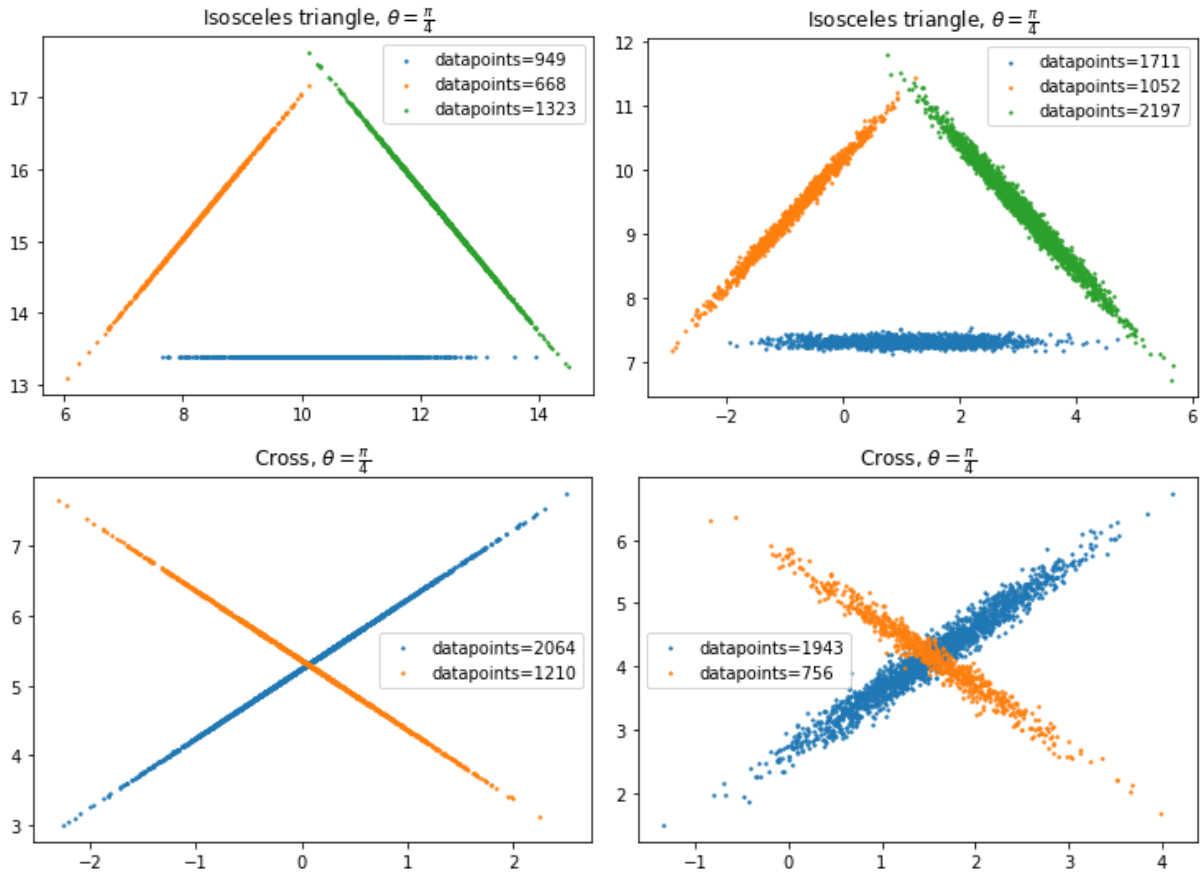


- A volume for each straight-line-like shape cluster can be obtained by modifying the eigenvalue (λ) that controls the vertical axis of the line. Here, for each "line", we randomized small λs to preserve the straight-line shape.



Triangle and cross shapes:

Similarly, we can generate Triangle and cross shapes using a combination of multivariate normal distributions:



3. Dataset type 3 - generated using parameterization and multivariate normal distributions noise:

Using the parametric equation of curves or surfaces, we expand the variety of exciting clusters that can be generated using multivariate normal distributions. For example, the position of a point that moves on a curve in three-dimensional space is determined by the time needed to reach the point starting from a fixed origin. If x, y, z are the coordinates of the point, the movement is thus described by a parametric equation

$$x = f(t)$$
$$y = g(t)$$
$$z = h(t),$$

where t is the parameter and denotes the time. We can sample t-points from a normal distribution and then generate the curve. In addition, a noise to the curve or the surface can be added using samples obtained using a multivariate normal distribution.

$$x = f(t) + e_1$$
$$y = g(t) + e_2$$
$$z = h(t) + e_3,$$

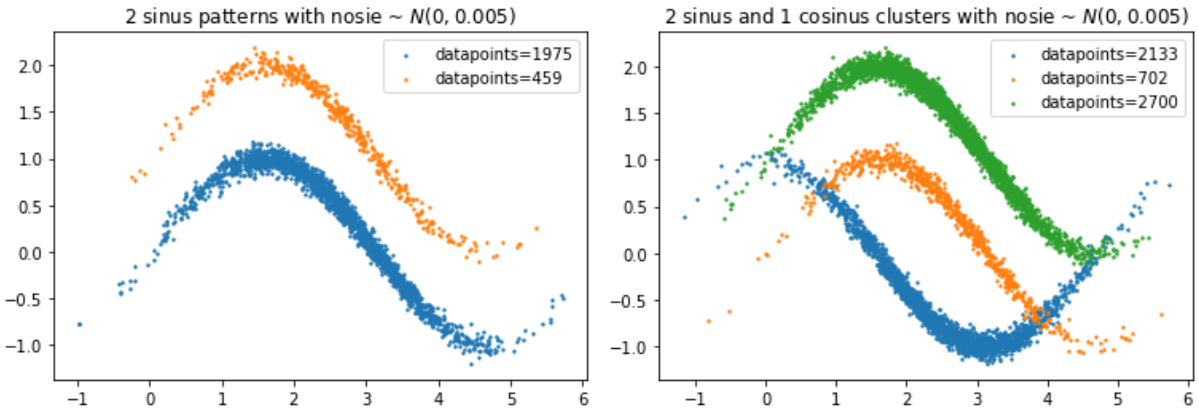
where e_i are sampled from a multivariate normal distribution $N(0, \Sigma)$.

Sinusoidal pattern:

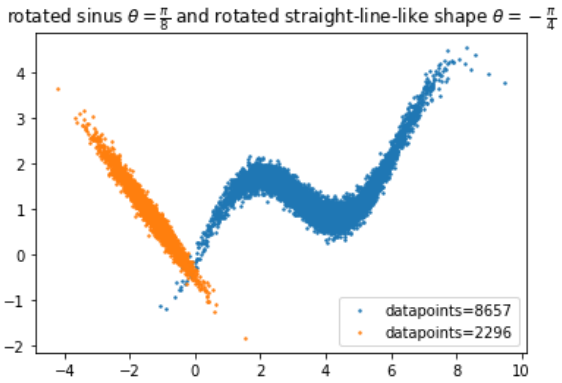
A simple example is generating a sinusoidal pattern cluster. The x-points ($x's$) and the noise ($e's$) are generated using a normal distribution, and the y-points ($y's$) are obtained using:

$$y(x, e) = \sin(x) + e \text{ or } y(x, e) = \cos(x) + e$$

For example, here are two examples of clusters where the x-points and the noise are independent variables:



In addition, we can generate rotated sinusoidal clusters by sampling x-points and the noise from a multivariate normal distribution with dependent variables. It is multiplying the unit covariance matrix by the rotation matrix:

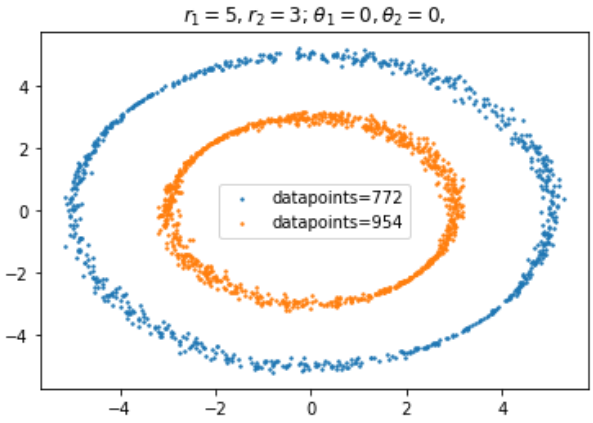


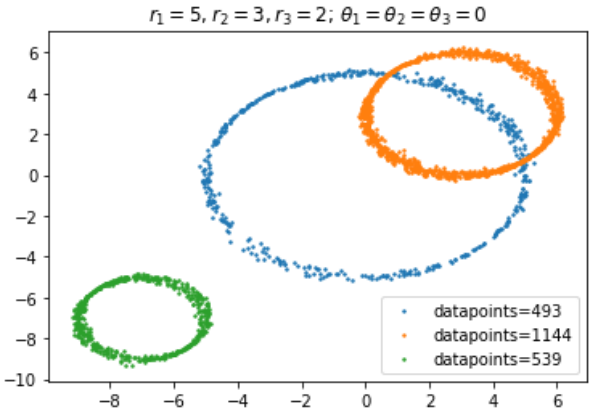
Circles and sphere patterns

Using the same mythology, we use the parametric equation for the ellipse:

$$x = x_0 + a\ cos(\theta) + e_1$$
$$y = y_0 + b\ cos(\theta) + e_2$$

To generate circle pattern clusters (in circle $a = b$) where θ and the noise are sampled from a multivariate normal distribution.
For example, here are two examples of clusters where the x-points and the noise are independent variables:

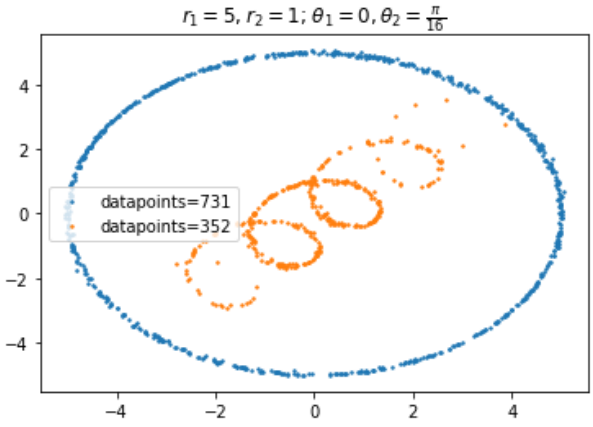


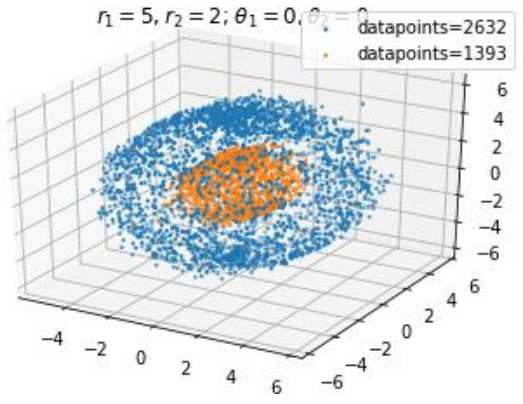


As before, we can also sample the noise and θ from a dependent multivariate normal distribution or use to use the parametric equation for a sphere

$$x = x_0 + r\sin(\theta)\cos(\phi)$$
$$y = y_0 + r\sin(\theta)\sin(\phi)$$
$$z = z_0 + r\cos(\phi)$$

to achieve some interesting patterns:



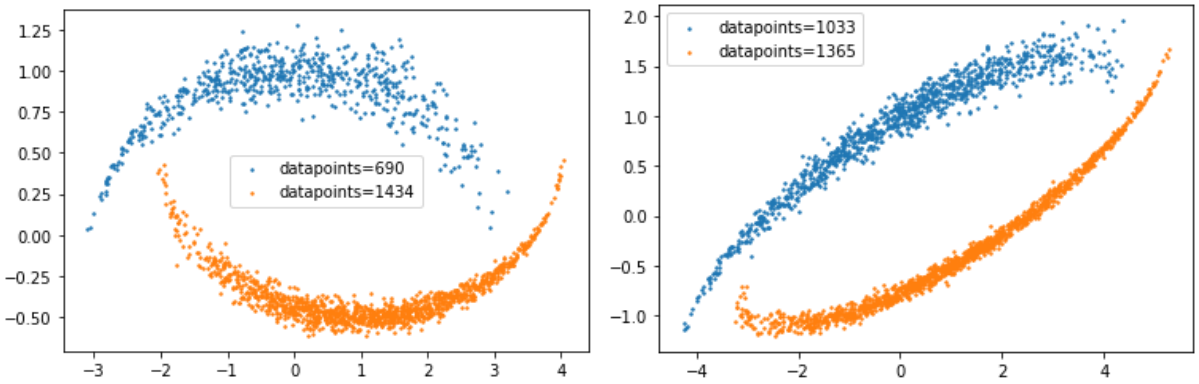


Moon pattern:

Using the parametric equation for half ellipse:

$$x = x + e_1, \qquad |x| \leq a$$
$$y = \frac{b}{a}\sqrt{a^2 - x^2} + e_2$$

we can generate two interleaving half ellipses, and rotate them by making the x-points and the noise sampled from a dependent multivariate normal distribution:



4. Dataset type 4 - Real datasets:

Iris dataset:

It includes three iris species with 50 samples each and some properties of each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

Number of Instances: 150 (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the class

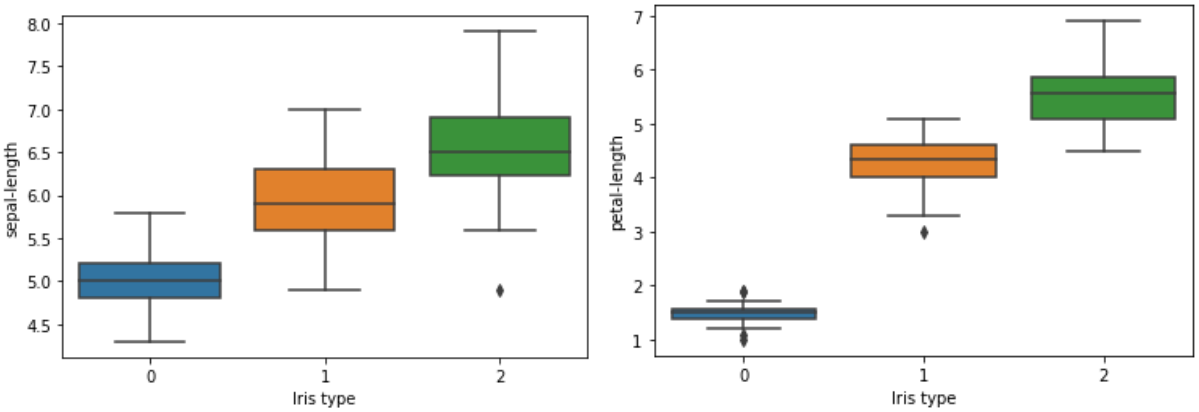
Attribute Information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
    - Iris-Setosa
    - Iris-Versicolour
    - Iris-Virginica

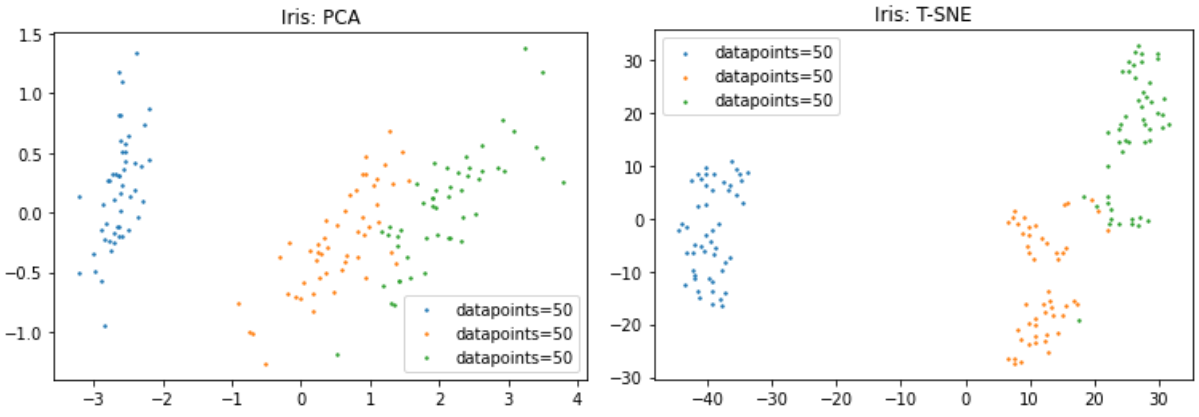
```
:Summary Statistics:

============== ==== ==== ======= ===== ====================
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
============== ==== ==== ======= ===== ====================
```

The high correlation to the class might indicate that this feature is important for separation:



We can visualize the dataset in 2D using dimension reduction techniques PCA or T-SNE:



We notice the Iris type that is entirely spreadable from the other two.

Wine dataset:

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

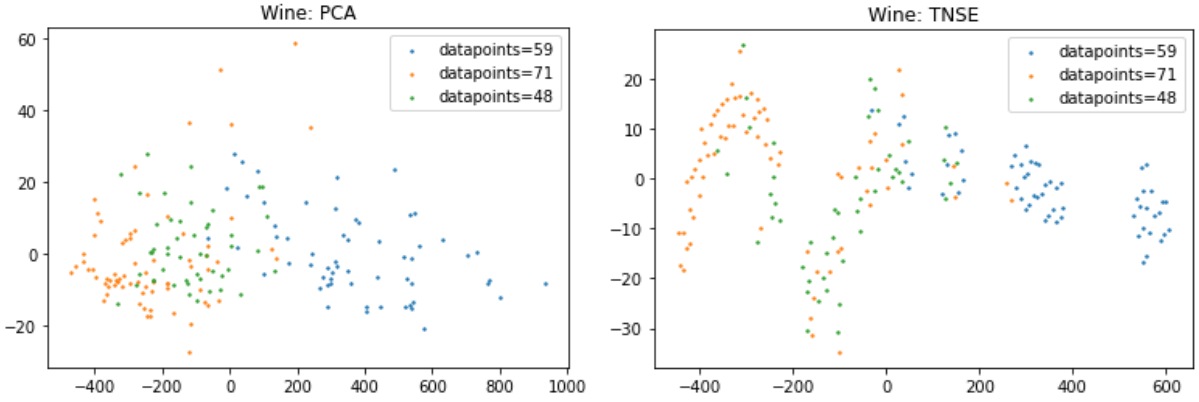
Number of Instances: 178

Number of Attributes: 13 numeric, predictive attributes and the class

Attribute Information:

- Alcohol
- Ash
- Magnesium
- Flavanoids
- Proanthocyanins
- Hue
- Proline
- Malic acid
- Alcalinity of ash
- Total phenols
- Nonflavanoid phenols
- Color intensity
- OD280/OD315 of diluted wines
- class:
    - class_0 (59)
    - class_1 (71)
    - class_2 (48)

In this dataset, the PCA and T-SNE fail to visualize the classes as distinct clusters:



2. Task 1 – Maximum Likelihood Estimate (MLE)

In this task, we implemented the maximum likelihood estimate (MLE), based on the Expectation Maximization (EM) algorithm, for clustering a dataset of unknown classes. We tested the algorithm on the datasets generated in task 0.

The EM algorithm assumes that the nature of the probability function is multivariate normal. For this reason, we expect the algorithm to have better results on the gaussian dataset than on the other datasets.

