ArcaAnalytics

Binary Data Clustering

Iezzi Patrizio

Contents

Preface

The purpose of this study is to show new techniques of data transformation, clustering and clustering evaluation compared to what is already in use in the project INPS ArcaAnalytics.

The analysis is limited to enlighten strengths and weaknesses of methods without going into deep-down parameters tuning.
Be aware to take the study as it is without jumping into conclusions. Because, as mentioned above, a more rigorous approach can address to a completely different optimal.

The faced challenge is to clustering data (exclusively) dichotomous.
At beginning, some data manipulation is made to edge original data with a new categorical variable (number of categories = number of unique vectors), in order to apply another clustering technique.

The main characteristics of input data type are:

- Can not be seen on a classical Cartesian diagram-so, no topological structure of observed phenomenon-.
- Hypothetical clusters are usually unknown.
- Reduction of data dimensions is required to see plots (2D or 3D).
- Techniques of dimensionality reduction, like PCA, could remove variance (features) potentially relevant.
- Few clustering's algorithms and few distance metrics.
- Few clustering evaluation indexes for unsupervised learning

1. Input Data

The input data (Fig 1) is composed of **id** (key-column) and a **list of archives** defined by numbers (1, 2, 3, 401, 800, ecc..).
The column **cd_archivi** points out what archives (columns) have 1. So, user with id AOX17369 has active archives 150 and 800.

	id	cd_archivi	1	2	3	4	5	6	7	9	...	401	411	700	747	772	800	900	902	903	904
0	AOX17369	['150', '800']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0
1	BAI320884	['150']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	BAI789577	['150']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	BAI887483	['150']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	BAI945071	['150']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
569550	RSS3508030	['28']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
569551	RSS3556113	['28']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
569552	RSS3571649	['23']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
569553	RSS3635556	['34']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
569554	CSR2276924	['35']	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

569555 rows × 94 columns

Fig 1 Input Data

Excluding first two columns, each user is represented by a Boolean array like [0,0,0,0,1,0...,1,0].

2. Libraries

Below a list of major packages used for this study.

Package	VersionNote:
hdbscan	0.8.28
kmodes	0.12.2
matplotlib	3.5.3
matplotlib-inline	0.1.6
numpy	1.21.6
pandas	1.3.5
pickle5	0.0.12
pyclustering	0.10.1.2
scikit-learn	1.0.2
scikit-learn-extra	0.3.0
scipy	1.7.3
tensorflow	2.11.0
tensorflow-estimator	2.11.0
tensorflow-intel	2.11.0
tensorflow-io-gcs-filesystem	0.31.0
umap-learn	0.5.6
Python	3.7.12

3. Data Preparation

Before starting any study, a data cleaning and sampling is highly recommended even though input data comes from another data preparation process.

The sample in exam has main correlations as listed:

19-91 are highly correlated = 0.68
56-156 are highly correlated = 0.66
156-91 are correlated = 0.58
700-706 are always together, let us drop one of them
700-6 are correlated = 0.58

To see correlation matrix, click here

Column (archive) 706 has been dropped from df.

Working with a matrix ~500'000 x 94 is very expensive for certain algorithms, thus a sampling is required. The *Pandas* library offers a sample() function that helps to reduce dimension (row) without losing or distorting variance. Below a comparison.

Original variance	**Sample variance**
cd_archivi	**cd_archivi**
['2'] 0.100018	['2'] 0.096304
['2', '73'] 0.054086	['2', '73'] 0.055921
['23'] 0.052636	['101'] 0.050391
['101'] 0.050474	['23'] 0.050303
['28'] 0.026809	['28'] 0.028970
['26'] 0.023545	['21'] 0.024405
['21'] 0.022993	['26'] 0.023703
['35'] 0.021608	['35'] 0.021420
['34'] 0.021373	['17'] 0.021333
['17'] 0.021125	['34'] 0.021069

In the table above there are the 10 most frequent archives (used column cd_archivi for readability). Original variance column is from original dataset (500k rows), Sample variance is from a sample of 1/50 times original dataset (~12k rows).
As clear, the general variance is not distorted in sample dataset.

4. Distance Metrics

Distance metrics that well suit Boolean and categorical data are Hamming, Jaccard and Cosine. Jaccard is preferred since gives importance when value is 1 (asymmetric binary data).

4.1. Jaccard

The Jaccard coefficient measures similarity between finite sample sets and is defined as the size of the **intersection** divided by the size of the **union** of the sample sets. The similarity is defined by a value between 0 and 1. To obtain the dissimilarity, just compute the difference between 1 and Jaccard similarity.

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

An example in Fig 2 shows the result. The intersection (numerator) is 1, the union (denominator) is 2 and the similarity is 0.5. Dissimilarity 1-0.5 = 0.5

```
3 c = [0, 0, 0, 1]
4 d = [0, 1, 0, 1]
5 distance.jaccard(c,d)
```

```
0.5
```

Fig 2 Jaccard example

4.2. Hamming

More easily, Hamming distance is just the number of positions at which the corresponding values are different. As usual, an example below (Fig 3)

```
3 c = [0, 0, 0, 1]
4 d = [0, 1, 0, 1]
5 distance.hamming(c,d)
```

```
0.25
```

Fig 3 Hamming example.

The library used is *scipy.distance* applies Hamming distance this way (Fig 4)

$$\frac{c_{01} + c_{10}}{n}$$

Fig 4 Hamming formula from scipy

So, for example in Fig 3, the result is ¼ = 0.25

4.3. Others

Other distance metrics used for some validation indexes are **Euclidean**, **cosine**, **dice**, **Rogerstanimoto**, **Chebyshev**, **Manhattan**, **Minkowski**.

5. Data Transformation

In order to apply different algorithms and visualize clusters, data transformation is fundamental.

5.1. PCA - Principal Component Analysis

PCA is a dimensionality reduction done by a new coordinate system which aims to capture the largest variance in the data. For more information see here.

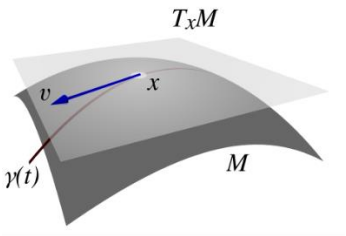
In this study will be analyzed PCA with 3 and 5 principal components.

5.2. UMAP - Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualization similarly to t-SNE, but also for general non-linear dimension reduction. The algorithm is founded on three assumptions about the data.

- The data is uniformly distributed on **Riemannian** manifold.
- The Riemannian metric is locally constant (or can be approximated as such).
- The manifold is locally connected.

From these assumptions it is possible to model the manifold with a fuzzy topological structure. The embedding is found by searching for a low dimensional projection of the data that has the closest possible equivalent fuzzy topological structure.



For more information, visit links below.

1802.03426 (arxiv.org)

How UMAP Works — umap 0.5 documentation (umap-learn.readthedocs.io)

interpretation - Clustering on the output of t-SNE - Cross Validated (stackexchange.com)

Related to above link, there's explanation on why Data Transformation techniques chosen are UMAP and MDS (instead of t-SNE). Furthermore, UMAP's looping cycle is better than t-SNE, since it is faster on large dataset.

Understanding UMAP (pair-code.github.io)

Another good explanation-a little bit ironical- can be found here and here.

A relevant feature of this Data Transformation technique is that automatically group together similar points by giving them similar coordinates.

5.3. MDS - MultiDimensional Scaling

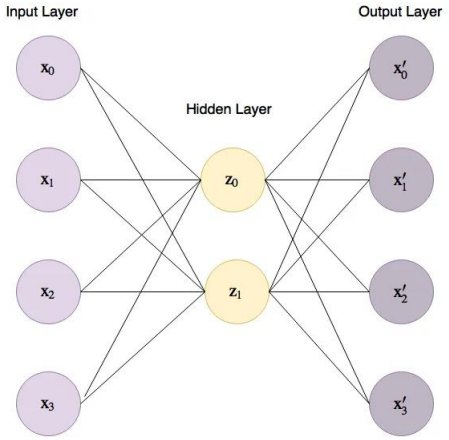
MultiDimensional Scaling is essentially a technique to plot a matrix with a lot of dimensions.