Assuming the number of classes known (C) and that each class is normally distributed, we wish to find the distribution parameters that maximize the log-likelihood:

$$\widehat{\Theta} = arg\,max_{\Theta}\, ln[p(X|\Theta)] = arg\,max_{\Theta} \sum_{k=1}^{n} ln[p(x\_k|\Theta)]$$

where $\Theta = \{\mu_i,\ \Sigma_i,\ P(\omega_i) - \text{aprior probabilities}\}_{i=1}^{C}$

The algorithm:

1. Set initial parameters by using one of the following methods:
    a. -Randomly select samples as expectation vectors: $\{\mu_i\}_{i=1}^{C}$.
    -Set covariance matrices: $\{\Sigma_i\}_{i=1}^{C}$ to be the identity matrix I.
    -Set uniform apriori probabilities: $\{P(\omega_i)\}_{i=1}^{C}: P(\omega_i) = \frac{1}{C}; \forall i$
    b. K-means and calculating covariance for the obtained expectation vectors and apriori according to each cluster's relative number of observations.
2. Calculate the posterior probability:

$$p(\omega_i|x_n, \mu_i, \Sigma_i) = \frac{P(\omega_i)p(x_n|\omega_i, \mu_i, \Sigma_i)}{\sum_{j=1}^{C} p(\omega_j)p(x_n|\omega_j, \mu_j, \Sigma_j)}$$

   where the likelihood probability is the multivariate normal probability density function.

3. Update the parameters:

$$P(\omega_i) = \frac{1}{N}\sum_{k=1}^{N} p(\omega_i|x_k, \mu_i, \Sigma_i)$$

$$\mu_i = \frac{\sum_{k=1}^{N} p(\omega_i|x_k, \mu_i, \Sigma_i)\, x_k}{\sum_{k=1}^{N} p(\omega_i|x_k, \mu_i, \Sigma_i)}$$

$$\Sigma_i = \frac{\sum_{k=1}^{N} p(\omega_i|x_k, \mu_i, \Sigma_i)\, (x_k - \mu_i)(x_k - \mu_i)^T}{\sum_{k=1}^{N} p(\omega_i|x_k, \mu_i, \Sigma_i)}$$

4. Stopping condition: go back to step 2 if the change in parameters is larger than some ϵ.
5. Having the optimal parameters, we can classify each to the class.

V-measure

Given the knowledge of the ground truth class assignments of the samples, it is possible to define some intuitive metrics using conditional entropy analysis.

In particular, Rosenberg and Hirschberg (2007) define the following two desirable objectives for any cluster assignment:

- Homogeneity (h): each cluster contains only members of a single class.
- Completeness (c): all class members are assigned to the same cluster.

Given those definitions, they also define the V-measure as the harmonic of the two:

$$v = \frac{2 \times h \times c}{h + c}$$

which is bounded below by 0.0 and above by 1.0 (higher is better).

Homogeneity and completeness scores are formally given by:

$$h = 1 - \frac{H(C|K)}{H(C)}$$

$$c = 1 - \frac{H(K|C)}{H(K)}$$

where $H(C|K)$ is the conditional entropy of the classes given the cluster assignments:

$$H(C|K) = \sum_{c=1}^{|C|} \sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \log\left(\frac{n_{c,k}}{n_k}\right)$$

and $H(C)$ is the entropy of the classes:

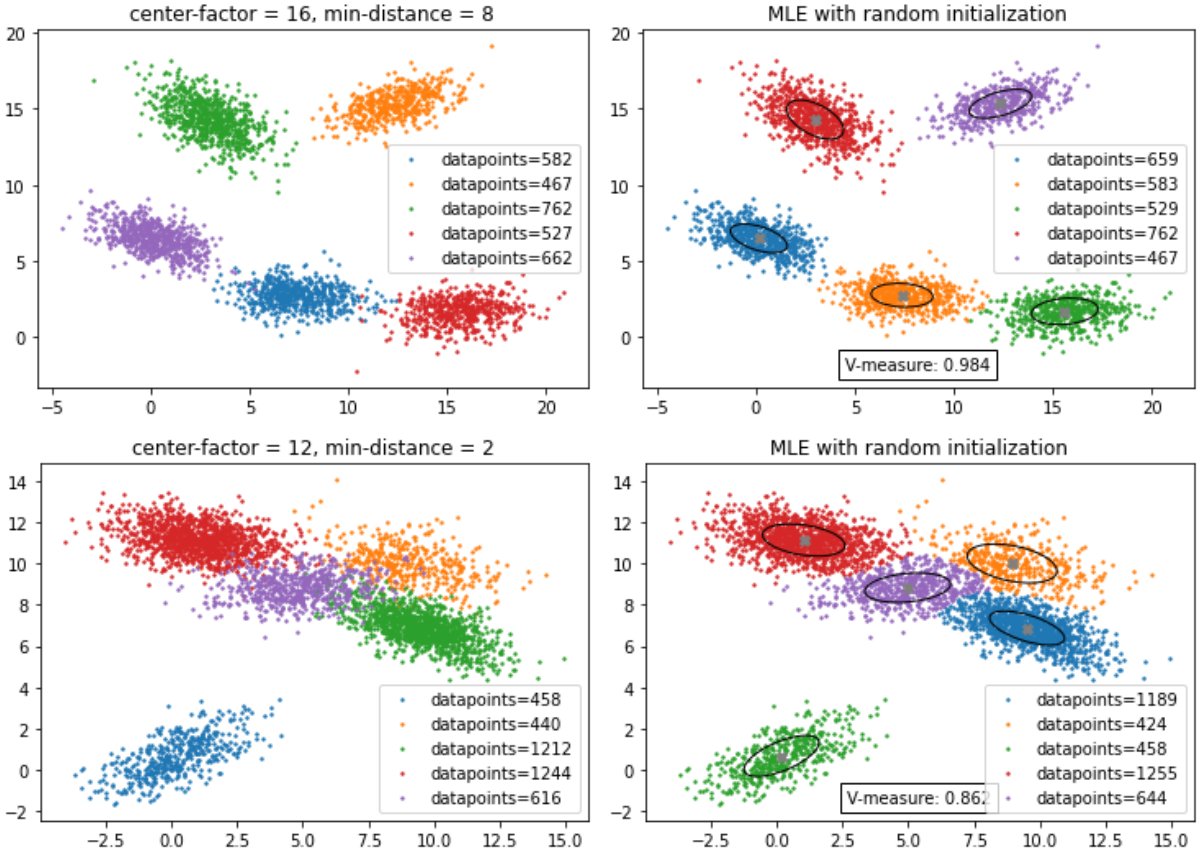
$$H(C) = \sum_{c=1}^{|C|} \frac{n_c}{n} \log\left(\frac{n_c}{n}\right)$$

with n the total number of samples, n_c and n_k the number of samples belonging to class c and cluster k, and finally $n_{c,k}$ the number of samples from class c assigned to cluster k.

The conditional entropy of clusters given class $H(K|C)$ and the entropy of clusters $H(K)$ are defined symmetrically.

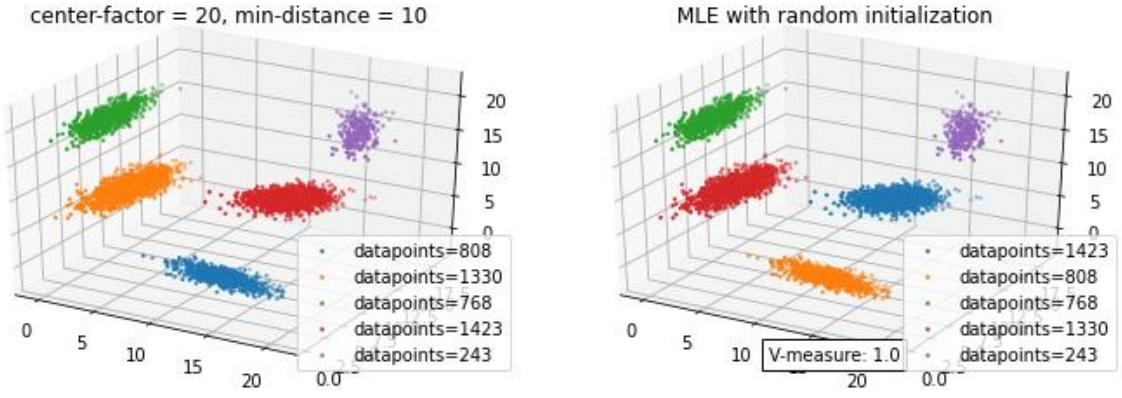
1. Dataset type 1 - Data generated using random multivariate normal distributions:

As expected, the MLE model, even with the random initialization, succeeded very well with the most sparable example that we generated as it assumes normality of the data:

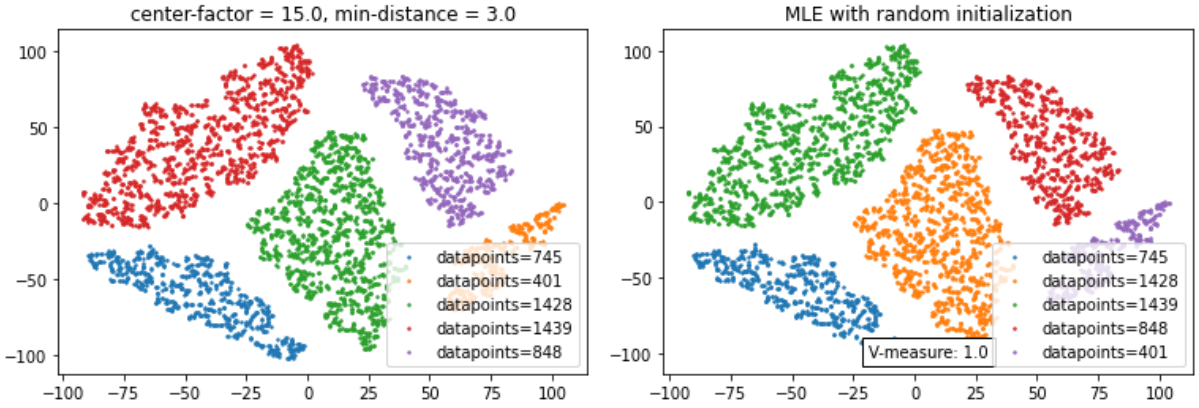


Similarly, in higher dimensions:

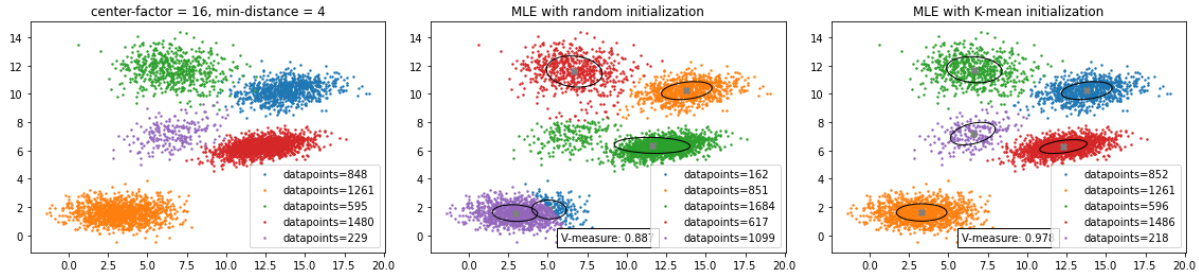
3-dimensional data



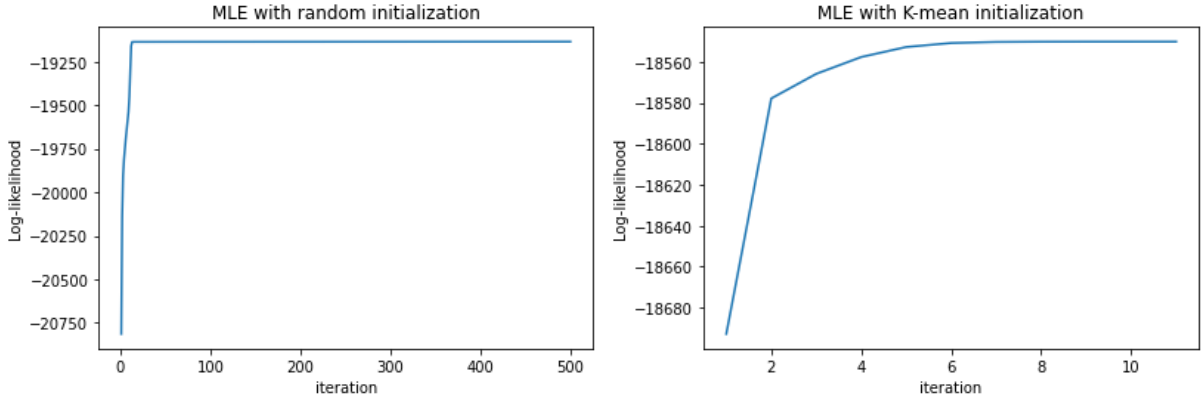
5-dimensional data



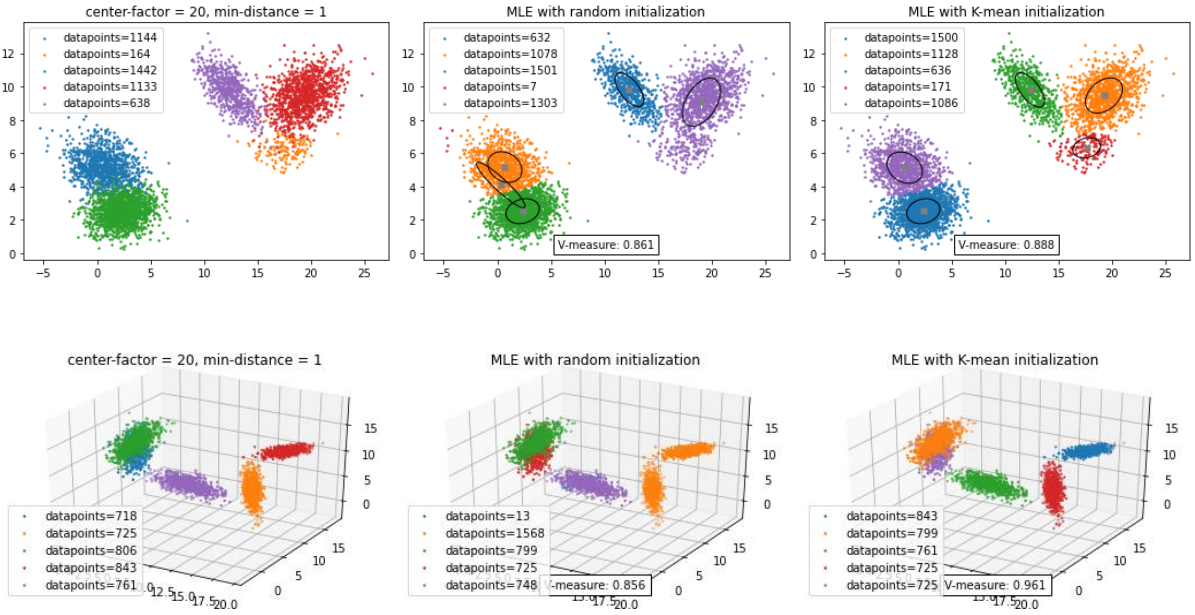
However, since the MLE algorithm is highly dependent on the initial centroids, some random initializations provide unsatisfactory performance in more complicated sets. In addition, we noticed that initializing the centroids with the K-means outcomes usually improves the performance significantly:



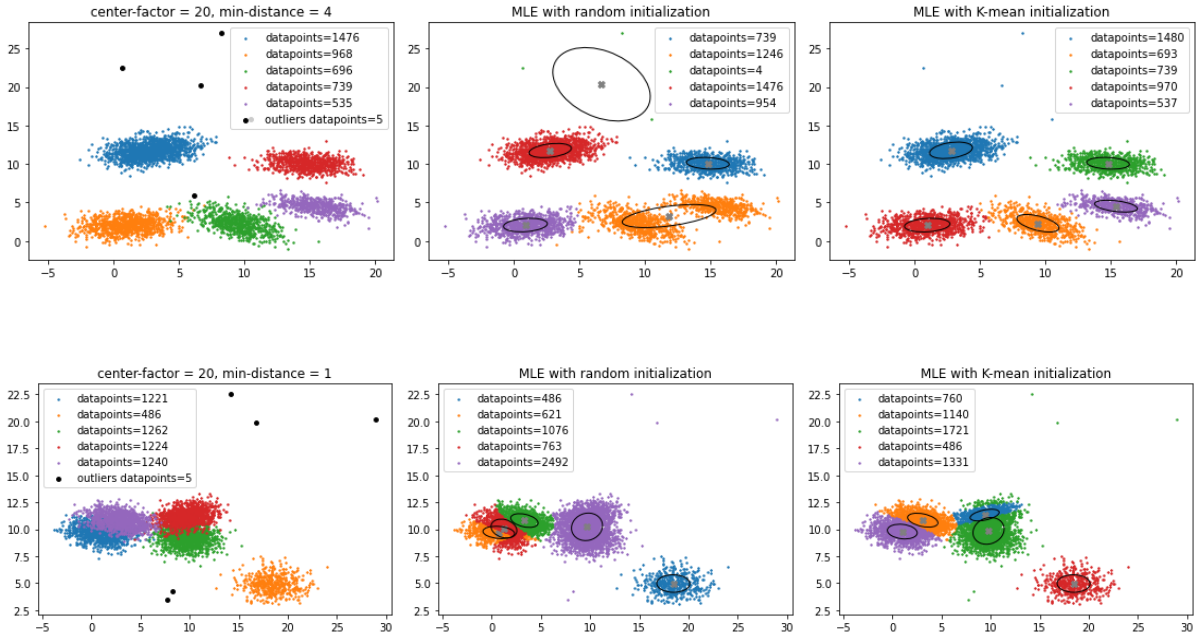
In this example, the log-likelihood of random initialization run did not converge even after 500 iterations that we set as the maximum number of iterations, while in the K-means initialization, it around 10 iterations to coverage:



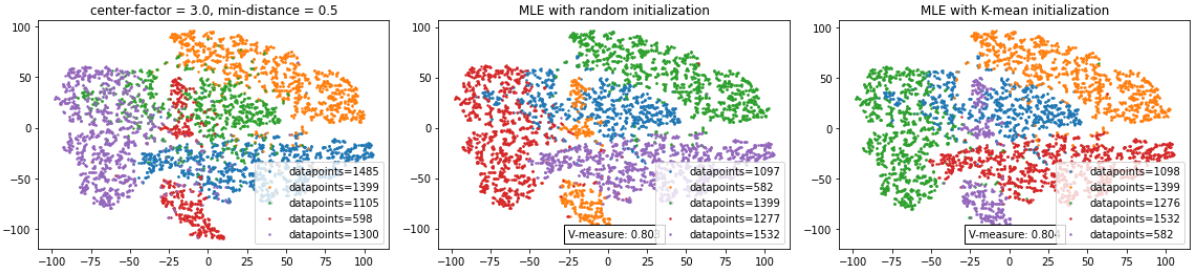
Here are additional sets in which the K-means initialization improved the MLE performance significantly:



Interestingly, the MLE algorithm with the random initialization performed poorly on the dataset with the outliers, and the K-means initialization managed to overcome the outliers:



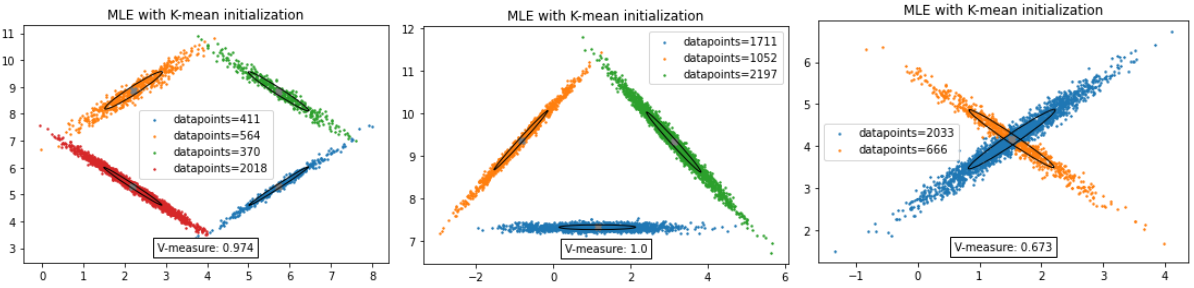
Nevertheless, in some cases, the K-means initialization did not provide a noticeable benefit as in the following 5-dimensional example. Probably due to the nature of the clusters that have small min-distance:



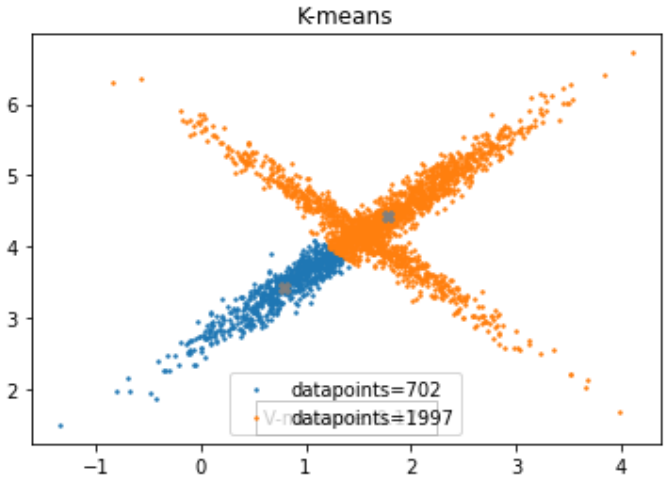
To conclude, we noticed that the MLE performs better with the K-means initialization, and therefore, in the following test, we will show only this variant.