Given a distance matrix with the distances between each pair of objects in a set, and a chosen number of dimensions, N, an MDS algorithm places each object into N-dimensional space (a lower-dimensional representation) such that the between-object distances are preserved as well as possible (source: Wiki).

5.4. Auto Encoder

Auto Encoders are neural networks where the network aims to predict the input (the output is trained to be as similar as possible to the input) by using less hidden nodes (on the end of the encoder) than input nodes by encoding as much information as it can to the hidden nodes.

A basic auto encoder for a 4-dimensional dataset would look like the figure on the right.



The lines connecting between the input layer to the hidden layer are called the “encoder” and the lines between the hidden layer and the output layer the “decoder”.

Auto Encoders starts with some random low dimensional representation and gradient will descent towards their solution by changing the weights that connect the input layer to the hidden layer, and the hidden layer to the output layer.

The problem is substantially a minimization of error (Mean Squared Error,). At each iteration, the model compares predicted data with desired output with MSE and modifies weights to reduce loss.

6. Data Visualization

This chapter is an application of Data Manipulation techniques.

6.1. PCA

2D plot with **30.89 %** of explained variance :

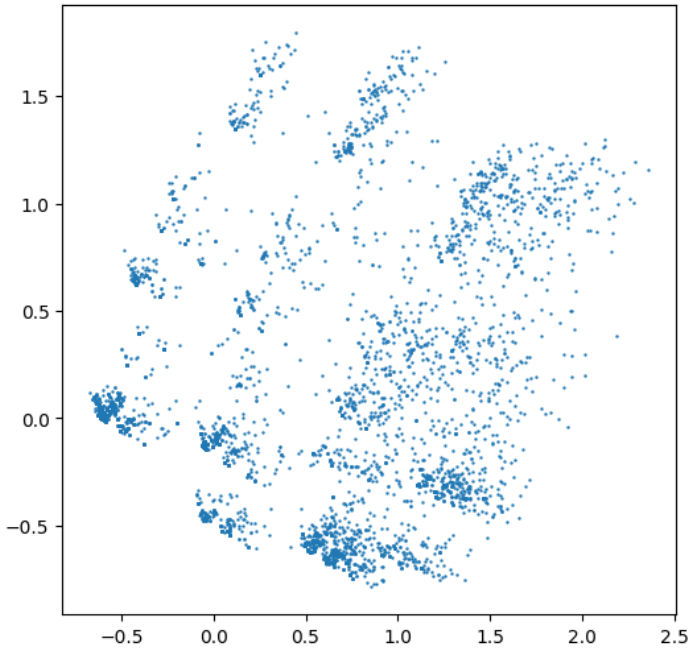


Fig 5 PCA (n_components = 2)

3D plot with **45.5 %** of explained variance:

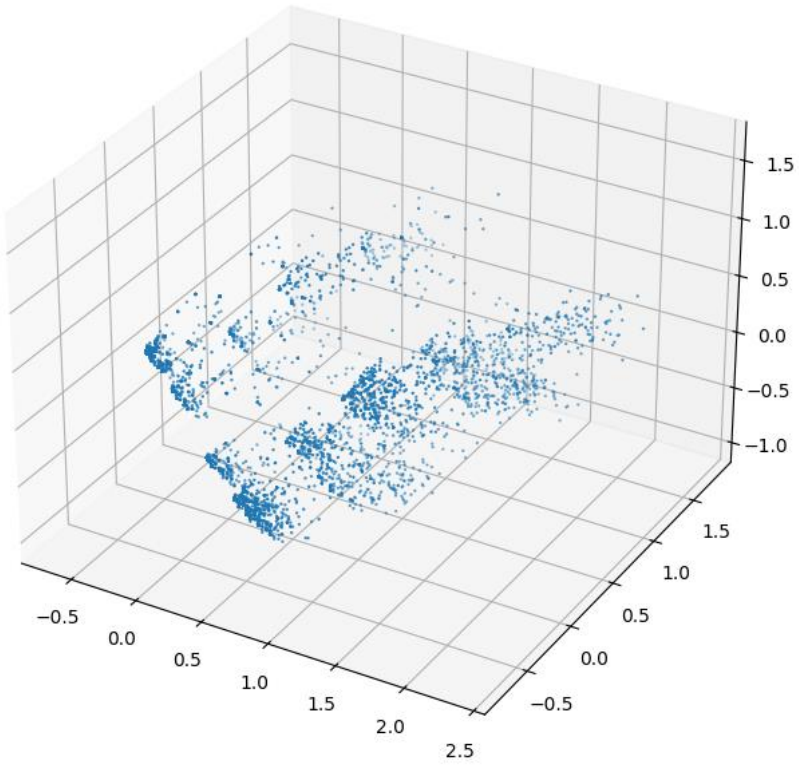


Fig 6 PCA (n_component = 5)

6.2. UMAP

UMAP is more customizable so, the images that follow, will be a bouquet of plots with different parameters. The scope is to select the best plots by how well separated and defined 'clusters' are.
It's chosen to reduce the sample dataset in 2 and 3 dimensions (info here).

Main hyperparameters:

- n_neighbors

low values of n_neighbors will force UMAP to concentrate on very local structure (potentially to the detriment of the big picture), while large values will push UMAP to look at larger neighborhoods of each point when estimating the manifold structure of the data, losing fine detail structure for the sake of getting the broader of the data.

Interval tested = [5, 15, 30, 60]

- min_dist

controls how tightly UMAP is allowed to pack points together. Low values of min_dist will result in clumpier embeddings -useful in clustering-, large value will focus on the preservation of the broad topological structure instead.

Interval tested = [0.0125, 0.05, 0.2, 0.8]
Chosen values are obtained from a study on distance matrix.
With **Jaccard**, some minimum values are:

array([0. , 0.08333333, 0.09090909, 0.1 , 0.11111111, ...])

With **Hamming**, some minimum values are:

array([0. , 0.01086957, 0.02173913, 0.0326087 , 0.04347826, ...])

Being a triangular matrix, max and min are 0 and 1. Excluded 0 that refers to identical Boolean array, minimum distances with Jaccard start with 0.08, 0.09, 0.1, etc. So, min_dist = 0.05 just starts aggregating similar vectors into 'clusters. With min_dist = 0.0125 no groups are created; each unique vector has its own coordinates.

With Hamming, minimum distances are 0.01, 0.02, 0.03, etc. With min_dist set to lower, 0.0125, there are slightly different vectors already grouped.

Below some plots on how they look on 2Dimension and 3Dimensions.
Fast recap:

- Jaccard + min_dist = 0.0125: just an UMAP representation of original input, without groups.
- Hamming + min_dist = 0.0125: almost identical vectors are already grouped together

2D PLOT with UMAP n_components = 2

[Ctrl + Click for a better resolution image]

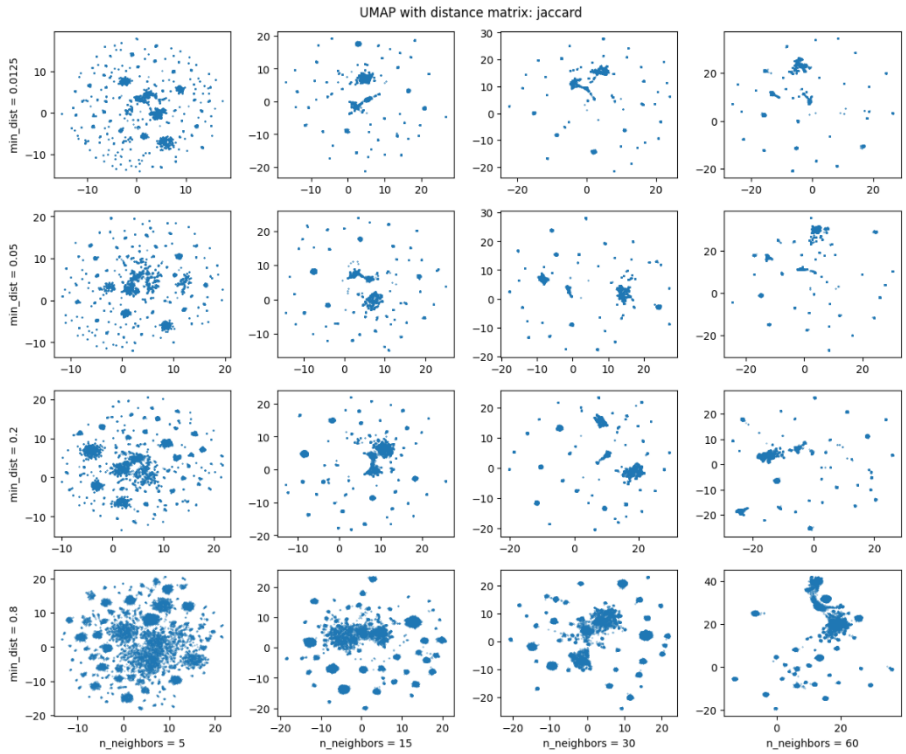


Fig 7 UMAP + Jaccard

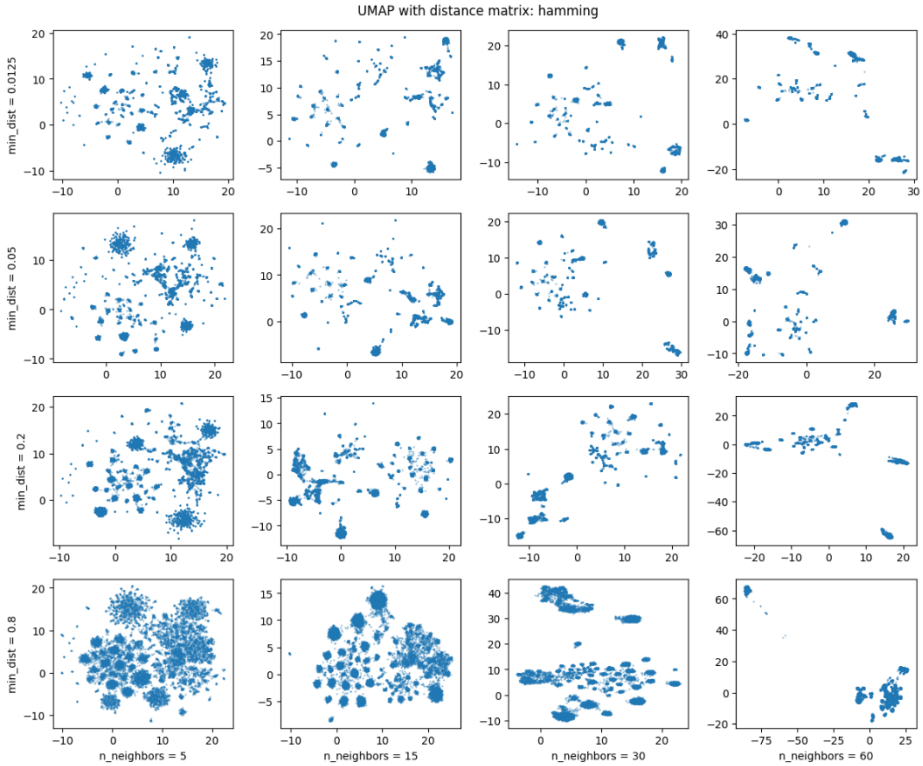


Fig 8 UMAP + Hamming

3D PLOT with UMAP n_components = 3

3D plots cannot be seen like 2Ds, below a list of images of the best ones.