2. <u>Dataset type 2 - Data generated using craft multivariate normal distributions:</u>

While being able to generate some interesting patterns with this variant of data, the clusters are easy to detect, as we can see below:

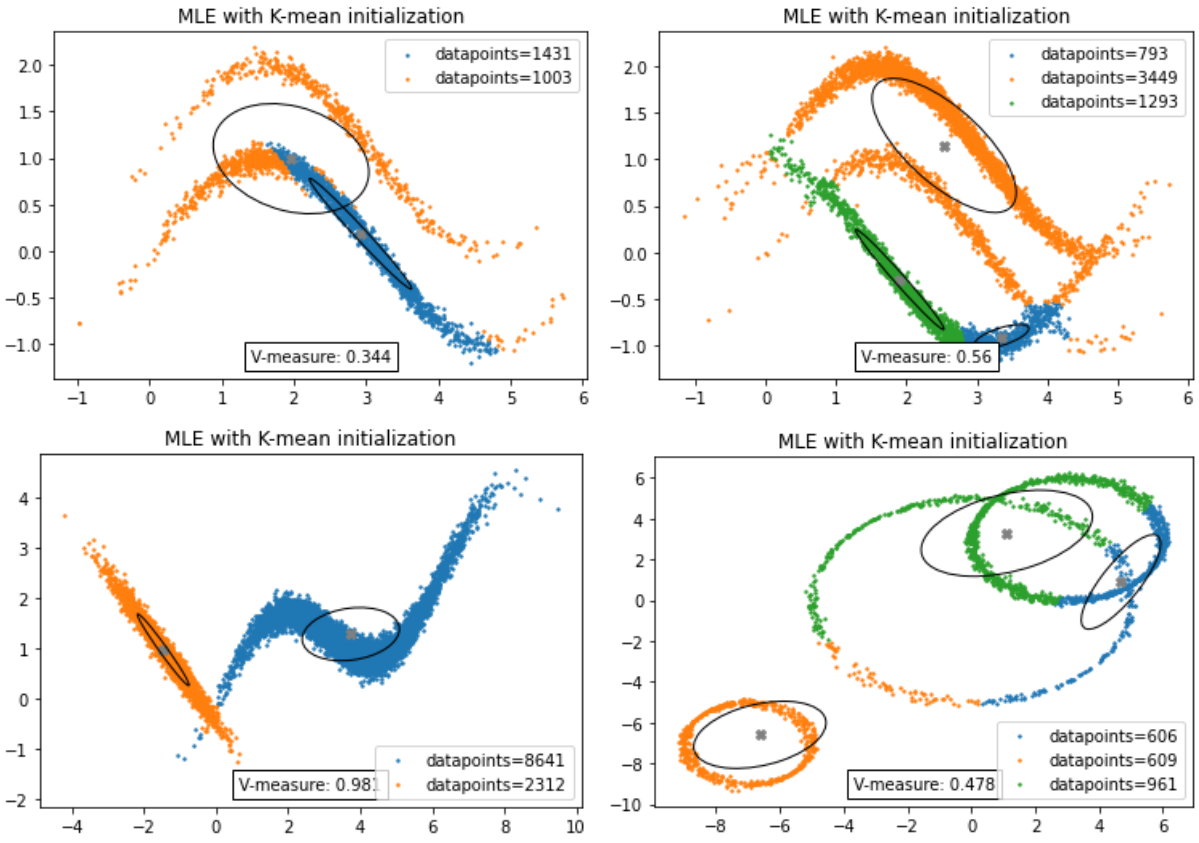


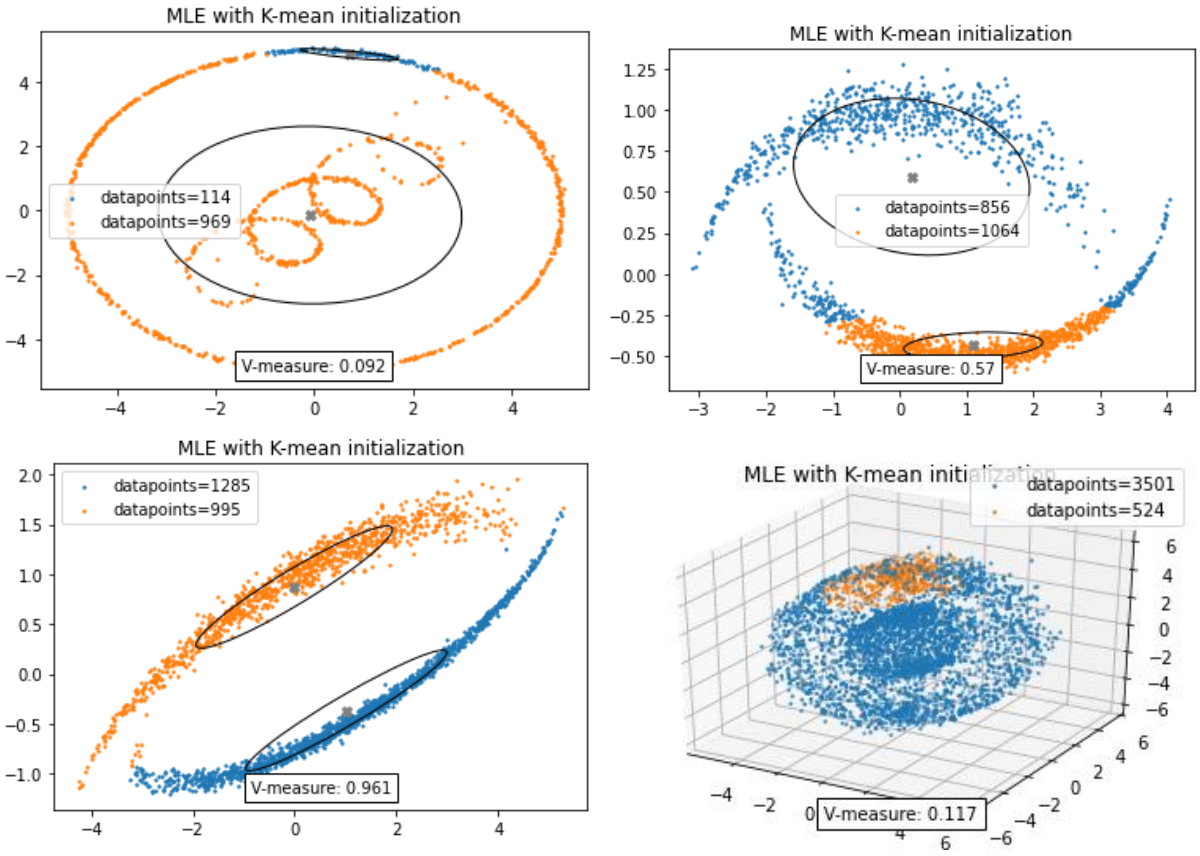
The last example is interesting since the model successfully clusters most of the points (not in the intersection) thanks to the covariance matrix it learned. In this case, a K-means algorithm will fail since it tries to learn only the mean of the data:



3. Dataset type 3 - generated using parameterization and multivariate normal distributions noise:

In this dataset, the patterns do not form gaussian shapes (not normally distributed), and therefore, we expect it to perform worse than in previous datasets:



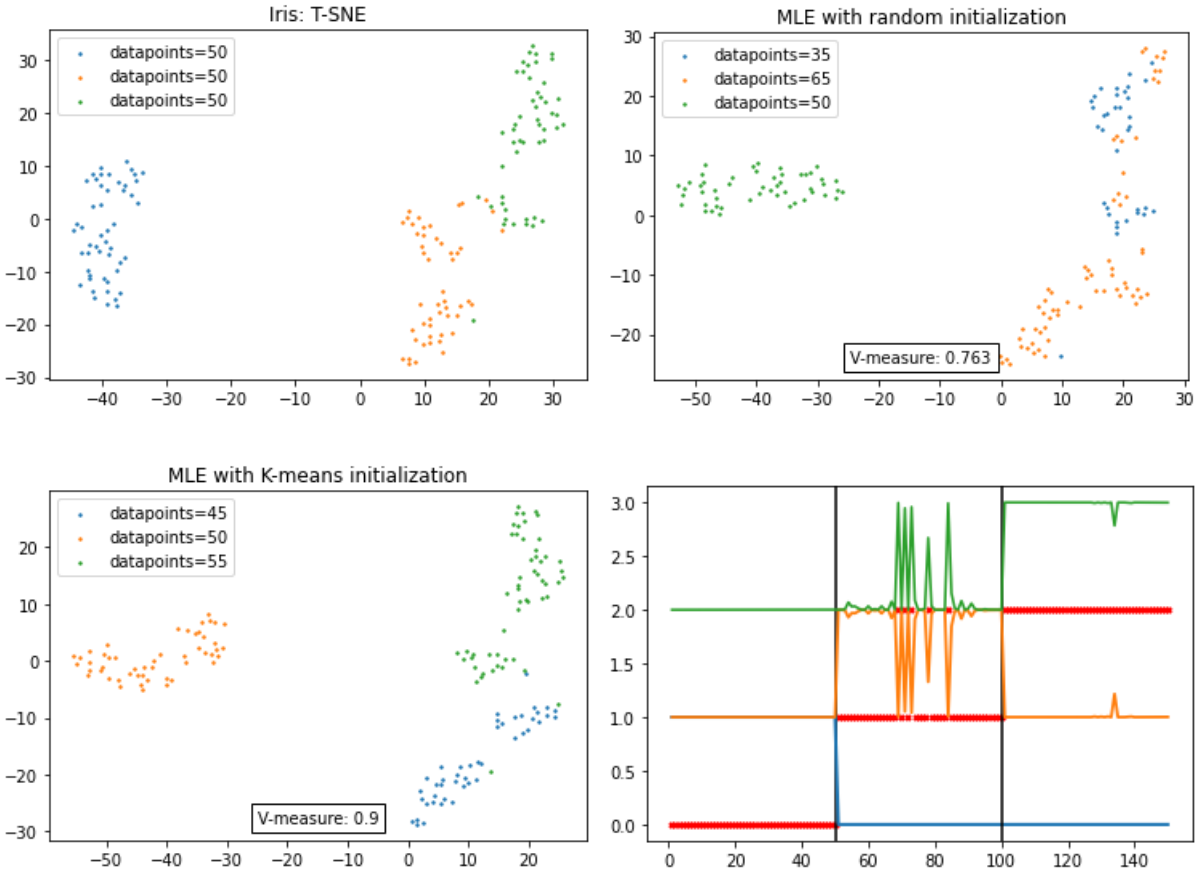


It is interesting to see the ellipse shapes that the MLE learned as they indicate the inability of the model to cluster those patterns with the parameters it learned.

4. Dataset type 4 - Real datasets:

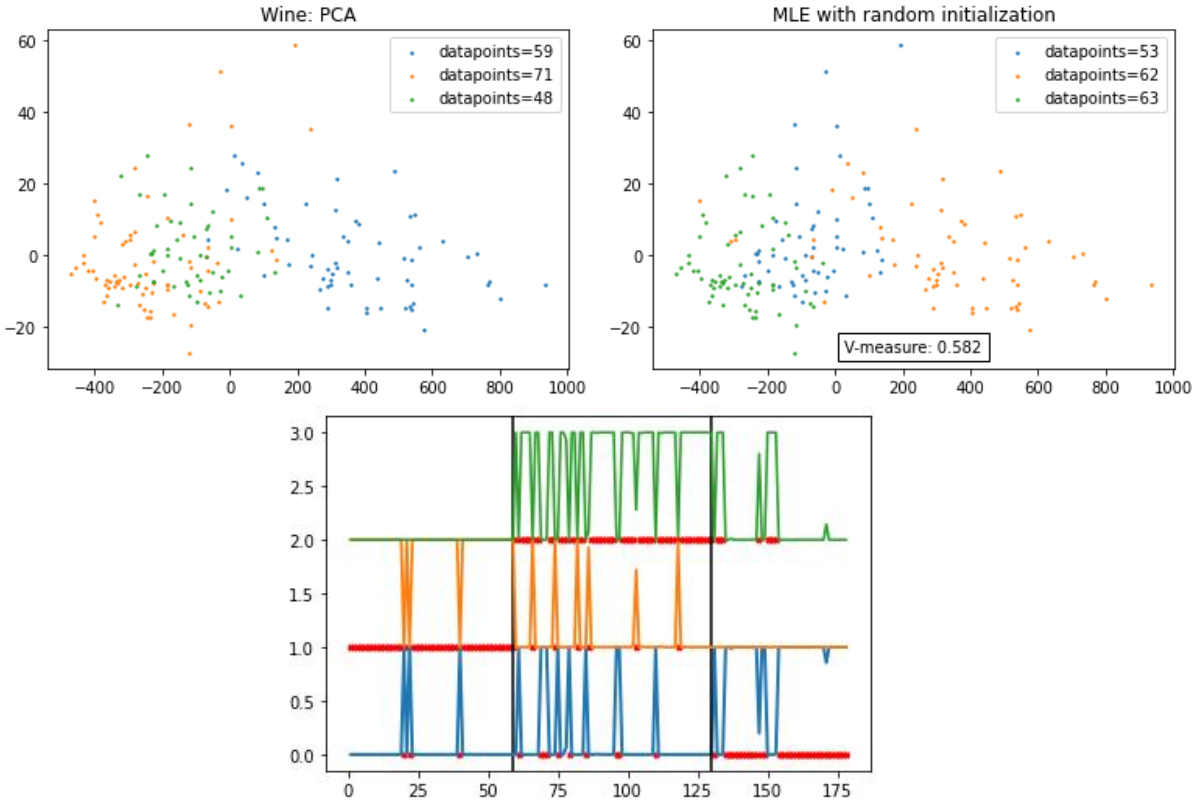
The last type of datasets that we tested is the Iris and Wine datasets. Since many nature measurements are distributed normally, the MLE should be a good option for a baseline model.

First, we tested the performance on the Iris dataset. Here are the results:



As we can see again, the MLE is highly dependent on the initialization, and the K-means initialization achieved very good performance with only 5 misclassified samples. As expected, one of the clusters is entirely separable.

Second, we tested the performance on the Wine dataset, which should be much more complicated. Here are the results:



While it is hard to notice the separation in the PCA (or T-SNE) visualization, we can see in the last figure that the number of mistakes made by the model is reasonable for a dataset that is harder to cluster than the Iris. This is probably due to its high dimensionality.

3. Task 2 – Unsupervised Optimal Fuzzy Clustering (UOFC)

In this task, we implemented the Unsupervised Optimal Fuzzy Clustering (UOFC) based on the fuzzy c-means (FCM) algorithm. The primary goal of the UOFC algorithm is to find the optimal number of clusters and cluster the data accordingly (with fuzzy memberships).

The algorithm starts with one cluster (centroid located in the mean of the entire data) and gradually increases the number of clusters in each iteration by adding an imaginary centroid in a region of low memberships. Each iteration starts with the FCM, followed by the modified fuzzy c-means algorithm based on maximum likelihood estimation (FMLE). After each iteration, we calculated the different criteria of clustering validity. In the following examples, we will show the clusters based on the optimal number found by the model using a majority vote of the different criteria. Here is the list of the different criteria that we would like to maximize:

1. The 1/hypervolume criterion: $\frac{1}{HV}$ where HV is the fuzzy hypervolume of all c clusters.

2. The partition density criterion (PD):

$$\frac{\text{The sum of "memberships" in all c clusters of the central members}}{\text{the fuzzy hypervolume of all c clusters}}$$

3. The average partition density using the central members' criterion (APDC):

$$\frac{1}{c}\sum_{i=1}^{c}\frac{\text{The sum of "memberships" in the } i\text{th cluster of the central members}}{\text{the fuzzy hypervolume of the } i\text{th cluster}}$$

4. The average partition density using the maximal members' criterion (APDC):

$$\frac{1}{c}\sum_{i=1}^{c}\frac{\text{The sum of "memberships" in the } i\text{th cluster of the maximal members}}{\text{the fuzzy hypervolume of the } i\text{th cluster}}$$

5. The normalized partition indexes criterion: $\frac{1}{NPI}$ where NPI FCM loss function is multiplied by c to consider the number of clusters.