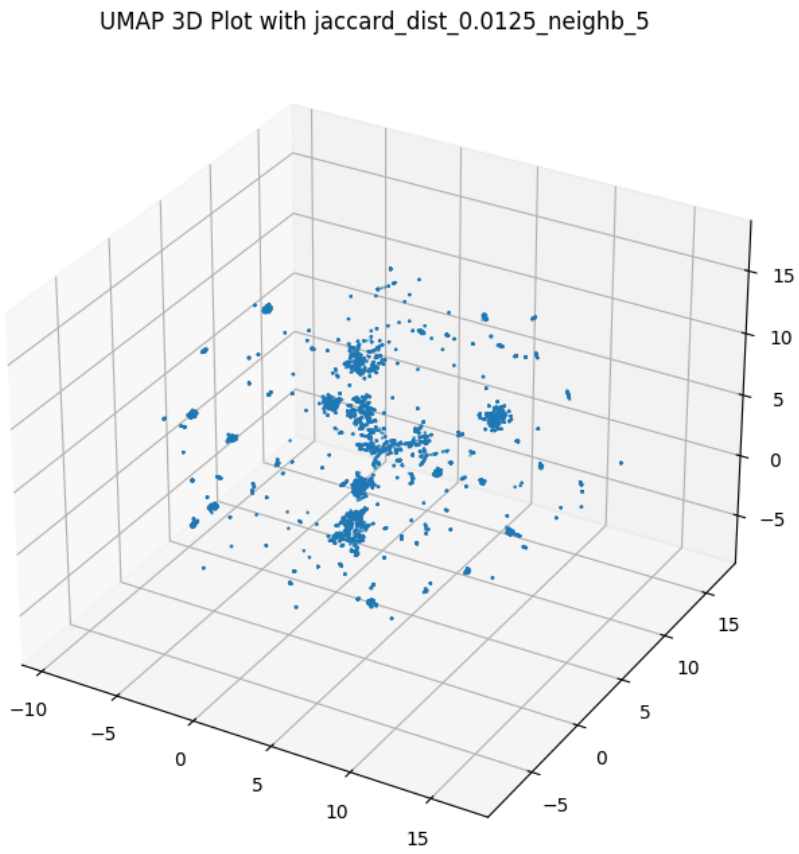


Fig 9 UMAP + Jaccard with min_dist = 0.0125

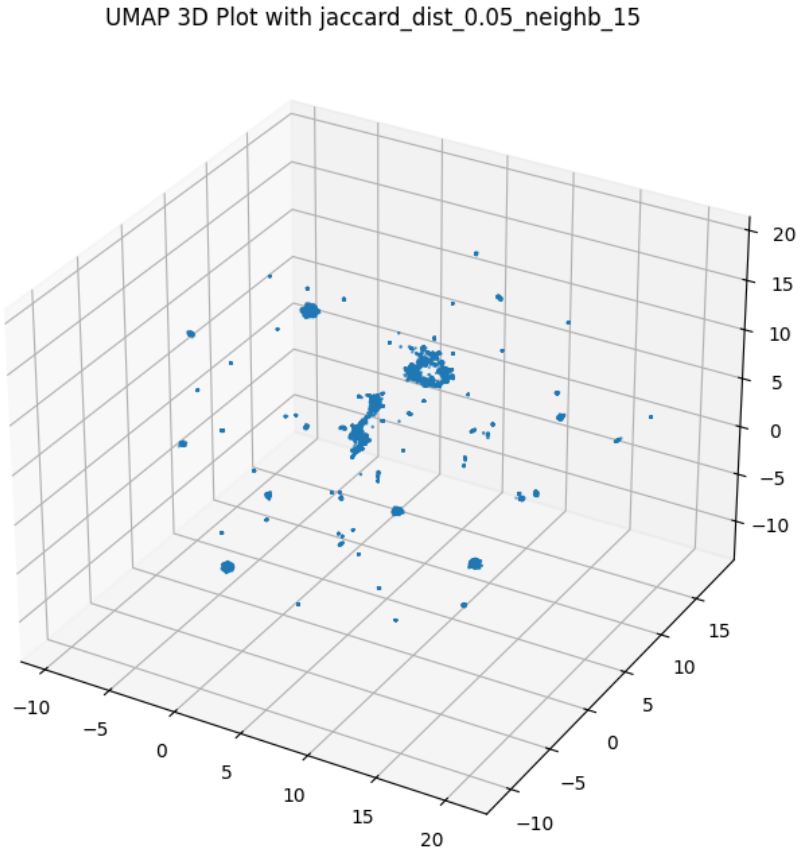


Fig 10 UMAP + Jaccard min_dist = 0.05

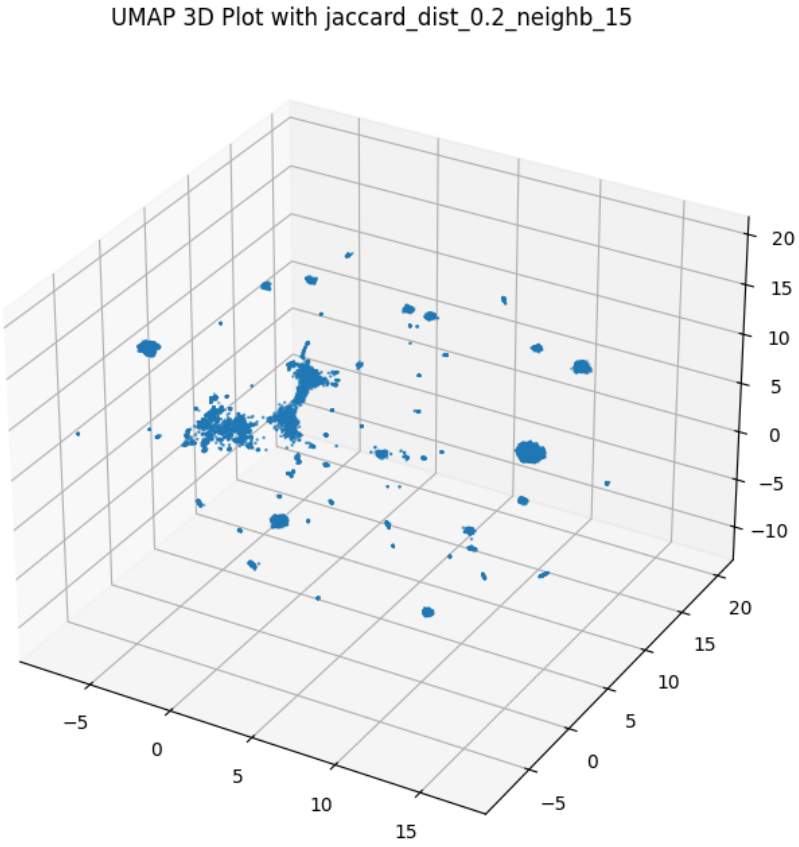


Fig 11 UMAP + Jaccard min_dist = 0.2

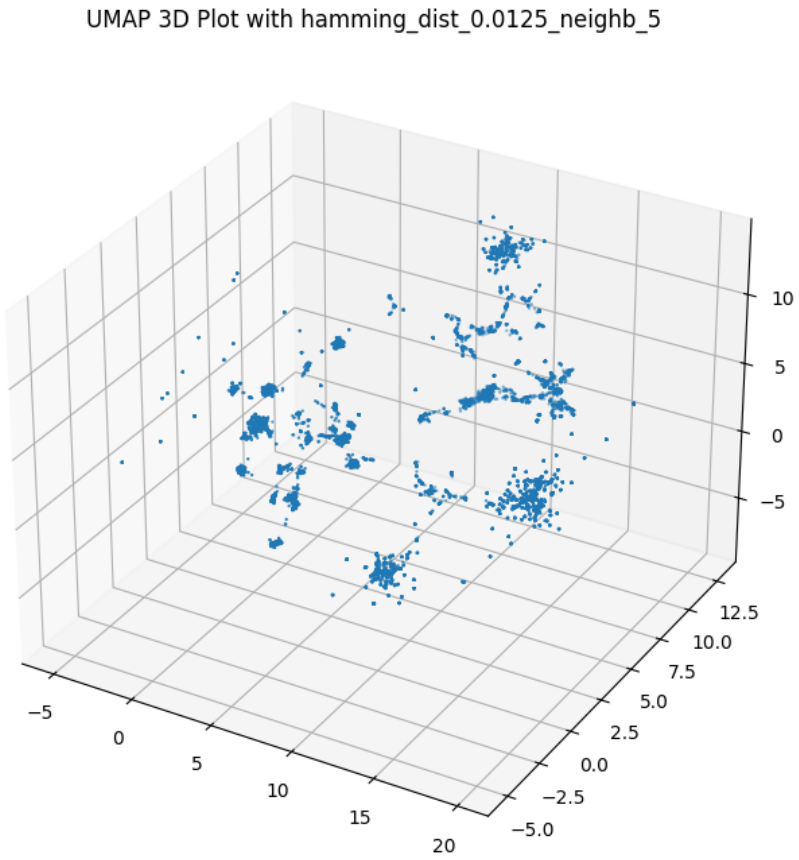


Fig 12 UMAP + Hamming min_dist = 0.0125

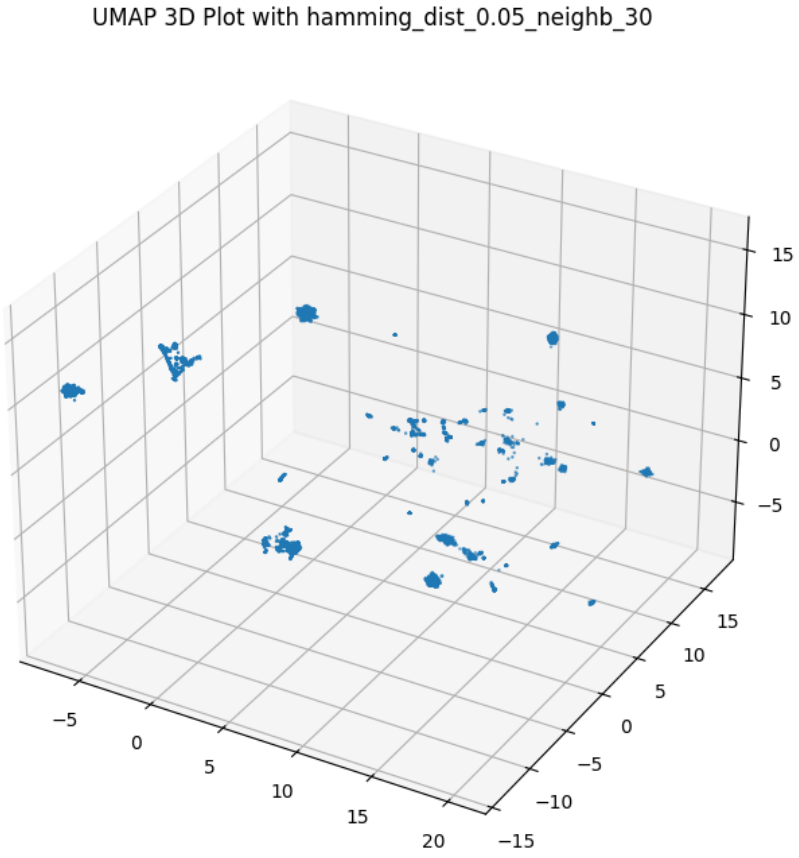


Fig 13 UMAP + Hamming min_dist = 0.05

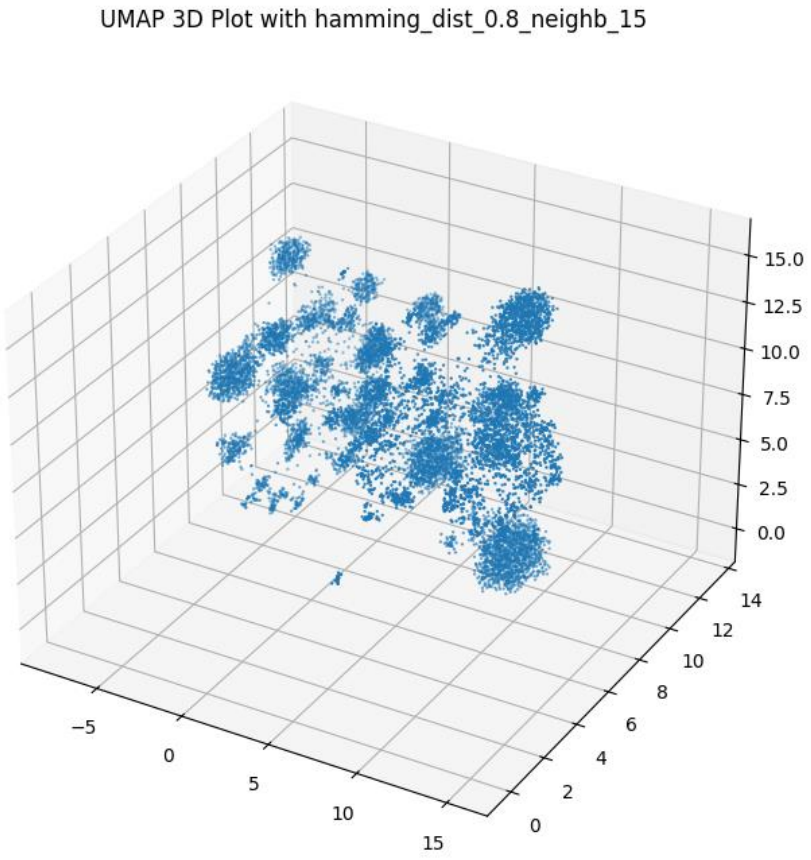
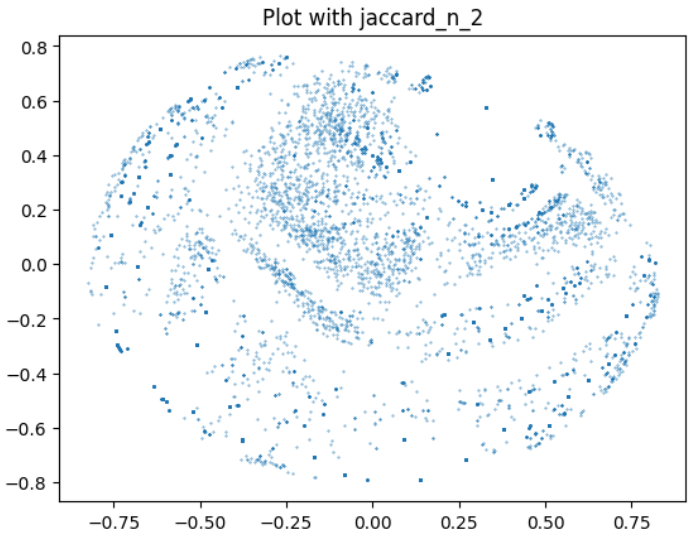
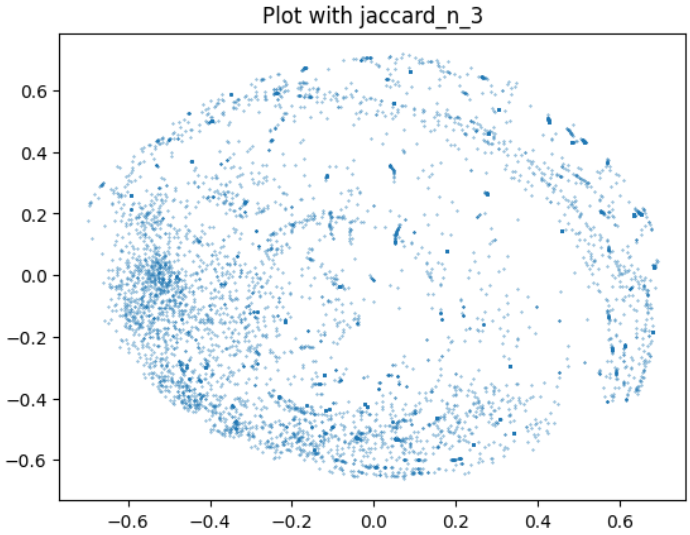


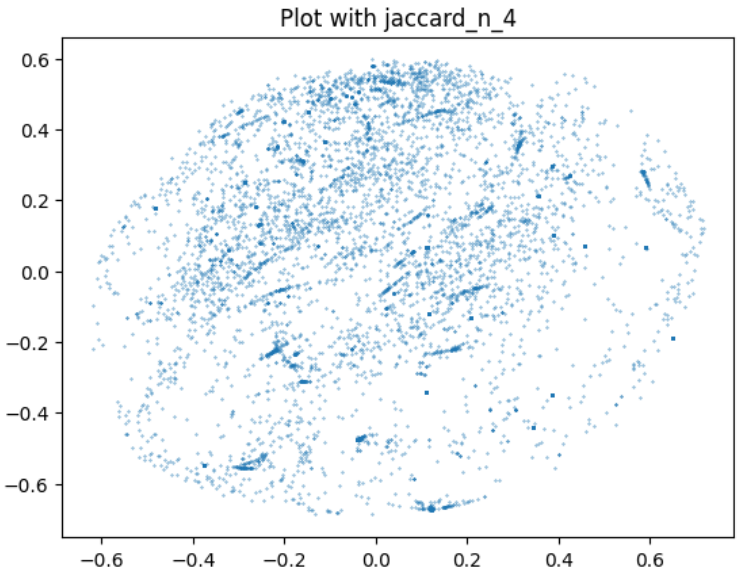
Fig 14 UMAP + Hamming min_dist = 0.05

6.3. MDS

MultiDimension Scaling is well known for being particularly **slow**. To correctly define the number of dimensions, it should run several times (like PCA with explained variance) but it would require days. In this document, MDS was tested only on pre-computed distance with Jaccard, and with n_components from 2 to 4 (~5 hours running).