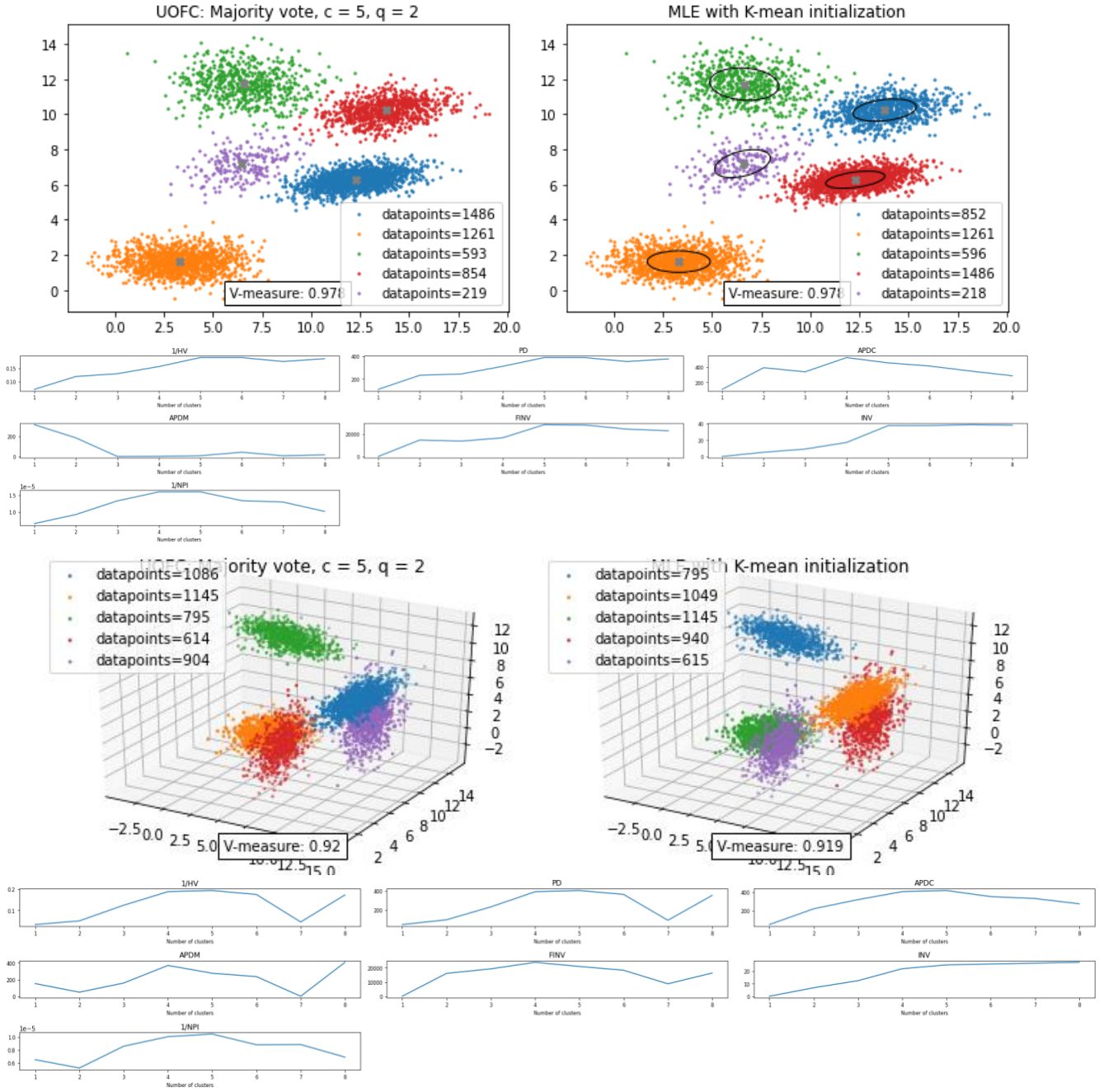
6. The invariant criteria (INV): $tr(S_W^{-1}S_B)$ where S_W and S_B are the within- and between-cluster scatter matrices, respectively.

7. The fuzzy invariant criteria (FINV): same as INV where S_W is the sum of the fuzzy covariance matrices obtained by the UOFC algorithm, and S_B is formed using the centroids obtained by the algorithm.

Generally, we noticed that the UOFC algorithm performs similarly to the MLE algorithm as it is based on the same Mahanalobis distance. Therefore, we will mainly focus on the compelling examples:

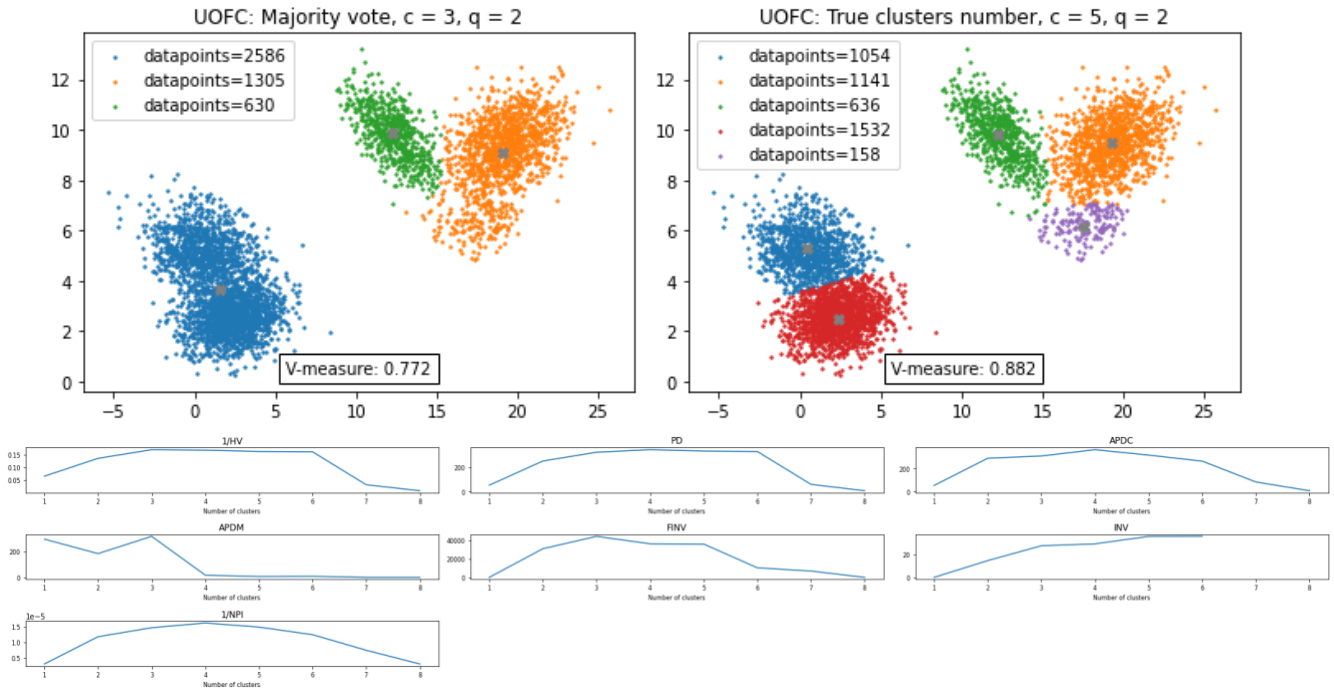
1. **<u>Dataset type 1 - Data generated using random multivariate normal distributions:</u>**
   Generally, the UOFC, with our majority voting, succeeds in finding the correct number of clusters.

   We will start with the datasets that the UOFC was able to find their optimal number of clusters:

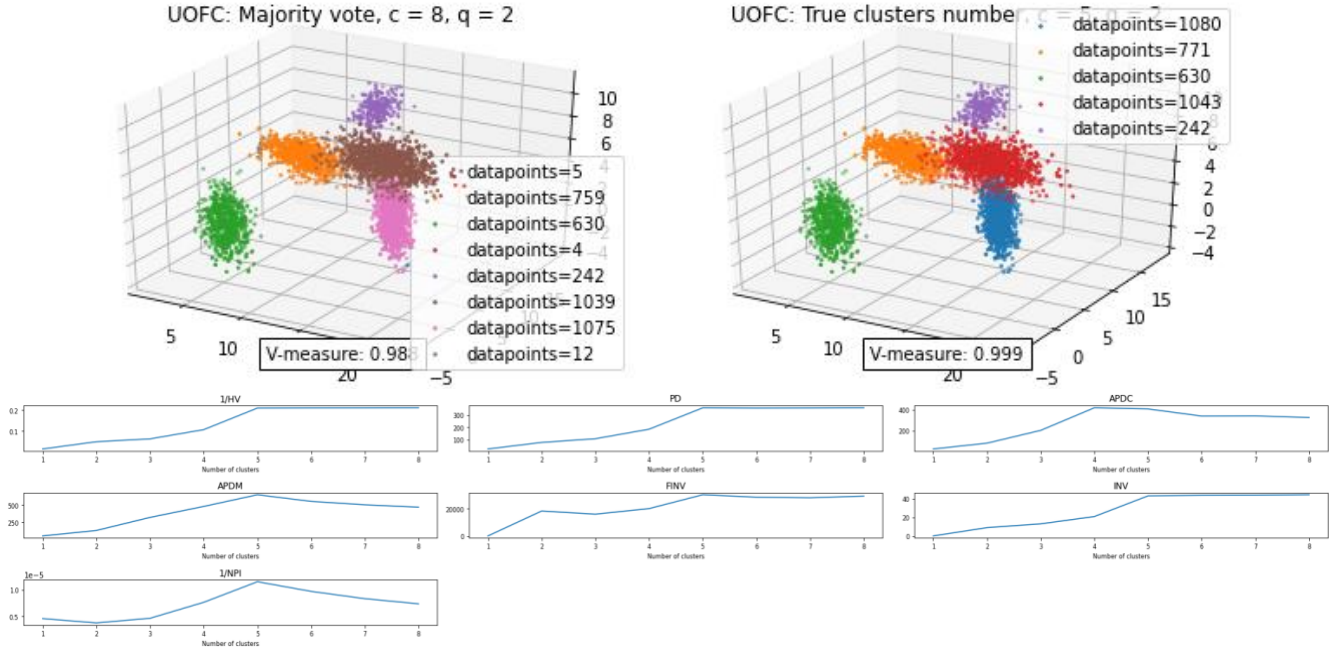


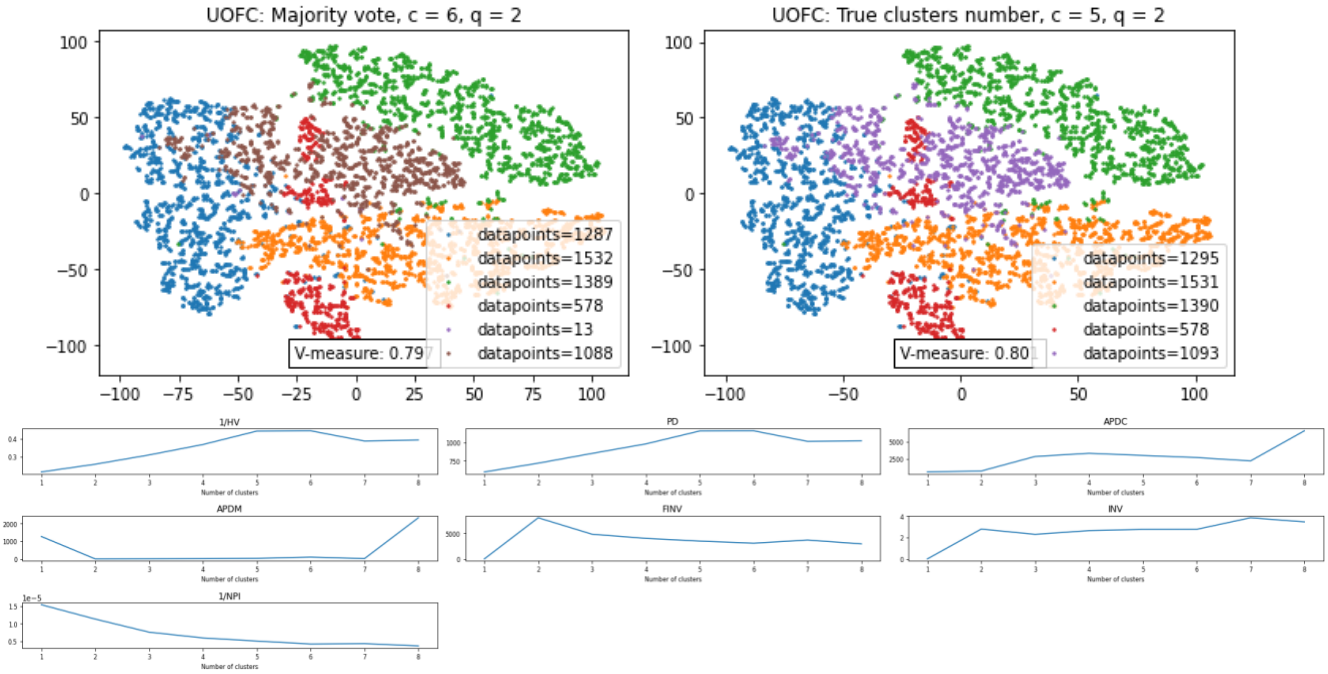
   As we can notice from the examples above, while some criteria identify the number of 5 clusters as the optimal one, when we used the majority vote, we could confidently say that the number of clusters is 5. In addition, as expected, the UOFC algorithm obtained similar results to the MLE algorithm (according to the V-measure).

In the following examples, we will show datasets where our approach based on the UOFC failed to find the optimal number of clustering:



While the number of clusters is incorrect, the number of three clusters is very reasonable due to the shape of the Gaussian. However, as we will see in the following two examples, the UFOC finds an optimal number that is not reasonable, especially when looking at the number of elements in the redundant clusters:



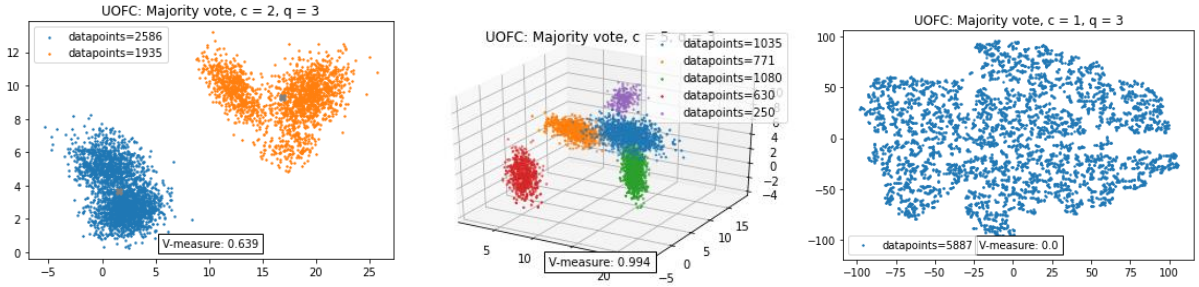


As we can see, the cluster validity criteria, especially in the first example, achieved high scores for the correct number of clusters, but for unclear reasons, they preserved high scores even when the different clusters have a minimal number of elements. This might be due to the high dimensionality of the data, or the high variance generated data points that can be considered clusters.

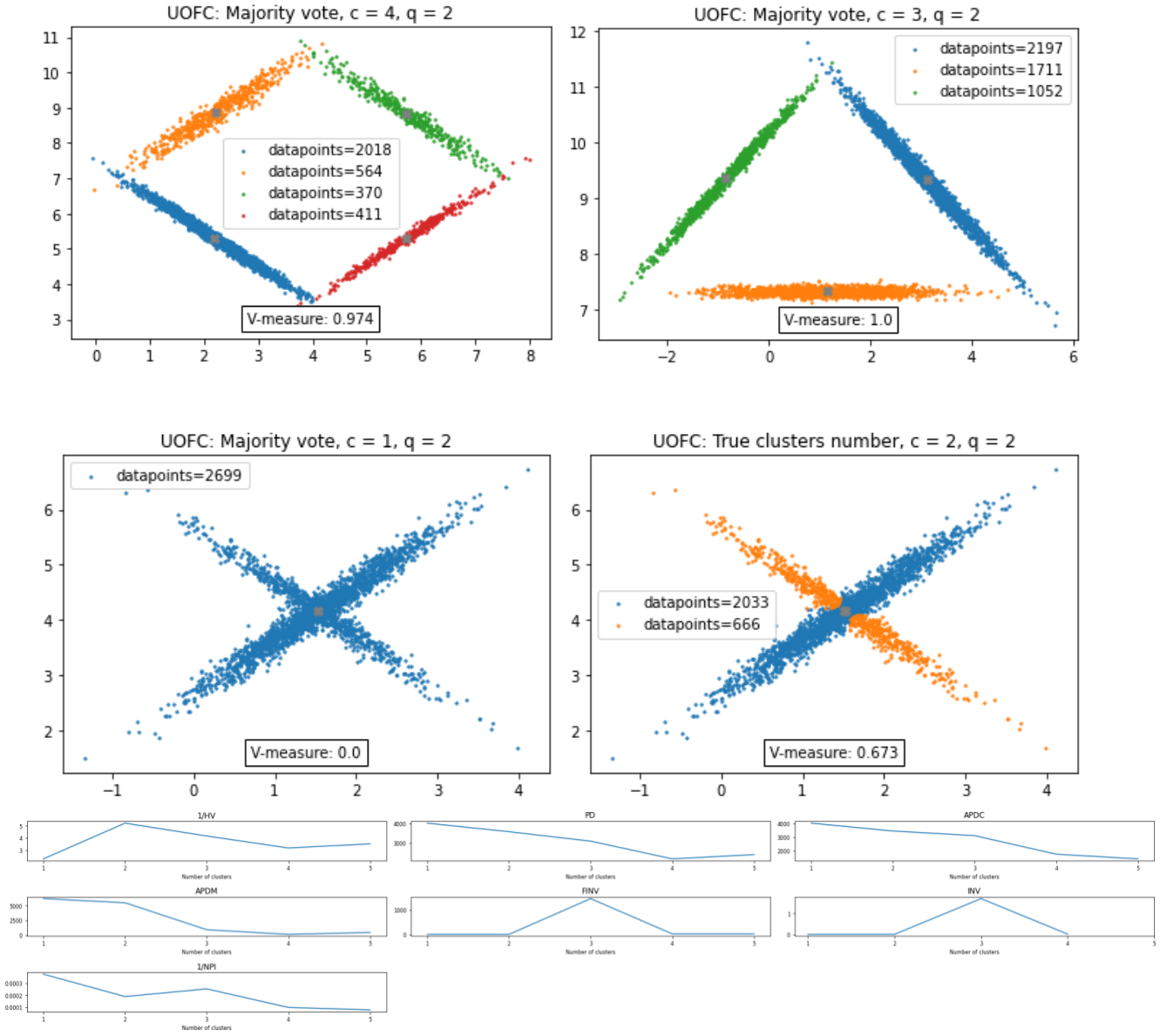
We tried to change the fuzziness parameter (q) and use fuzziness decay, such

$$q(t) = q_0 \alpha^t, \qquad a < 1$$

to find whether the UFOC criteria will be more precise; however, we found that it is not stable and can even worse the performance in some cases:



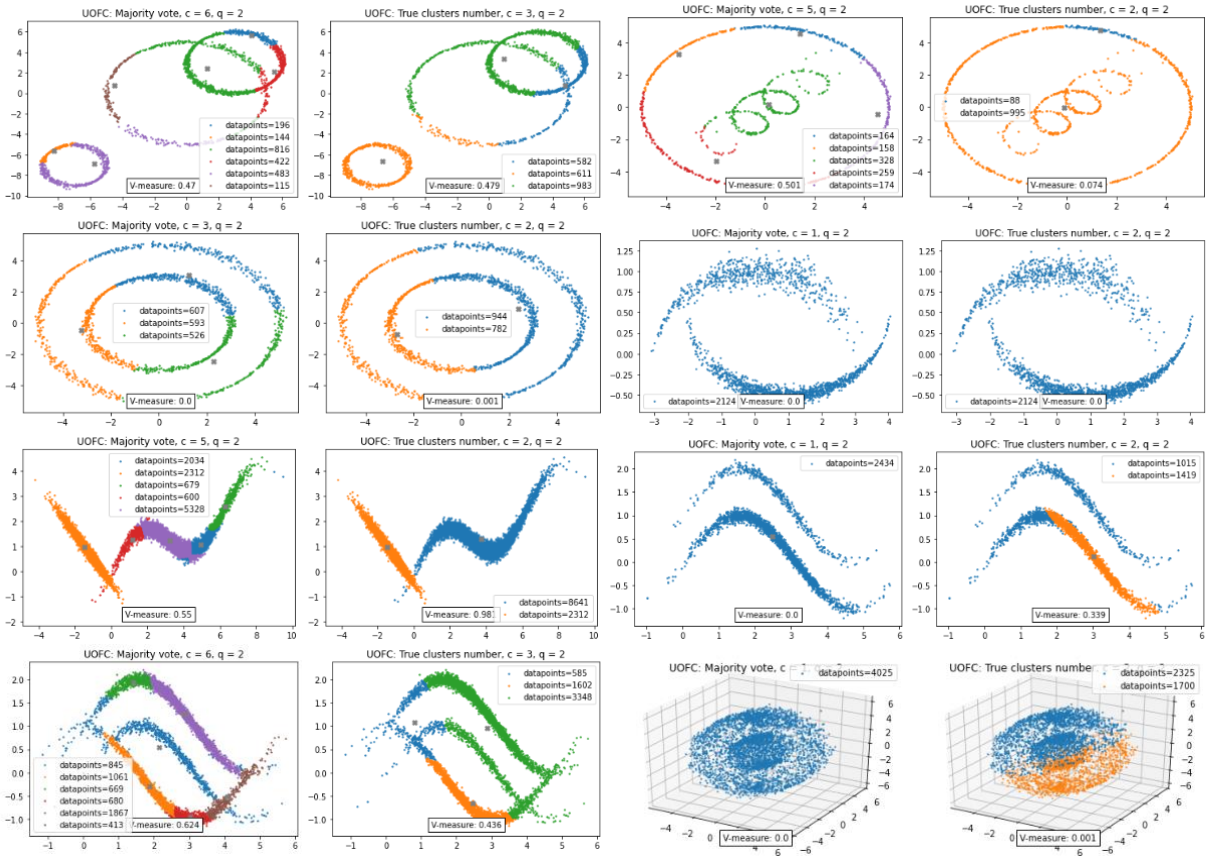
2. **<u>Dataset type 2 - Data generated using craft multivariate normal distributions:</u>**
As expected, in this dataset, since most of the patterns are straightforward, the UOFC algorithm succeeded well with the expectation of the cross pattern where most of the criteria found the maximal value with only one cluster:



Interestingly, the hypervolume criterion was precise in this and all previous examples.