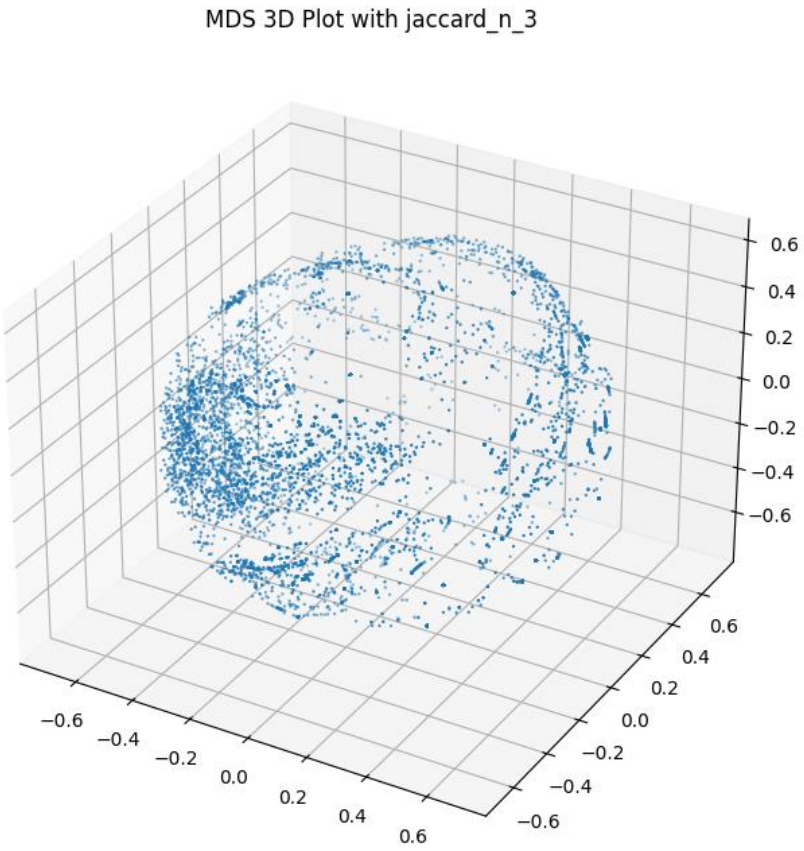


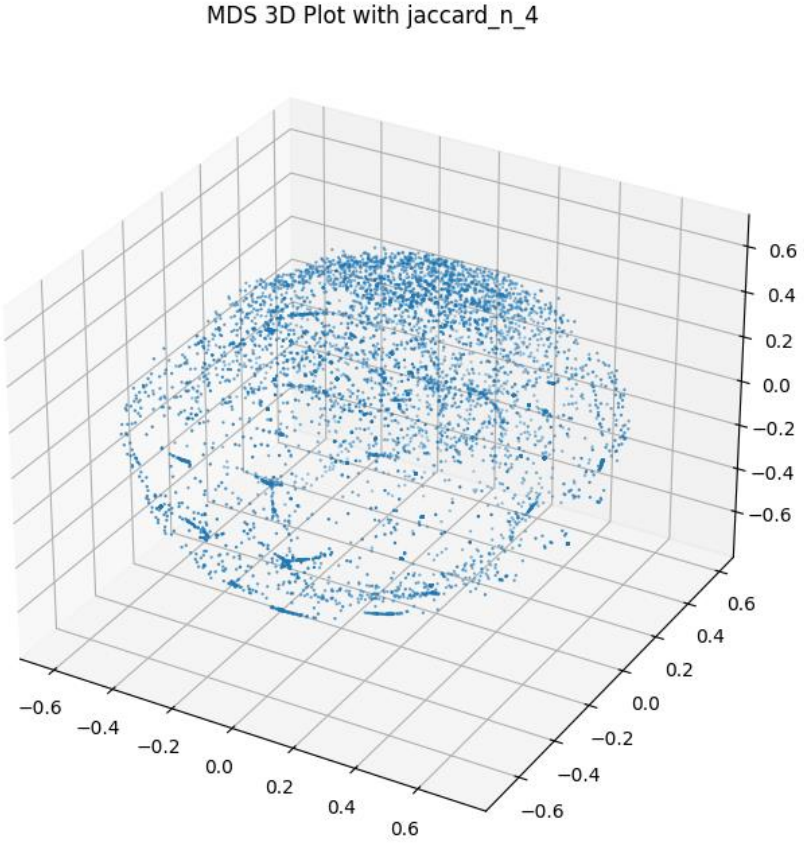




Unfortunately, it is time-expensive so cannot be tuned for better results (stress function for comparison as explained here).

Anyway, below some 3D plots of MDS with components equals to 3 and 4.



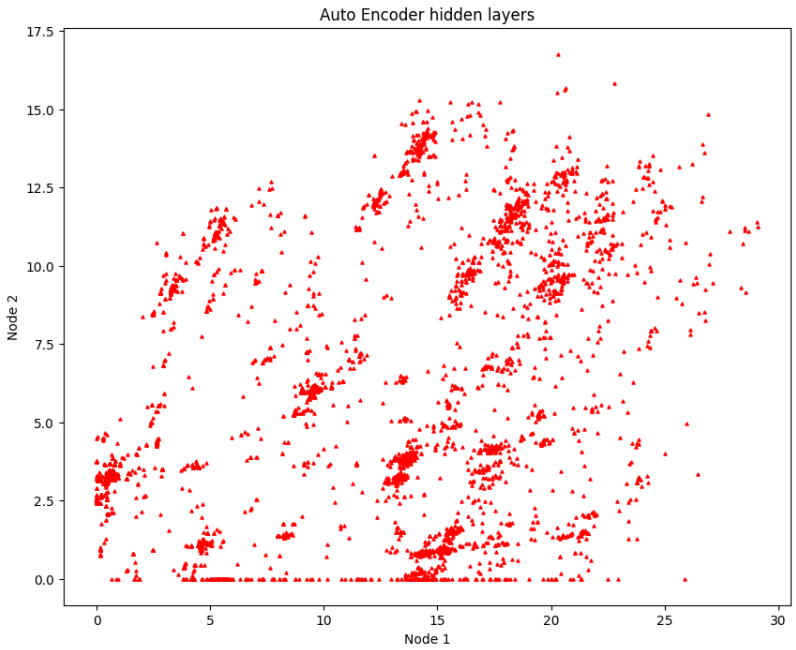


It seems to produce spherical data.

6.4. Auto Encoder

As already explained, Auto Encoder works with the same train and test data so, in this case, boolean vectors. The neural network created has an input layer of 92 neurons (same of boolean vector's dimensions), two neurons into the hidden layer and 92 neurons for the output. After training, the hidden layer's values are extracted and managed as a data frame.

Below how it looks alike.



Auto Encoder with 2 neurons as the hidden layer

There are similarities, by viewing plots, between Auto Encoder and PCA. In this very restrictive dimensionality reduction, there are several records flattened with 0 on y-axis.

The table below is how it looks the input data frame 'reconstructed' by Auto Encoder and original data on the right. The more it looks like the input data frame and lower Mean Squared Error there has been.

Auto Encoder output

	1	2	3	4	5	6	7	9	10	11	12	...	301	401	411	700	747	772	800	900	902	903	904
0	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
7	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
9	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
12	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0

15 rows × 92 columns

Original Boolean data

	1	2	3	4	5	6	7	9	10	11	12	...	301	401	411	700	747	772	800	900	902	903	904
198862	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
431439	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
229408	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
307568	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
230513	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
510992	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
61561	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
48790	0	1	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	1	0	0
269735	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
156796	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
377875	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
480033	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
17284	0	1	1	1	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	0	0
327077	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
103455	0	0	0	1	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0

15 rows × 92 columns

Auto Encoder's output
For a better view, floats have been transformed into boolean (if x >= 0.5 = 1 else 0)

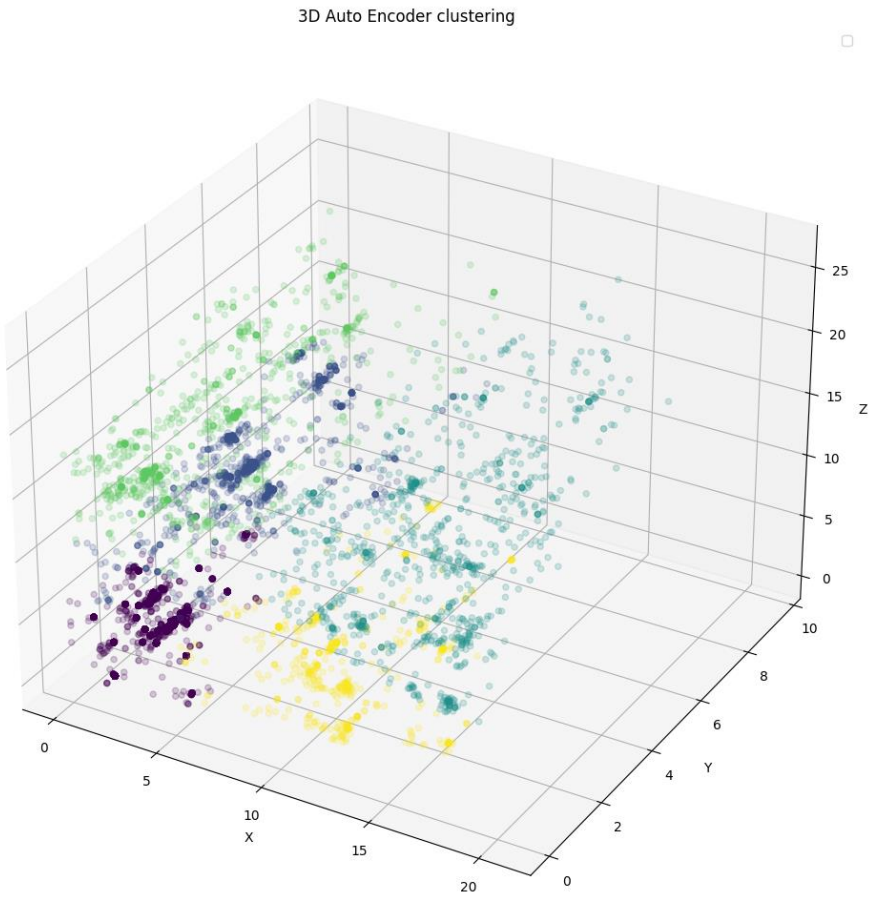
In particular, the model used has obtained a loss (estimated with Mean Squared Error) of 0.014.

```
{'loss': <tf.Tensor: shape=(), dtype=float32, numpy=0.014394801>}
```

Due to limited time available and due to high similarities with PCA model, this data frame is not studied furthermore.