3. **Dataset type 3 - generated using parameterization and multivariate normal distributions noise:**

We saw in Task 1 that the MLE failed to cluster the pattern we generated in those datasets. Since the UOFC is like MLE, we can expect it to fail on this dataset. Indeed, as we can see below, the UOFC failed to predict the number of clusters in each example. However, it is interesting to see the cluster that it created. Notice that in most of the cases, the UOFC found $Cmax$ to be the optimal number of clusters, and we can assume that if we take large $Cmax$, then it would have found it suitable as well due to the nature of those shapes.

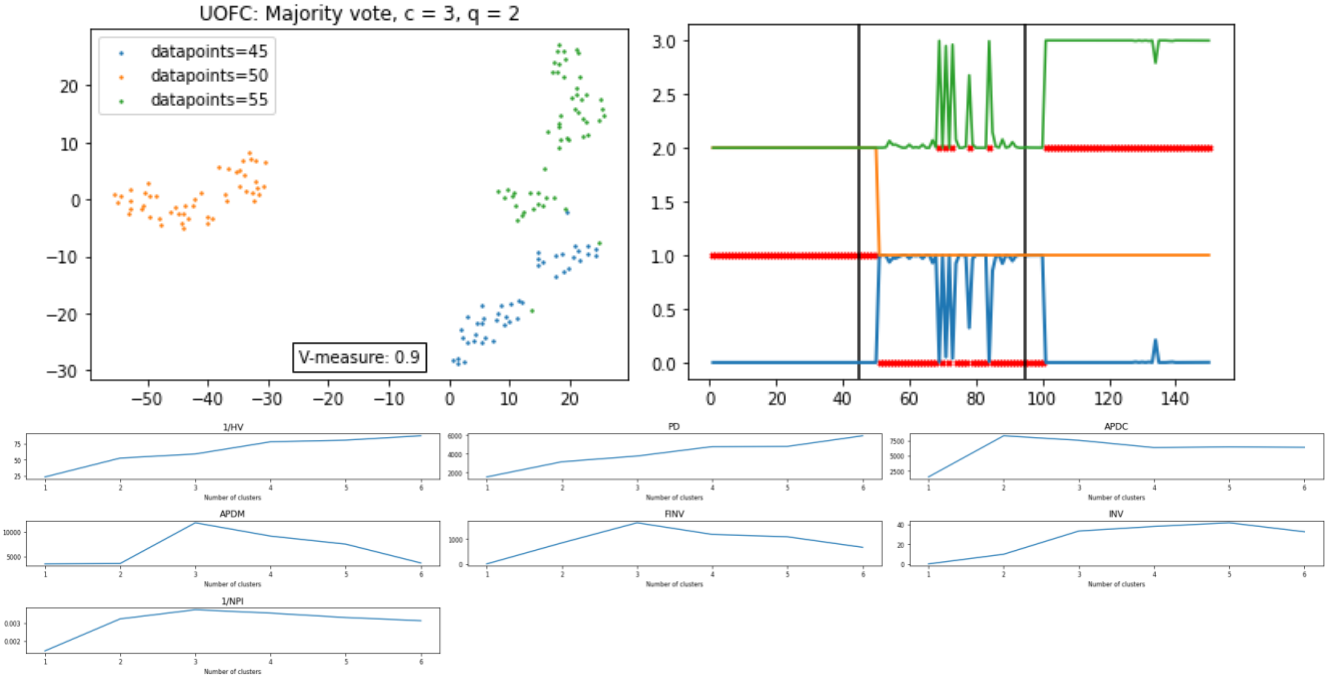


An additional point we see here again is that, in most cases, the MLE and UOFC provide the same performance for the actual number of clusters.

4. Dataset type 4 - Real datasets:

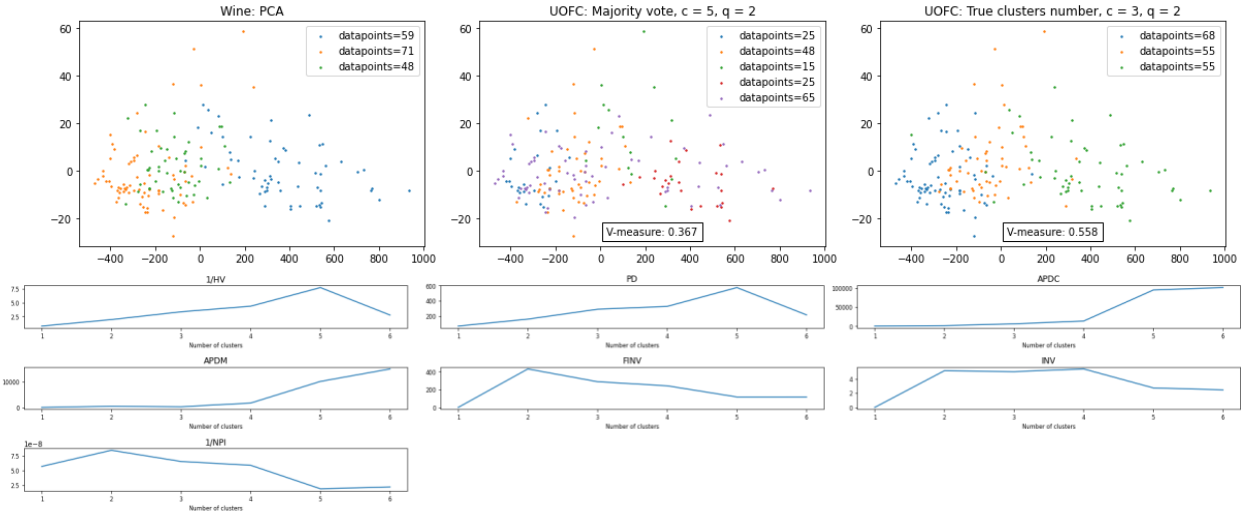
Iris dataset:

In this dataset, most criteria had the maximal value for 3 clusters, while the hypervolume criterion did not and increased as we took large clusters. The UOFC, like the MLE, succeeds in classifying all the flowers except for 5 correctly:



Wine dataset:

In the last dataset, most criteria had the maximal value for 5 or 6 clusters, while the actual number is 3. In terms of V-measure, even with the correct number of clusters, the UFOC obtained 0.558, which is lower than the MLE, which achieved 0.582:



4. Task 3 – Agglomerative Hierarchical Clustering

hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

- Agglomerative: This is a "bottom-up" approach - each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

- Divisive: This is a "top-down" approach - all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

In general, the merges and splits are determined greedily. The results of hierarchical clustering are usually presented in a dendrogram. In this task, we implemented the following agglomerative algorithm:

```
1 begin initialize c, ĉ ← n, D_i ← {x_i}, i = 1, ..., n
2         do ĉ ← ĉ − 1
3             Find nearest clusters, say, D_i and D_j
4             Merge D_i and D_j
5         until c = ĉ
6   return c clusters
7 end
```

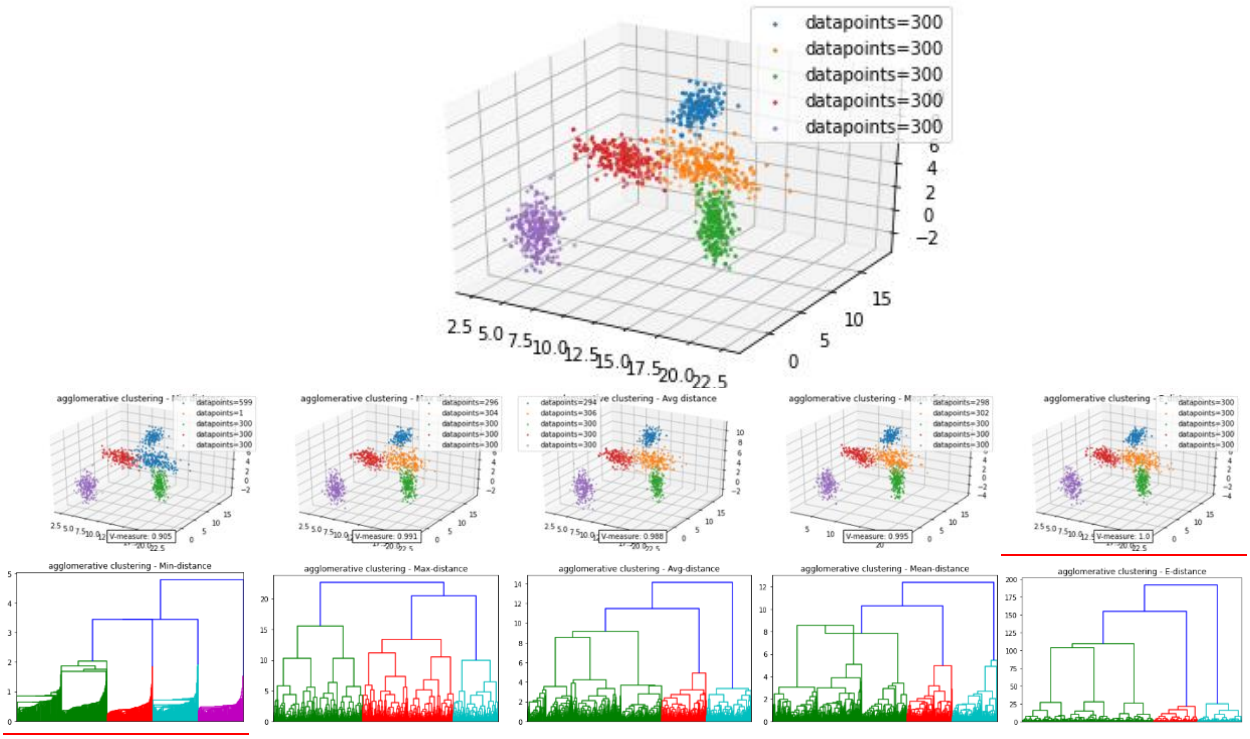
where we choose the nearest clusters according to 5 different dissimilarity distances:

1. Minimum: $\min_{x \in D_i, x' \in D_j} \|x - x'\|$
2. Maximum: $\max_{x \in D_i, x' \in D_j} \|x - x'\|$
3. Average: $\frac{1}{n_i n_j} \Sigma_{x \in D_i} \Sigma_{x' \in D_j} \|x - x'\|$
4. Mean: $\|m_i - m_j\|$
5. E: $\sqrt{\frac{n_i n_j}{n_i + n_j}} \|m_i - m_j\|$, a merger based on this distance will increase the sum-of-squared error as little as possible.

While the algorithm is intuitive, its time and memory complexities are high, making it implacable for large datasets. For this reason, and due to the many options, we randomly resized the clusters into a size of 300. In addition, we presented the clustering using a dendrogram. Here are the results we achieved on the different datasets:

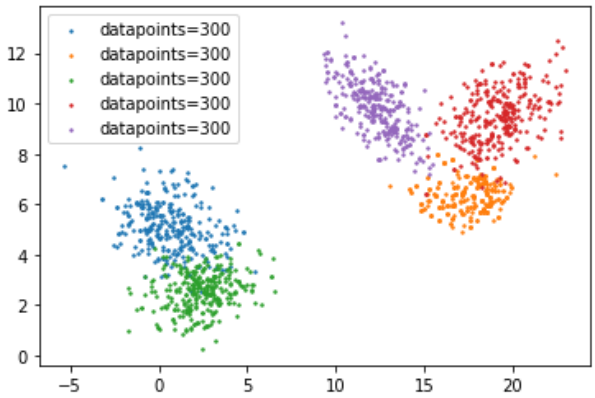
1. **Dataset type 1 - Data generated using random multivariate normal distributions:**

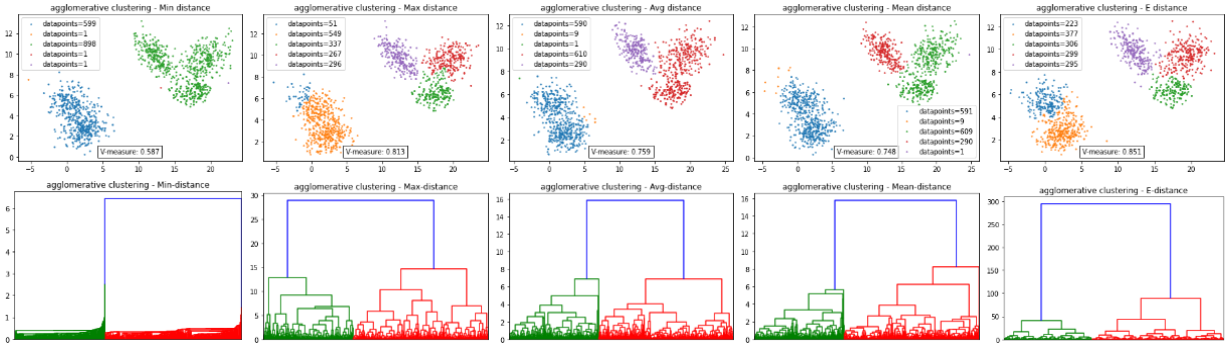
   We tested the agglomerative hierarchical clustering on the different datasets of this type. Since there are many datasets, we will show some interesting examples. Generally, we could notice that the algorithm performs well when the cluster points are not intersecting. For example, in the following example, except for the minimum distance, all the options worked very well, and the E-distance even resulted in a perfect score. Even the MLE algorithm that assumes normality of the data did not achieve this performance:



It is interesting to see the difference between the dendrogram structures. The minimum distance, for example, forms like a tail shape, and the E-distance form a more stable shape since it also considers the cluster sizes. The E-distance tends to favor growth by merging singletons or small clusters with large clusters over medium-sized clusters.

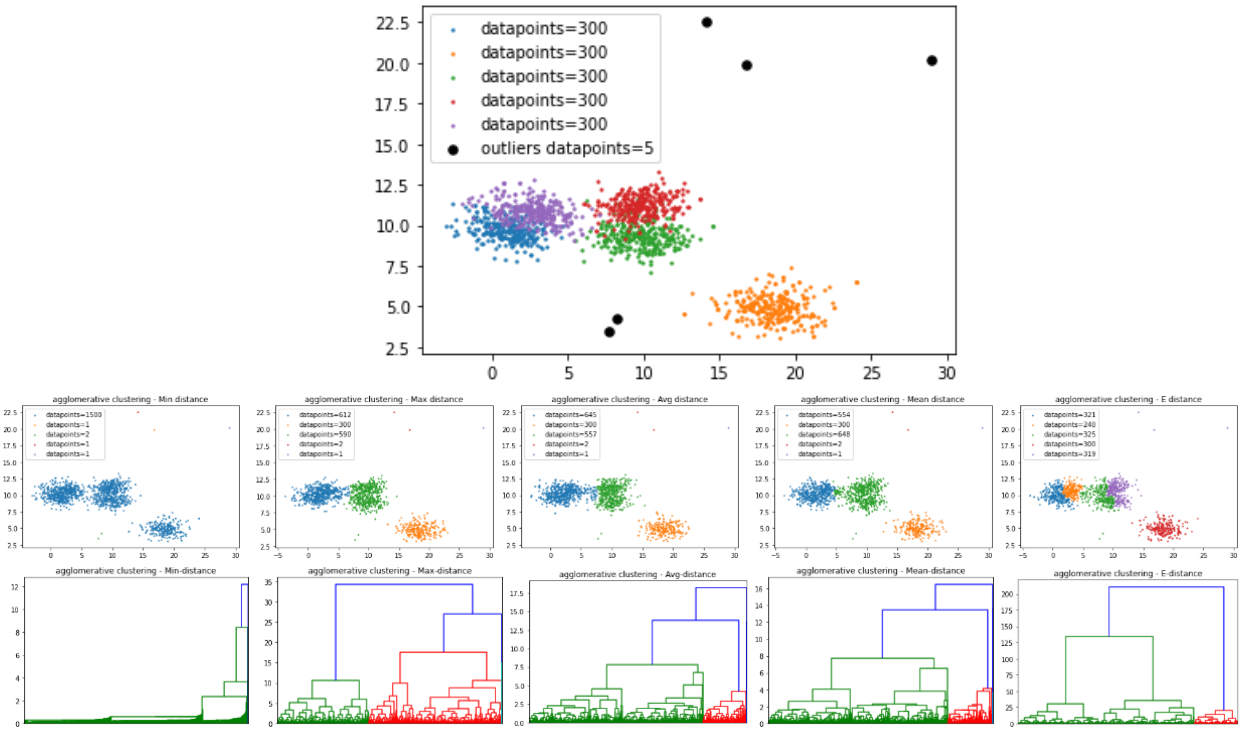
Here is an example where most distance criteria fail as some of the clusters might be considered single clusters (especially in the minimum distance that is very sensitive to cluster intersction. Again, we can see that the E-distance, thanks to its ability to consider the cluster sizes and distance between the means (the cluster centers), can distinguish even between the cluster and result in a similar V-measure to the MLE model, which achieved a score of 0.88:





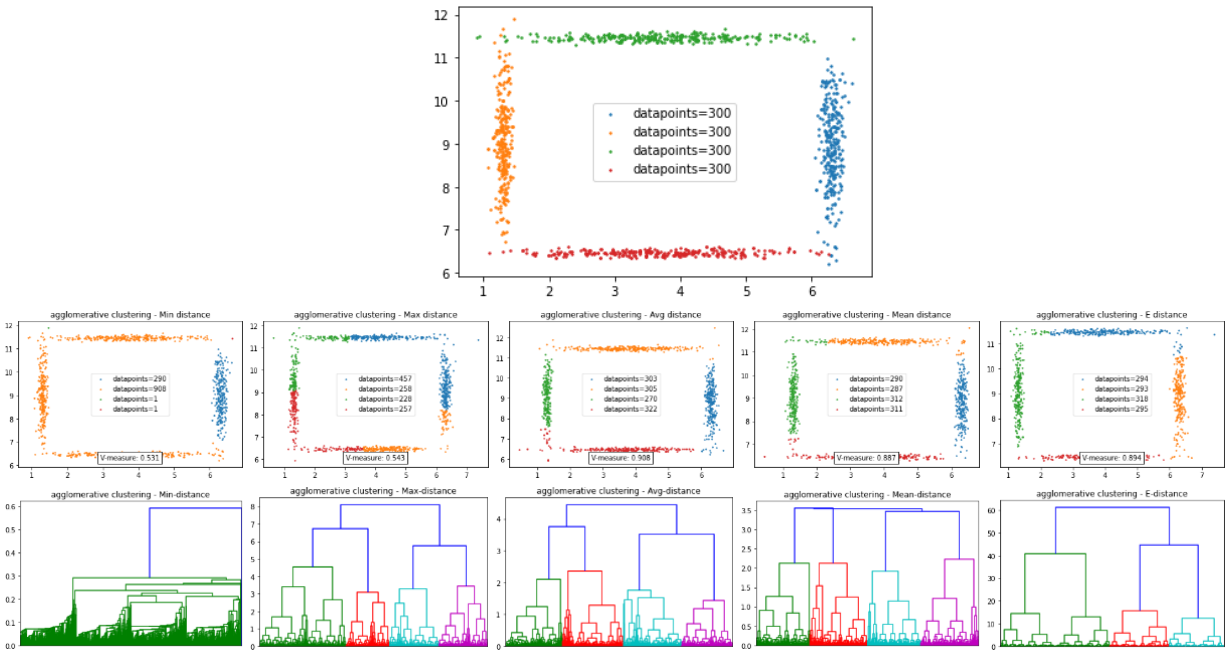
Again, it is exciting to see the dendrograms where similar clusters are merged first.

An additional example that we show here is the dataset with outliers. As expected, the minimum distance cannot overcome the outliers and is clustering the outliers as individual clusters. Other distances that do not consider the cluster size managed to overcome some outliers, but still, they could not identify the clusters correctly. The only distance that seems to manage outlier entirely is the E-distance. Nevertheless, its performance is still not optimal:

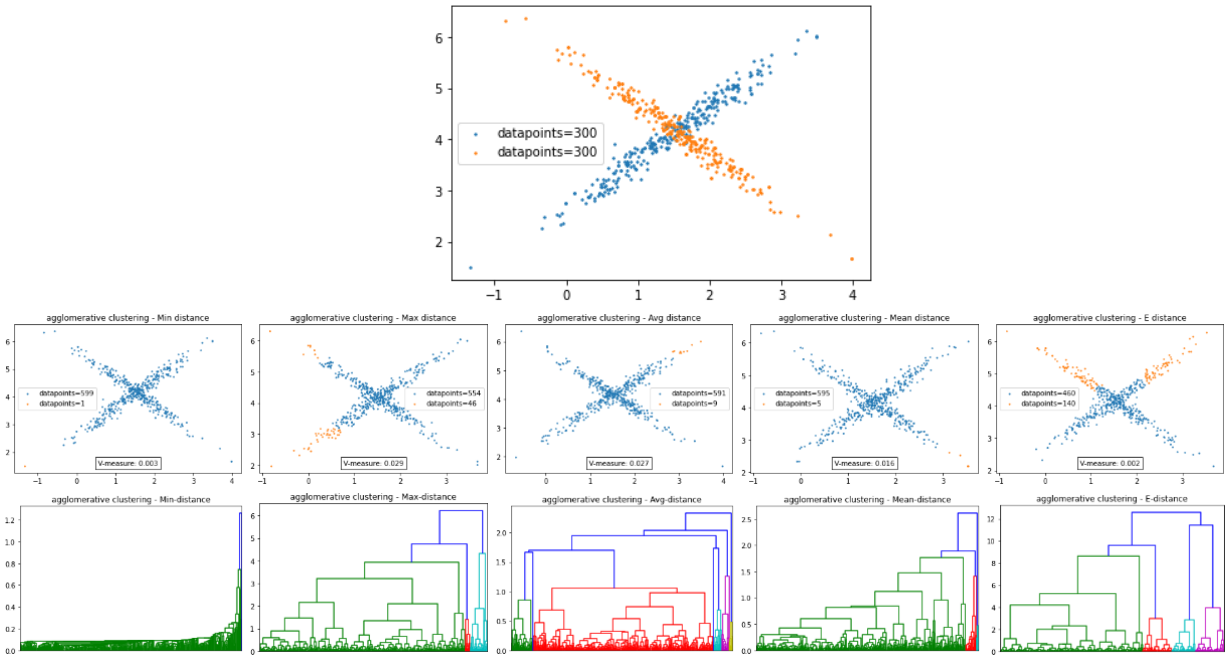


2. Dataset type 2 - Data generated using craft multivariate normal distributions:

This dataset is like type 1; therefore, when there is an intersection of clusters' points, the algorithm fails to cluster the points correctly. In the following example, we can see again that distances based on the average distance or the cluster mean work better when there is a such intersection:

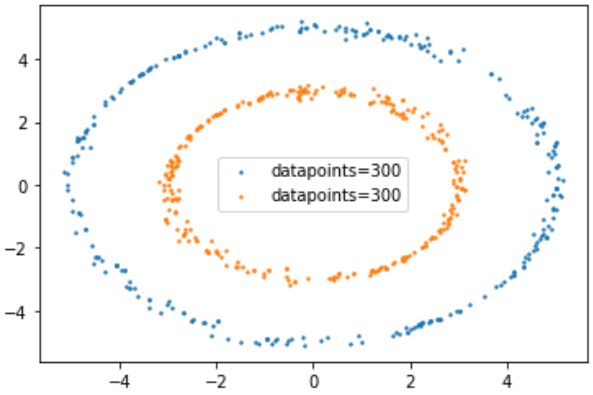


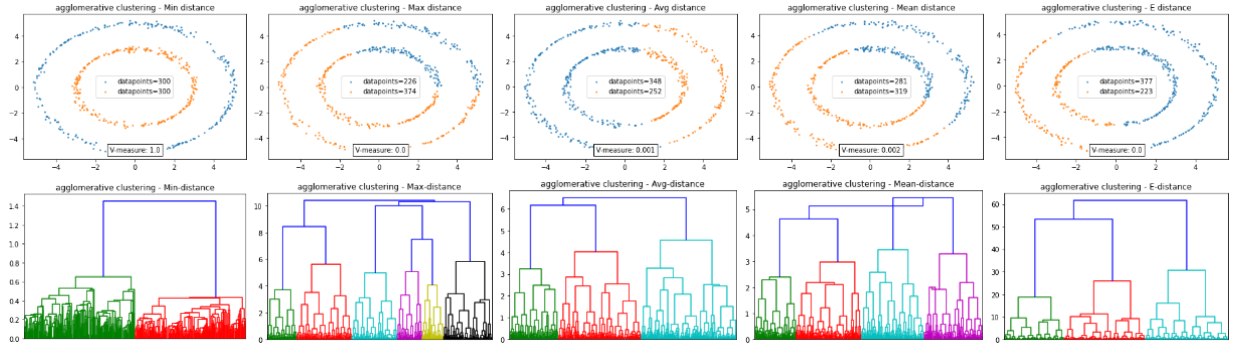
However, in the following cross pattern, where the cluster point intersect, and the cluster means are similar, all distances failed to achieve a good score.

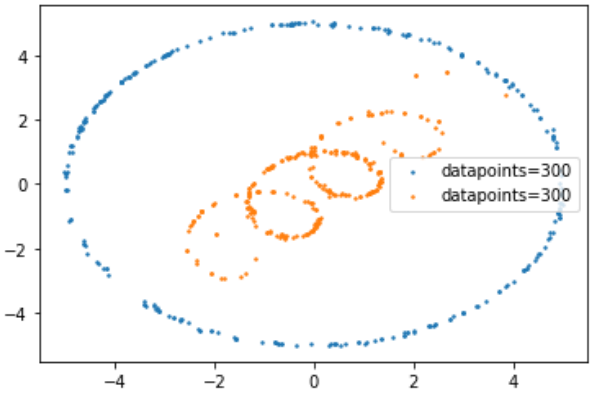


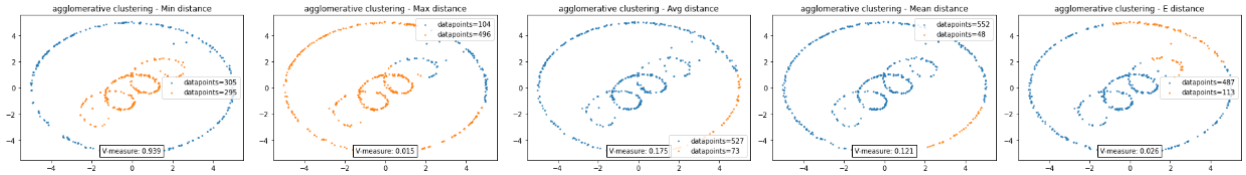
3. **Dataset type 3 - generated using parameterization and multivariate normal distributions noise:**

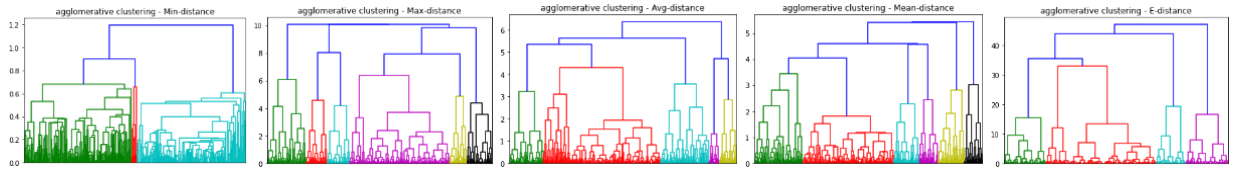
As we expected, the minimum distance is shining in this dataset type, as it is suitable for clustering chain patterns. We will show some examples:





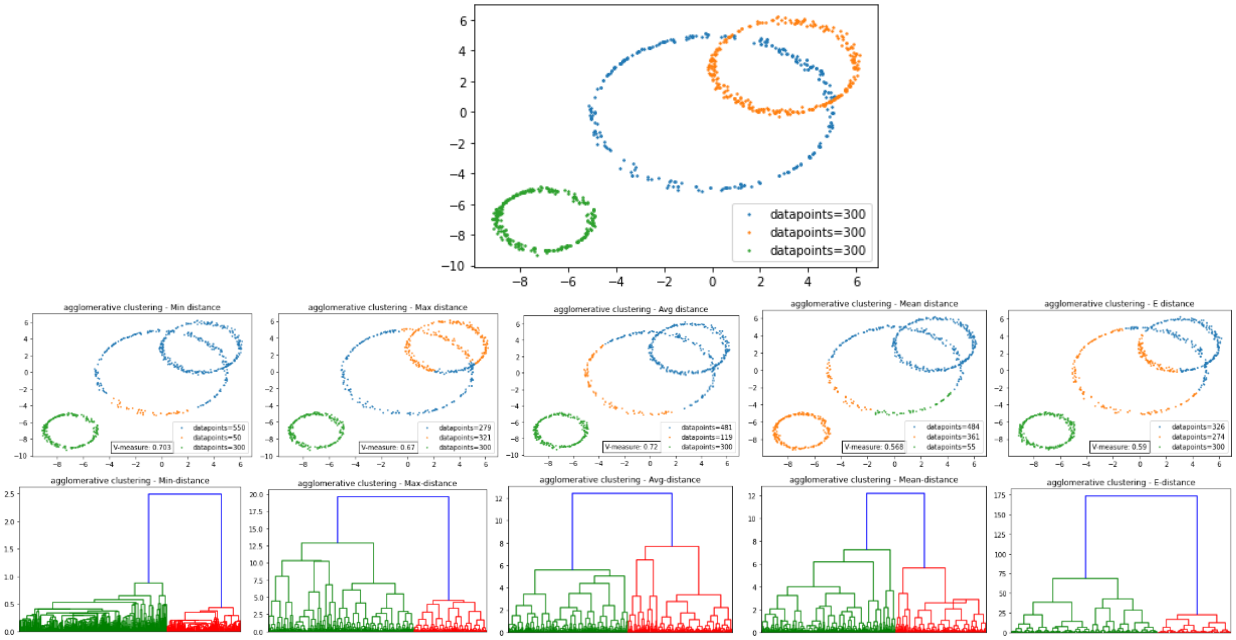




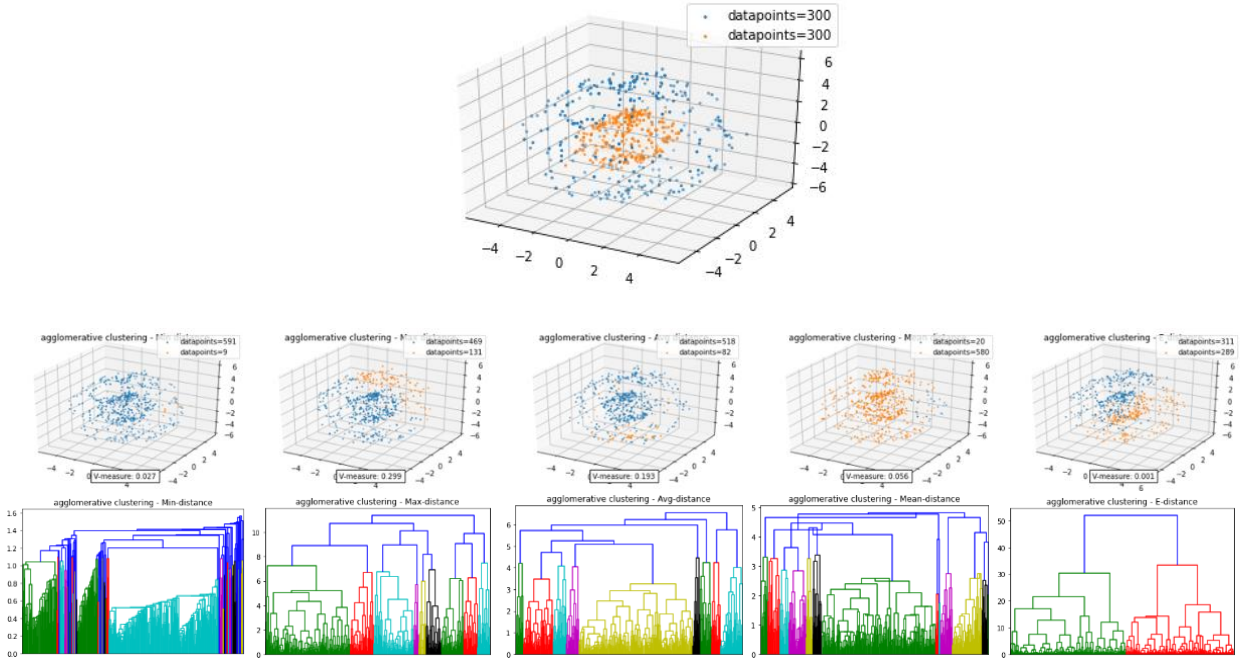


We resulted in similar results with the moon patterns.

While most of the patterns of this type were clustered perfectly with the minimum distance, when there is an intersection of points or noise in the dataset, the algorithm fails to cluster correctly. Here we can see the circles' patterns that intersect:



Here we can see the sphere shapes that share the exact origin (same centroids) that failed even with the minimum distance. This might be explained by the noise that where added:

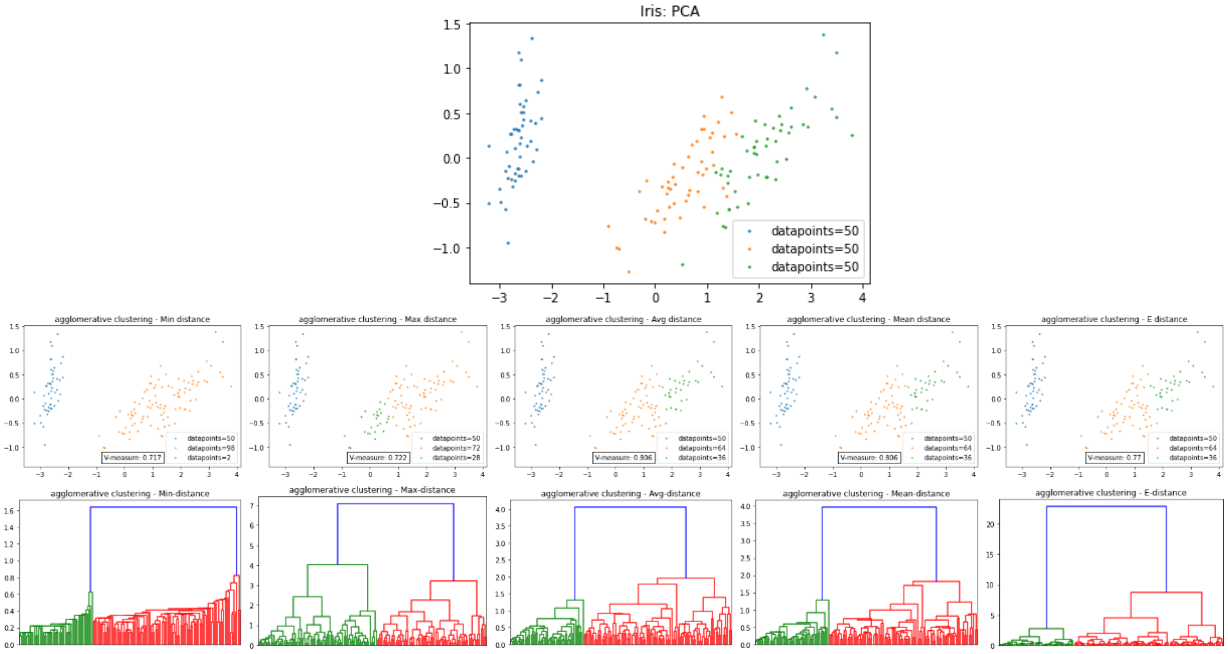


We obtained similar results with the sinusoidal patterns that were close to each other