Anyway, Auto Encoder could obtain better results with more tuning. What is shown above, has required just thirty epochs.
It was tried with 200 epochs hoping to find a local-minimum but, after 2/3 steps, there was always a very small improvement. Training ended by reaching maximum number of epochs instead of convergence.
Since it was time consuming and irrelevantly improvable, 30 epochs are enough (MSE graph lookalike f(x)=1/x).

Another try is made by using three dimensions instead of two.



Auto Encoder with 3 dimensions and K-Means clustering with K=5

The performance is better-obviously- if compared to two dimensions reduction above. As shown, MSE is lower.

```
{'loss': <tf.Tensor: shape=(), dtype=float32, numpy=0.011942712>}
```

Comparison between input (original) data and decoded data

Auto Encoder output

	1	2	3	4	5	6	7	9	10	11	12	...	301	401	411	700	747	772	800	900	902	903	904
11376	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11377	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11378	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11379	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11380	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0	0
11381	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11382	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11383	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11384	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11385	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11386	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11387	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11388	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11389	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
11390	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0

15 rows × 92 columns

Original Boolean data

	1	2	3	4	5	6	7	9	10	11	12	...	301	401	411	700	747	772	800	900	902	903	904
432495	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
57623	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
37868	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
112506	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
475349	0	1	0	0	1	0	0	0	0	0	0	...	1	0	1	0	0	0	0	0	0	0	0
327506	0	0	0	1	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
279181	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
82850	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
39510	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
204131	0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
46763	0	1	0	0	1	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
177730	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
514763	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
381898	0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
174118	0	1	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	1	0	0

15 rows × 92 columns

Auto Encoder's output

For a better view, floats have been transformed into boolean (if x >= 0.5 = 1 else 0)

7. Algorithms

7.1. Center-based Clustering Algorithms

Center-based algorithms are very efficient for clustering large databases and high-dimensional databases. Usually, center-based algorithms have their own objective functions, which define how good a clustering solution is. The goal of a center-based algorithm is to minimize its objective function. Clusters found by center-based algorithms have convex shapes and each cluster is represented by a center.
Therefore, center-based algorithms are not good choices for finding clusters of arbitrary shapes.

7.1.1. K-Means

K-Means is the most known clustering technique due to its simplicity and adaptability to big data. It is designed to cluster numerical data in which each cluster has a center called the mean. The distance metric is Euclidean.

Some limitations of K-Means like initialization centroids and pre-determined number of clusters are solved, respectively, by x-means (range of clusters, chooses best #clusters with BIC) the and k-harmonic Means (harmonic mean instead of minimizing distance within-cluster). Those two models are not applied in this document.

An example of K-Means output:

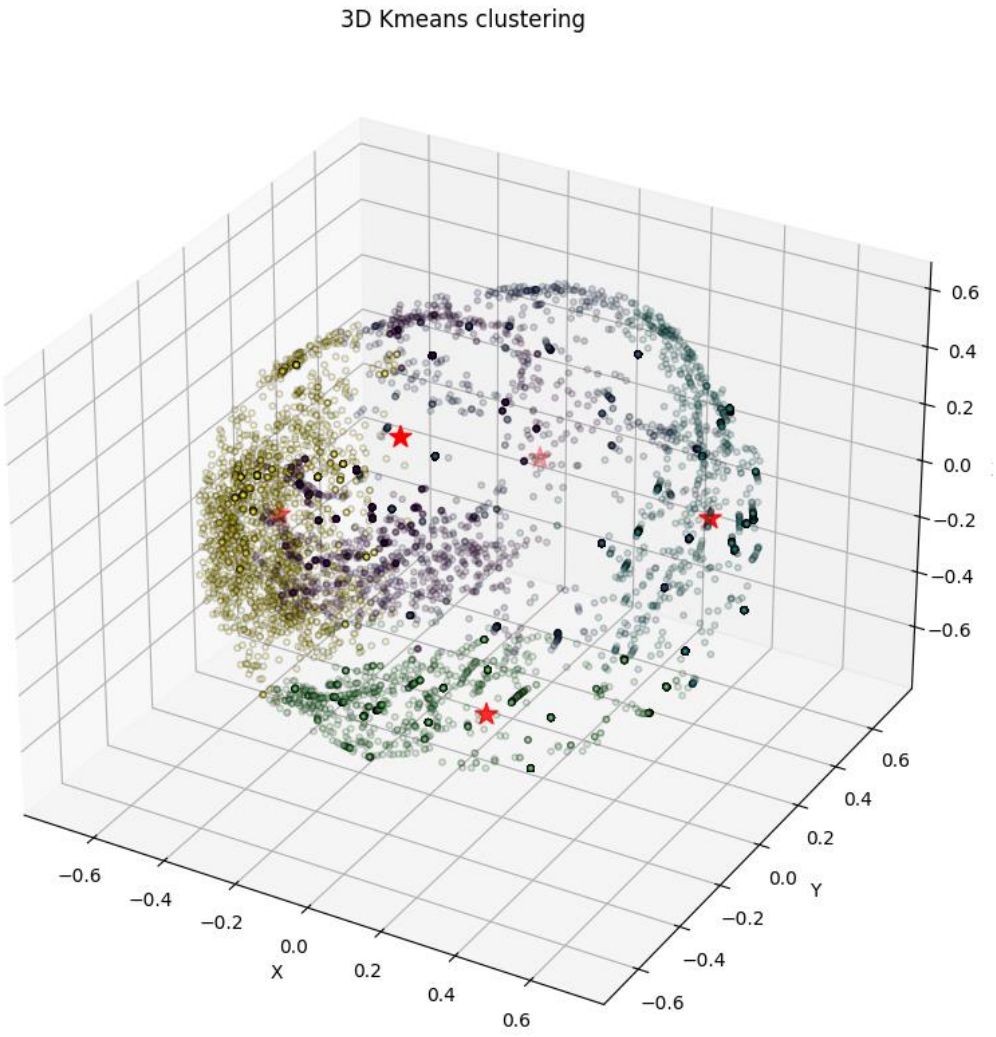


Fig 15 An example, K-Means with 5 clusters on MDS with Jaccard distance matrix and 3 dimensions

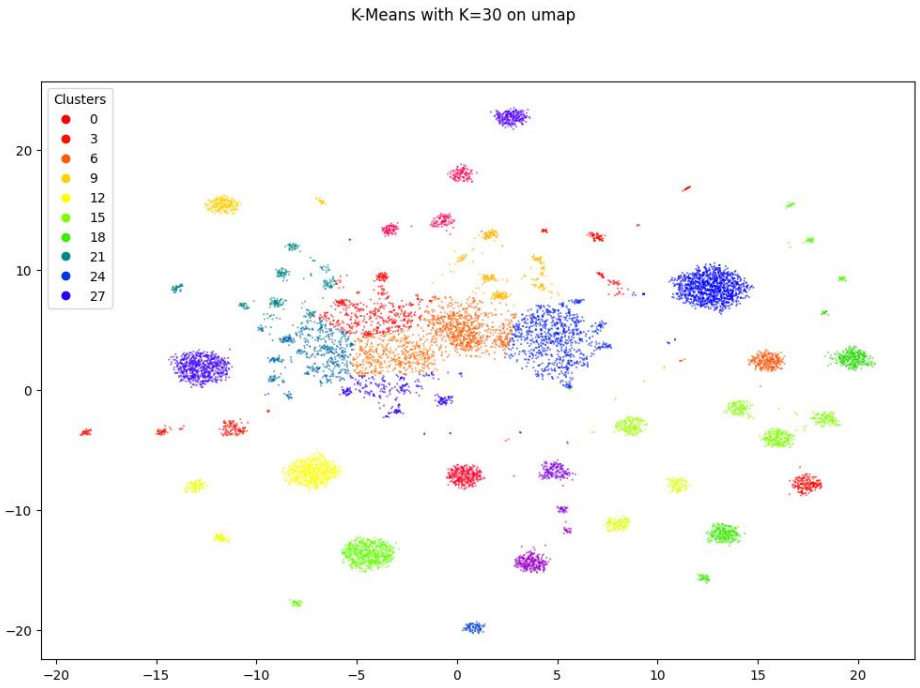


Fig 16 K-Means with k=30 on UMAP and Jaccard

7.1.2. KModes

The k-modes algorithm (Huang, 1997b, 1998) comes from the k-means algorithm, and it was designed to cluster **categorical data** sets (no need either to standardize data nor reduce dimensions). The main idea of the k-modes algorithm is to specify the number of clusters (say, k) and then to select k initial modes, followed by allocating every object to the nearest mode.