4. Dataset type 4 - Real datasets:

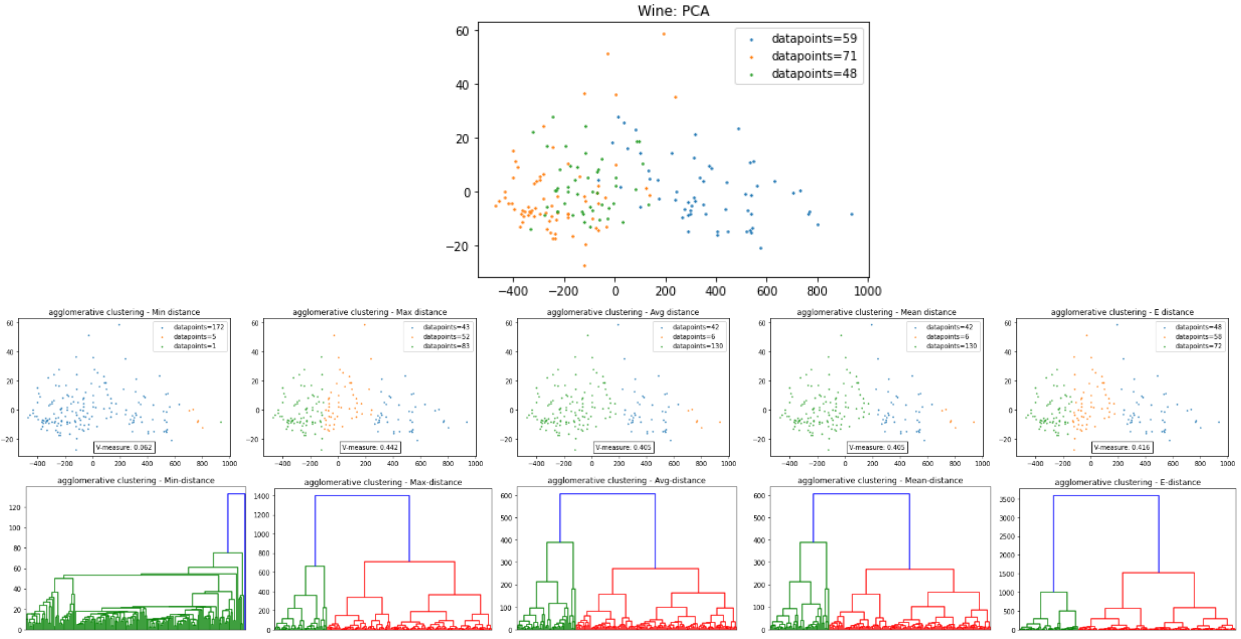
Iris dataset:

In the Iris dataset, all distances succeed in clustering the well-split cluster, however the performance over the more complicated clusters where reduced to the MLE algorithm:



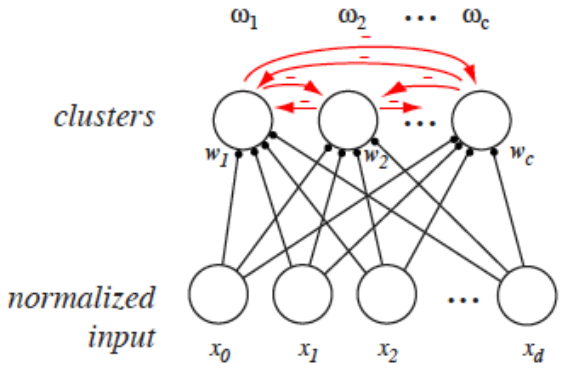
Wine dataset:

In this dataset, we know that the clusters are very close; therefore, as expected, the minimum distance obtained a very low score. The other distances failed compared to the UOFC and the MLE.



5. Task 4 – Competitive Learning

Competitive learning is an algorithm based on a neural network (see figure below) typically used to cluster high-dimensional data. This categorization occurs in the output layer, where neurons compete to learn unique data types represented within a large input dataset. Only one output neuron is activated for a given input, adopting a "winner-takes-all" strategy which is the key principle of competitive learning.



This simple network design consists of an input and output layer where the input layer is connected to the output layer via a series of weight vectors associated with each output neuron. We implemented the model according to the algorithm steps:

begin initialize $\eta, n, c, \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_c$
  $\mathbf{x}_i \leftarrow \{1, \mathbf{x}_i\}$ $i = 1, \ldots n$ augment all patterns
  $\mathbf{x}_i \leftarrow \mathbf{x}_i / \|\mathbf{x}_i\|$ $i = 1, \ldots n$ normalize all patterns
  do randomly select a pattern $\mathbf{x}$
    $j \leftarrow \arg\max_{j'} \mathbf{w}_{j'}^t \mathbf{x}$ classify $\mathbf{x}$
    $\mathbf{w}_j \leftarrow \mathbf{w}_j + \eta \mathbf{x}$ weight update or $W_j \leftarrow W_j + \eta(X - W_j)$
    $\mathbf{w}_j \leftarrow \mathbf{w}_j / \|\mathbf{w}_j\|$ weight normalization
  until no significant change in $\mathbf{w}$ in n attempts
return $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_c$
end

After testing different options, we chose the learning rate to be $\eta = 0.1$, and used learning rate decay:

$$\eta(t) = \eta_0 \alpha^t, \qquad a = 0.99$$

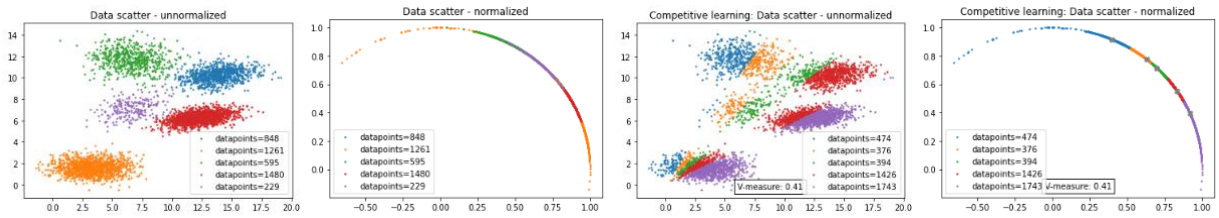
We stooped the iterations if the weight norm did not change for 100 iterations ($\epsilon = 10^{-5}$).

One of the main problems we found with this algorithm is finding the cluster centroids of the data projected onto the unit ball space (or circle in 2D) due to the L2-normalization. The weights should converge to the mean of different cluster data. In this case, when we normalize the data, some of the clusters overlap, making it hard for the algorithm to distinguish between the different clusters. We found that most of the data we generated suffered from this issue, and therefore we will show some examples where the algorithm

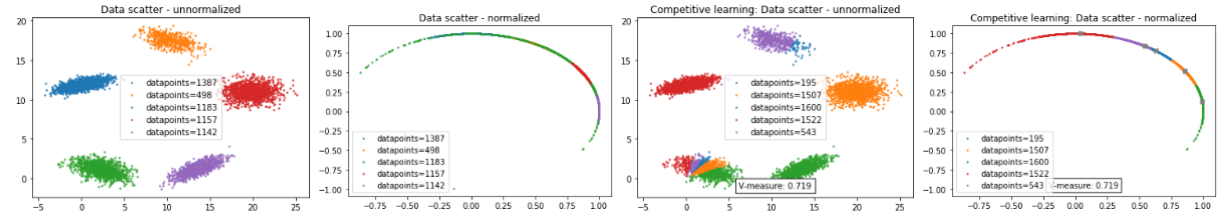
worked and others where it did not. In addition, we plot the data scatter plot before and after the normalization to show the model's difficulty in handling this data.

Examples where the model did not perform well (in most cases):

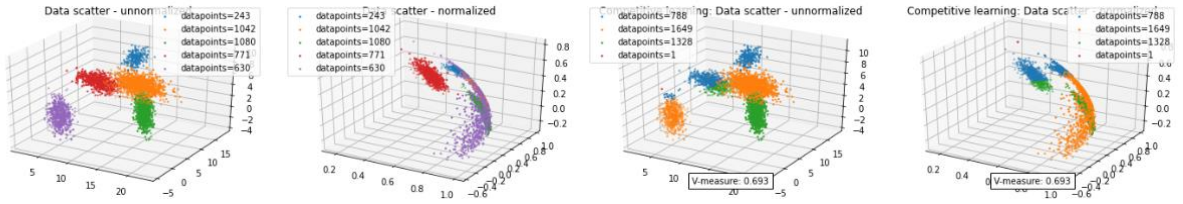
Dataset type 1:



The normalized data overlap, as seen in the normalized scatter plot. We saw that the chances of this issue occurring in 2-dimensional data are higher than in higher dimensions. For example, even in the following dataset, where the clusters are well spread, the algorithm failed to cluster correctly:

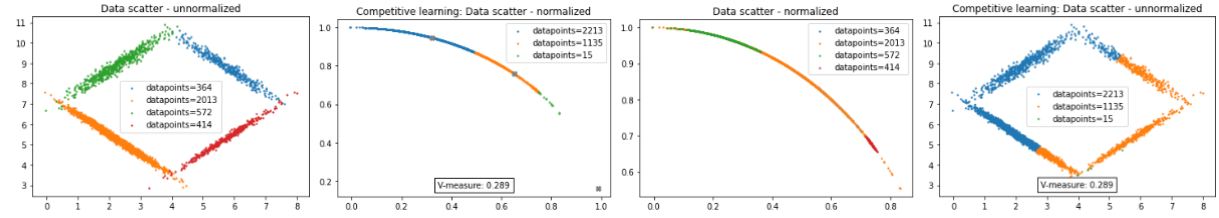


Here is an example in 3-dimensional data:

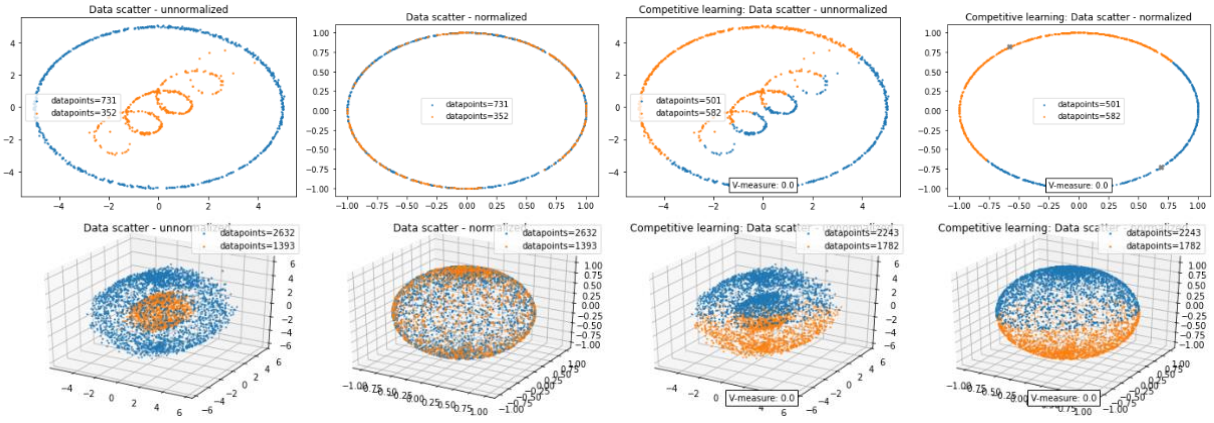


The algorithm also missed in most of the datasets of types 2, 3, and 4:

Dataset type 2:

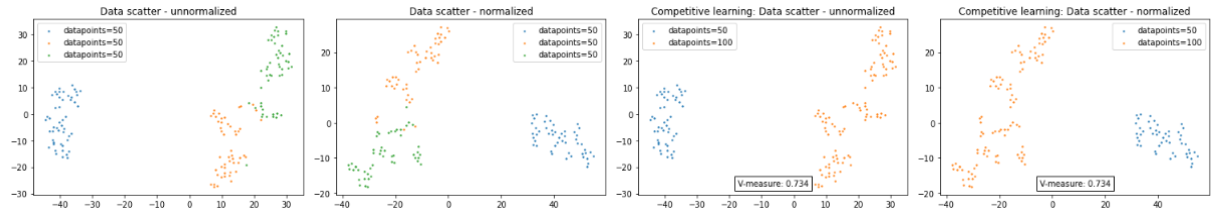


Dataset type 3:

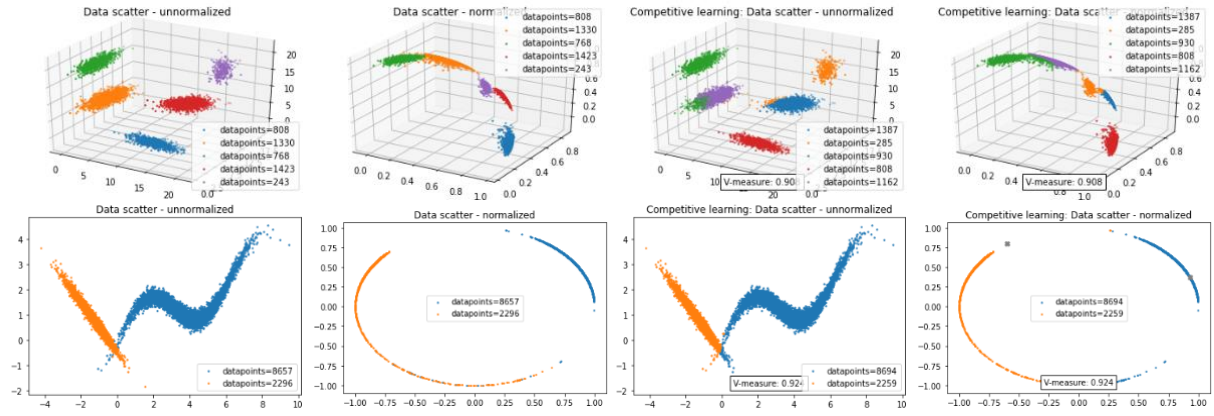


Dataset type 4 (Iris):

Notice that here the scatter plot after normalization is similar as is it the T-SNE visualization



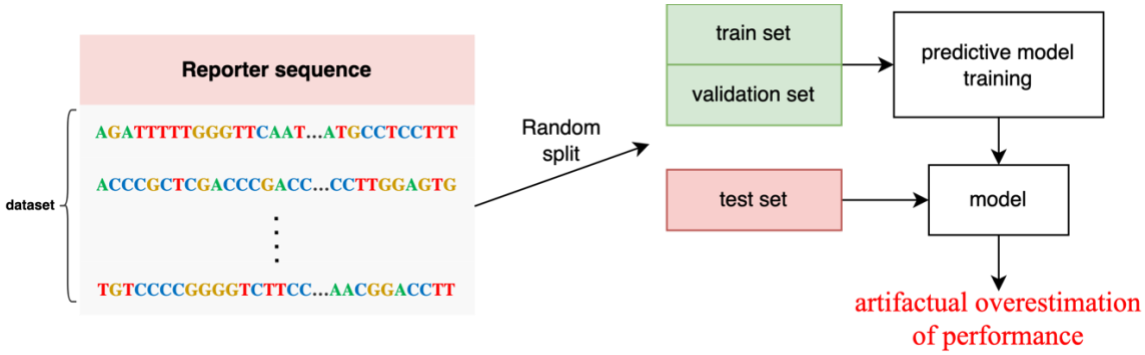
Examples where the model did perform well (only a few):



As we can see, most of the normalized points did not overlap, so the model clusters better than in the previous examples.

6. task 5 – Seminar Work – Community Detection

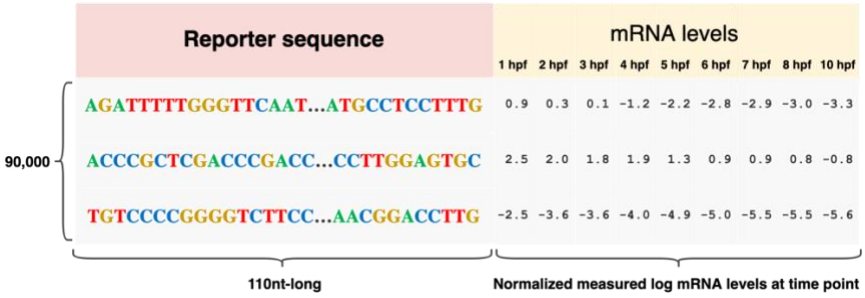
Computational methods are typically benchmarked on test data that is separate from the data used to train the method. In many areas of machine learning and statistical inference, data samples can be thought of as approximately independent samples from some unknown distribution describing the data. In this case, a standard approach is to randomly split available data into a training and a test set, fit a model to the training set, and evaluate the model on the test set. In computational biology, families of biological sequences are not independent because they are evolution related. Random splitting typically results in test sequences that are closely related or identical to the training sequence, leading to an artifactual overestimation of performance. The problem becomes more concerning for complex models capable of memorizing their training inputs, such as deep neural networks. This issue motivates strategies considering sequence similarity and splitting data into dissimilar training and test sets.



Test case:

Dataset:

We used as a test case a data set called UTR-seq. the UTR-seq is a massively parallel reporter assay to measure RNA degradation. The report library contains 90,000 designed 110-nt long sequences. The mRNA reporter libraries were injected into one-cell zebrafish embryos, and the mRNA levels were measured at nine-time points: 1- to 8-h post-fertilization (hpf) and 10 hpf. Therefore, the dataset contains 90,000 sequences, and for each sequence, nine measured mRNA levels.

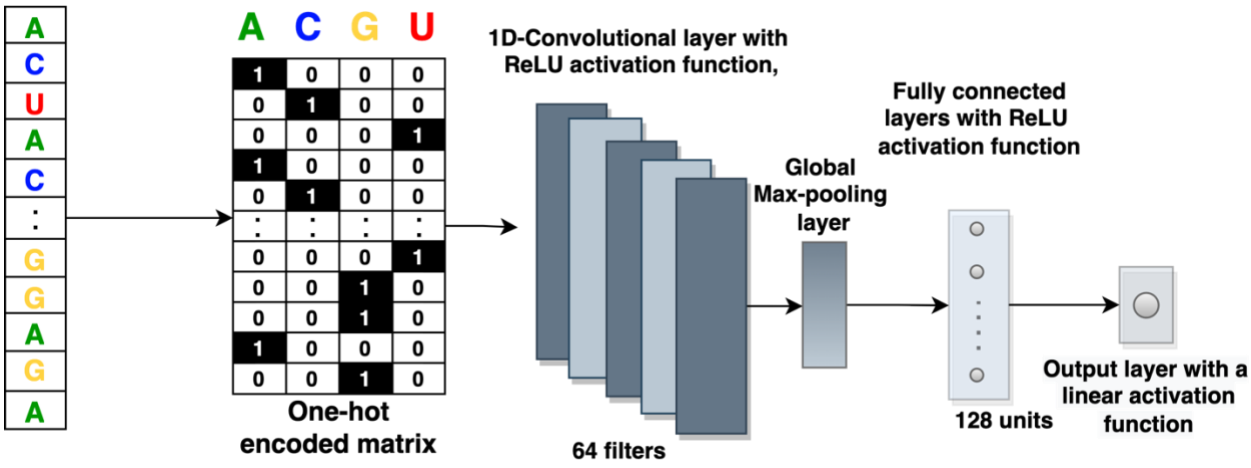


We aim to predict mRNA degradation dynamics as a function of 3′-UTR elements (the mRNA sequence).

To simplify the model generation, we formed a model that, given a sequence, predicts the mRNA degradation rate. We fitted a linear regression model to each reporter and used the slope as the mRNA degradation rate label.

Predictive model:

The predictive model we chose as a test case is a simple neural network based on 1-dimensional convolutional (CNN) as they are known to perform well in extracting essential sequence elements. We choose a simple architecture to avoid adding extra complexity to our test case. The model architecture is described in the following figure:



We trained the model with the mean squared error for 10 epochs. The optimizer that we used is Adam with a batch size of 64.

We evaluate the model's performance by computing the Pearson correlation of the predicted and actual mRNA degradation rates over a test set we defined.

Generation of a nonhomologous held-out test set:

The *UTR-seq* dataset sequences were generated with multiple overlapping sequence fragments from annotated 3′-UTRs. In a random partition of this set, there is a risk of having identical sub-sequences in the training and test sets, which may lead to over-estimation of prediction performance, as we explained above. Therefore, our goal is to partition the dataset into nonhomologous train and test sets.

We assume that if the partition is good, then the model's prediction performance on the test set will be reduced to performance using a random partition. The problem with this evaluation is that we cannot evaluate the performance over the same test set. To reduce the impact of this issue, we decided to perform 10 experiments for each partition approach. This way, for each approach, we will have 10 different random test sets based on the approach. The score of a partition approach would be the average of the experiments. In addition, since neural model training (especially on GPUs) is not a deterministic process, then in an experiment, we performed 10 model training and averaged the Pearson correlation scores.

Partition based on connectivity in a graph:

Before explaining the approach, let us define the term k-mer. Given an mRNA sequence of length l, which we can define as s $\in \Sigma^l$ where $\Sigma = \{A, C, G, T\}$. We define a k-mer as a set of all subsequences of length $k \in \mathbb{N}$.

In this approach, we aimed at having no two sequences from two parts contain the same k-mer for some $k \in \mathbb{N}$. Given N l-long sequences $\{s_i | 1 \leq i \leq N\}$, we define the k-mer overlapping indicator function, $I_k : (\Sigma^l \times \Sigma^l) \to \{0,1\}$, as

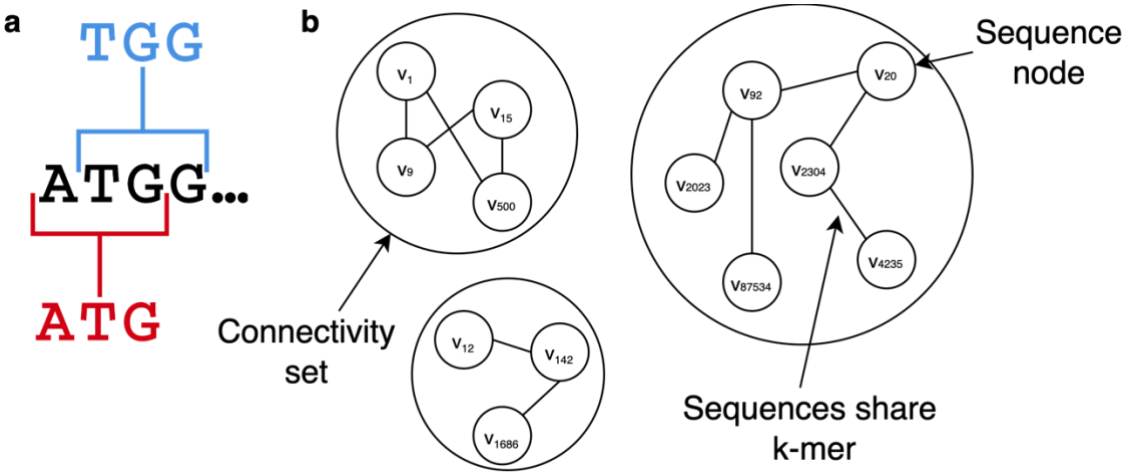
$$I_k(s_i, s_j) = \begin{cases} 1, & \exists 1 \leq m_1, m_2 \leq l - k \text{ such that } s_1[m_1 : m_1 + k - 1] = s_1[m_1 : m_1 + k - 1] \\ 0, & else \end{cases}$$

where $s_i[m : m + k - 1]$ is the k-mer in si starting at position m. Using the k-mer overlapping indicator function, we define the k-mer overlapping graph, $G_k = (V, E)$:

$$V = \{v_i \mid 1 \leq i \leq N, \text{where } v_i \text{ is the vertex corresponding to sequence } s_i\}$$

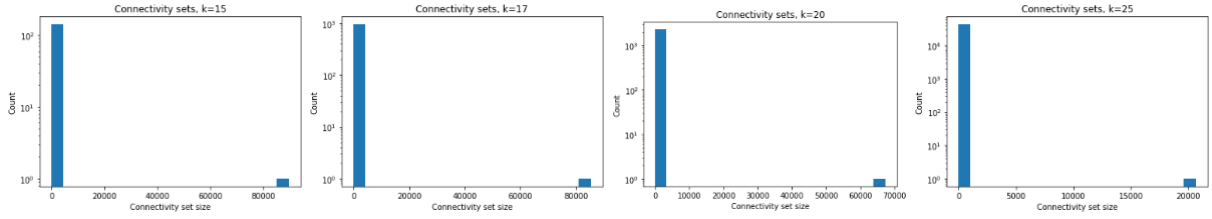
$$E = \{(v_i, v_j) \mid I_k(s_i, s_j) = 1\}$$

The k-mer disjoint sets are obtained by finding the connectivity sets in the graph. Following this process, our training and test sets did not contain any shared k-mer of the desired length.

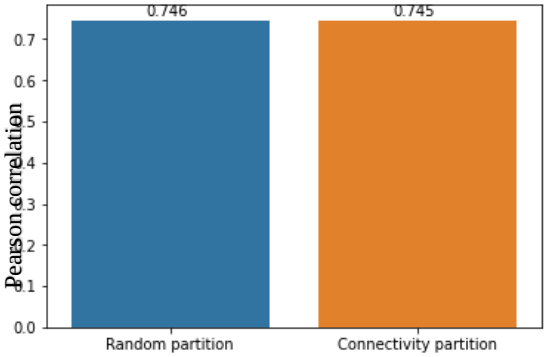


When examining this approach, we found that it suffers from two main problems:

1. When choosing small k, we ended up with one connectivity set that would not allow us to split the data into 80% of the train set and 20% of the test set. For this reason, we chose $k = 20$. Choosing larger values for k may allow a considerable homology between the sets, and therefore we should avoid this. When choosing $k = 20$, the graph that we constructed contains 524,752 edges. Note that when choosing $k = 20$, we still have one giant set, which will always be part of the train set. This lack of variability in choosing the sets is another drawback of this approach.



2. When we examined the model's performance, we found that the random partition and the connectivity partition with $k = 20$ achieved almost the same Pearson correlation score. This might indicate that our connectivity partition preserved the homology between the sequence set.



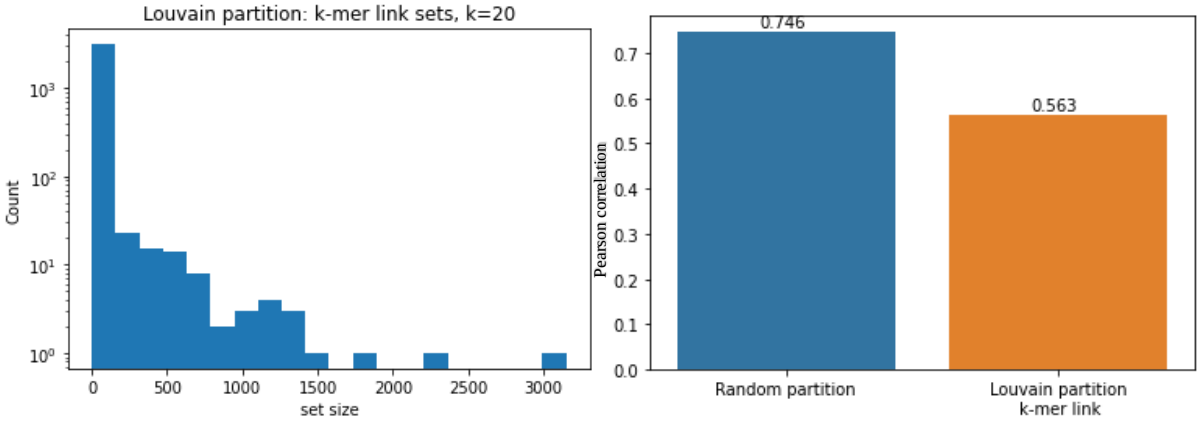
<u>Partition based on community detection:</u>

By defining the k-mer homology using a graph, aiming to find a partition of the graph (i.e., find communities), we could use the Louvain algorithm for community detection. The advantage of this this algorithm is that we do not need to know the number of communities in advance, and this is indeed the case in our problem. In addition, since the algorithm is based on the edges' weight, we can form different connections and weigh them differently. Therefore, we suggested a different approach to forming the network on which we apply the Louvain algorithm.

A. <u>Single k-mer connection – single weight:</u>

We define the same graph in this approach as in the connectivity partition. Since the Louvain algorithm handles unweighted graphs as well (using the adjacency matrix), we can apply the algorithm to the graph without any modification. Note that if we had used other clustering algorithms unsuitable for the graph, we would have handled with $\binom{90{,}000}{2} = 4{,}049{,}955{,}000$ edges.

The results of this type of partition for $k = 20$ are shown in the following figures:



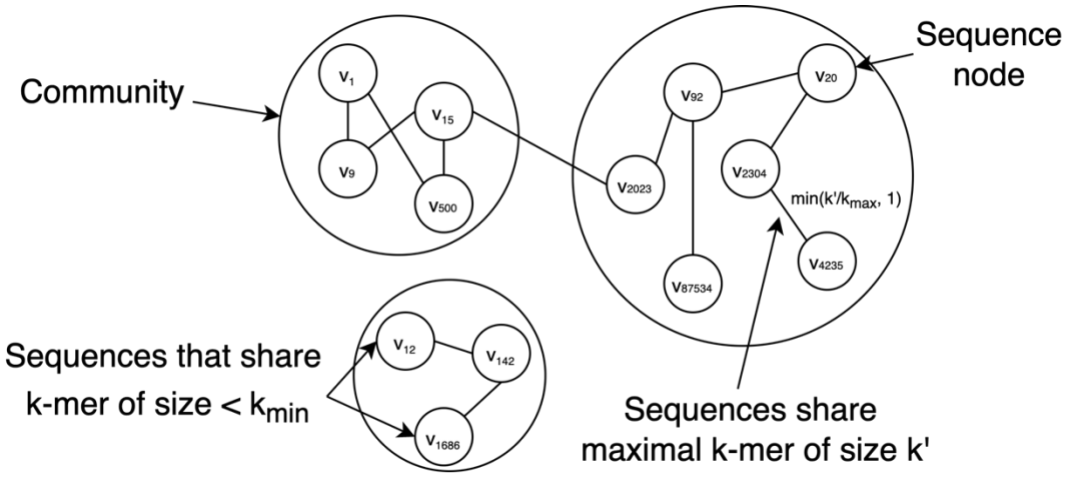
As we can see, we now have much more variability in choosing the train and test sets. More importantly, we can see that the performance over the test set was reduced significantly, and therefore, we can conclude that this approach could generate sets that share less information and are more likely independent.

B. Multiple k-mer connections – multiple weight:

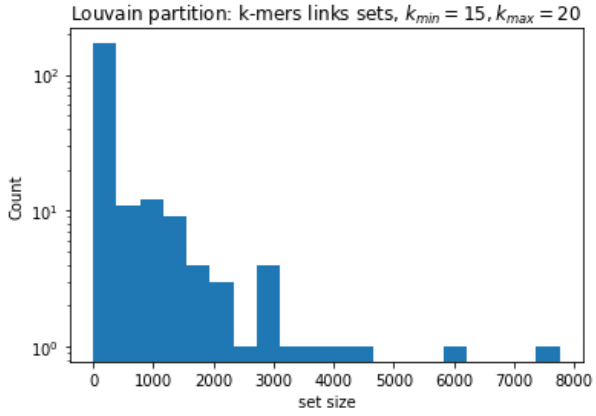
Since the Louvain algorithm support weights, we utilized this to improve the representation of our network. We added different weighting for different sizes of k-mer links. For example, we can choose k-mer sizes k_{min} and k_{max} ($k_{min} \leq k_{max} \leq l$) that define the weakest and the strongest link between two sequences in the following way:

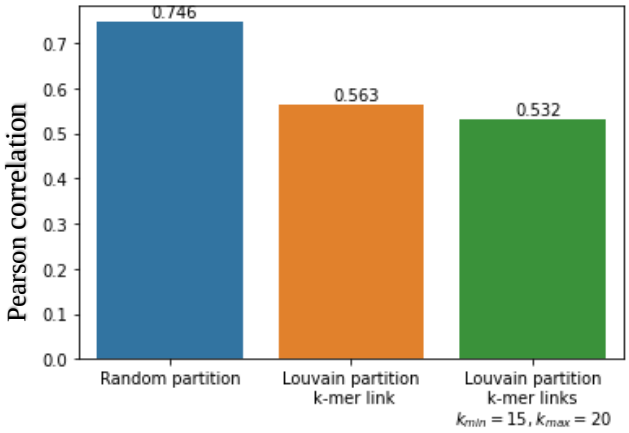
$$E = \{(v_i, v_j) \mid I_{k_{min}}(s_i, s_j) = 1\}$$

$$w((v_i, v_j)) = \frac{max_{k_{min} \leq k \leq k_{max}}(k \cdot I_k(v_i, v_j))}{k_{max}}$$



Given the formed network, we can apply the same Louvain to the network. We applied the algorithm for $k_{min} = 15$ and $k_{max} = 20$ which formed a network with 1,640,257 links. The results of this type of partition are shown in the following figures:





As we can notice, this formulation can increase the number of communities and even reduce the model's prediction performance on the test set.

C. <u>Sequence distance based Euclidian distance of k-mer counting frequencies:</u>

One of the problems with the previous approaches we suggested is that they define the sequence links according to only one connection between the sequences. For that reason, suggest an additional type of connection based on the k-mer counting frequencies:

For each sequence s_i and k, we can define $v_{i,k} \in \mathbb{N}^{4^k}$ as a frequencies vector. Each element in the vector contains the number of times a corresponding k-mer element occurs as a subsequence in s_i. Given $k' \in \mathbb{N}$, we can concatenate all $\{v_{i,k}\}_{k=1}^{k'}$ to

$$u_i = [v_{i,1} \quad v_{i,2} \quad \cdots \quad v_{i,k'}], \qquad u_i \in \mathbb{N}^S, \qquad S = \sum_{j=1}^{k'} 4^j$$

We this definition, we can form the counting frequencies matrix for all reporters as follows:

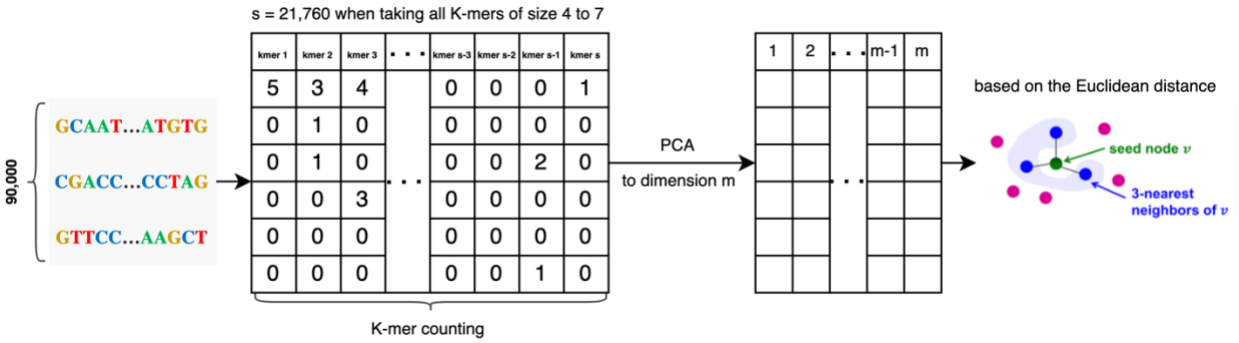
$$D = \begin{bmatrix} - \ u_1^T \ - \\ - \ u_2^T \ - \\ \vdots \\ - \ u_N^T \ - \end{bmatrix}, \quad D \in \mathbb{N}^{N \times S}$$

where each row represents a reporter.

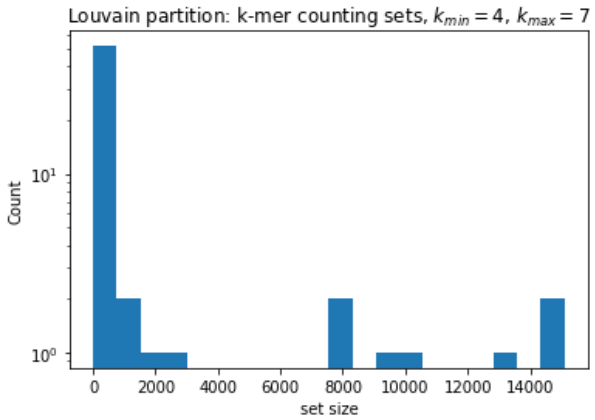
Now, when each sequence is defined using a vector u, we can compute the distance between sequences according to some metric. For simplicity, we choose the Euclidian distance; however, we can either choose any distance or similarity metric. Generally, when dimensionality increases, data becomes increasingly sparse, while density and distance between points, critical to clustering and outlier analysis, become less meaningful. In our case, the matrix we formed is spared by design, and by its formulation, it is also known that some columns are dependent. Therefore, we choose to use the PCA dimensionality reduction method to reduce the matrix diminution and, more importantly, to obtain the principal components of the data (in theory).

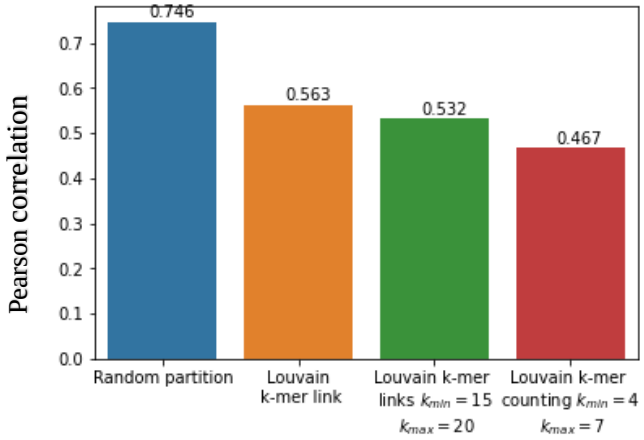
If we use the m singular vectors that correspond to m largest singular values, then the approximation of D would be a matrix $\widehat{D} \in \mathbb{R}^{N \times m}$. To form the graph based on the Euclidian distance, we still need to compute the distance between each of the sequences and represent this distance as an edge in the graph. As we explained above, this is not feasible as we will result in a graph with $\binom{90{,}000}{2} = 4{,}049{,}955{,}000$ edges. Therefore, we build a **k-nearest neighbor graph** based on the Euclidian distance. This way, we can produce a graph with the desired size. Once this graph is formed, we still cannot use the Louvain community detection algorithm since that it based on similarity and not distance. The convert the distance into similarity, we applied a simple transformation of the links' weights: if f $d(p_1, p_2)$ represents the Euclidean distance from a point p_1 to point p_2, then the similarity would be:

$$s(p_1, p_2) = \frac{1}{1 + d(p_1, p_2)}$$



In our case, we formed the counting frequencies matrix for all k-mers of size 4 to 7, resulting in a matrix of size $90{,}000 \times 21{,}760$. We reduce the matrix dimension to $m = 100$. After that, we formed the 10-nearest neighbor graph. This way, we ended up with a network with 733,261 weighted edges. The results of this type of partition are shown in the following figures:





We can see that with this approach, we achieve additional deterioration in the prediction performance while losing some variability in choosing the train and test set.