The k-modes algorithm uses the simple match dissimilarity measure to measure the distance of categorical objects.

It has some important properties:

- It is efficient for clustering large data sets.
- It also produces locally optimal solutions that are dependent on initial modes and the order of objects in the data set (Huang, 1998).
- It works only on categorical data.

Some examples of K-Mode's output plotted on 2D data frame:

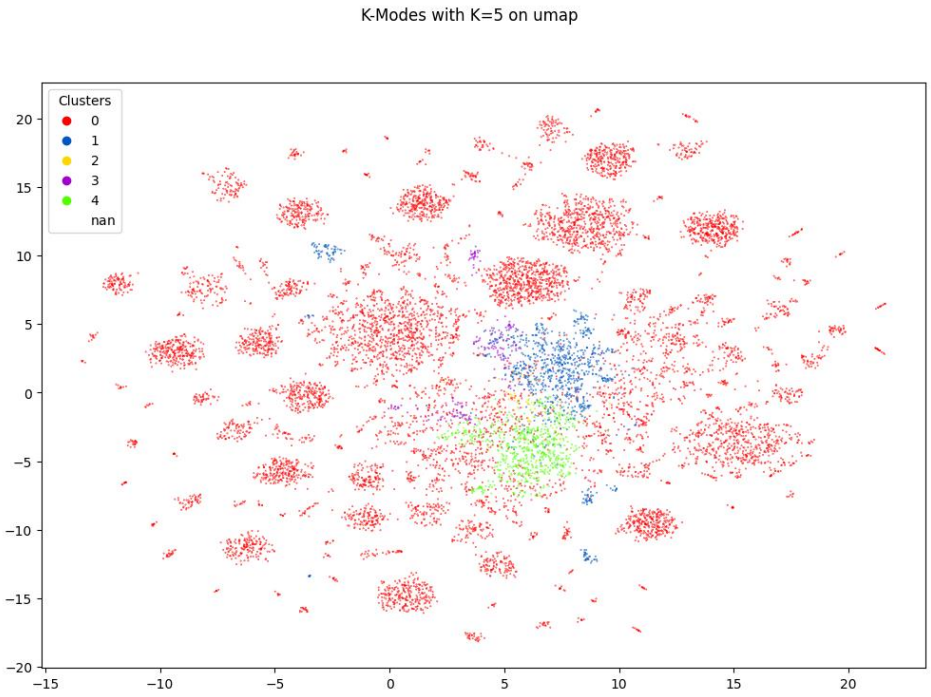


Fig 17 K-Modes with k=5 trained with boolean vectors, plot on UMAP

In the figure above, algorithm is trained on boolean vectors and the resulting clusters are plotted on UMAP data frame with Jaccard distance. There's clearly confusion. Keep in mind that boolean data frame and UMAP data frame represent the same sample in a different 'language, thus **it is not obvious that clusters are well defined and visually nice**.

Another try was done by training K-Modes only with column cd_archivi and the result printed on same UMAP df. It seems slightly better than previous one.

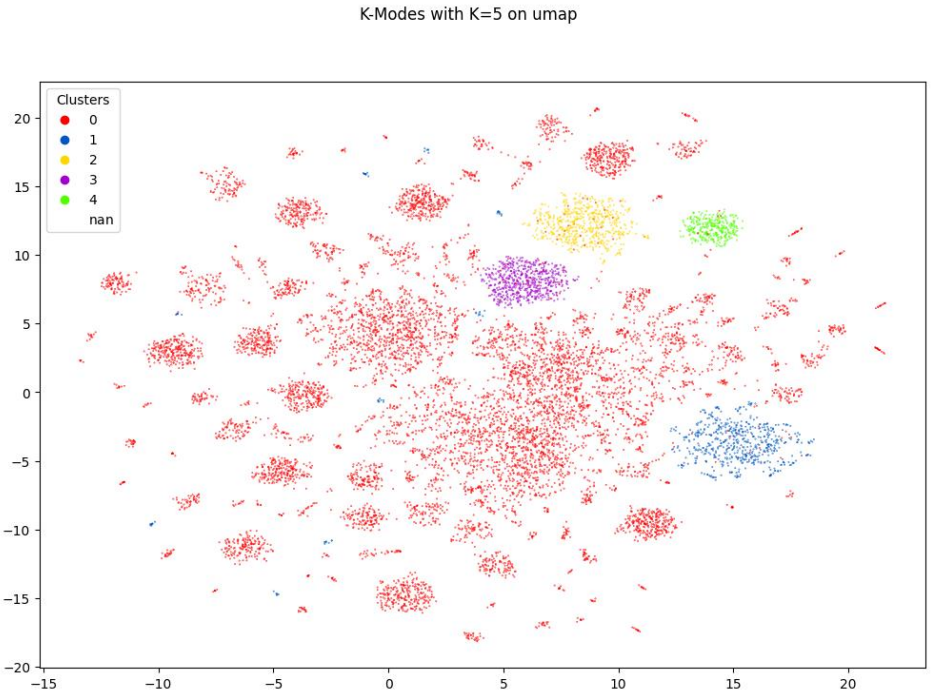


Fig 18 K-Modes with K=5 trained with cd_archivi, plot on UMAP

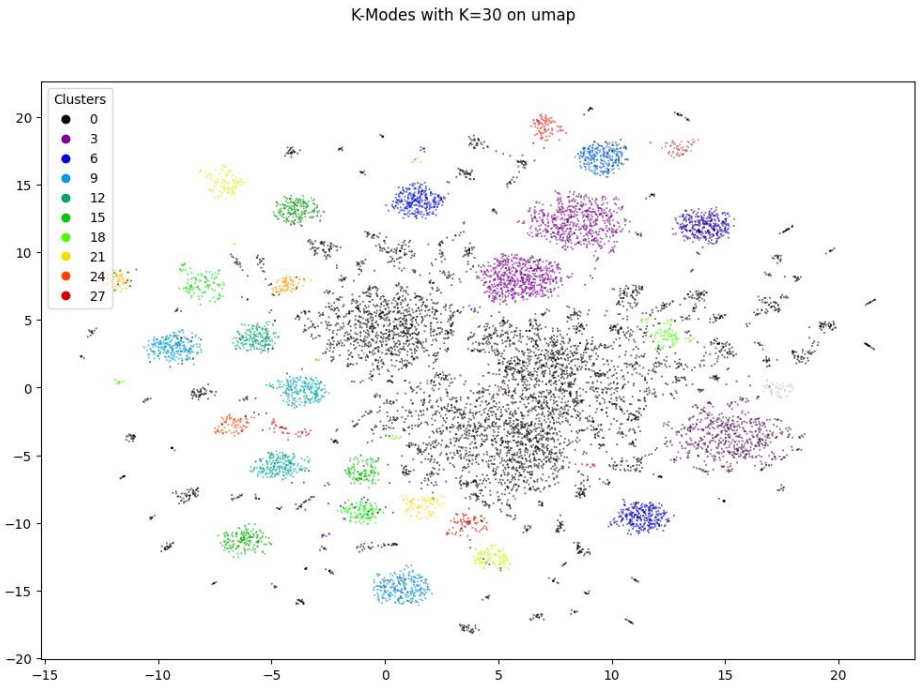


Fig 19 Another plot with K=30

7.1.3. CLARA

Clara is an optimized model for Big Data derived by PAM (Partitions Around Medoids) belonging to bigger family of K-medoids.

K-medoids is a clustering algorithm which is more resilient to outliers compared to K-means. Similar to K-means, the goal of K-medoids is to find a clustering solution that minimizes a predefined objective function.
The K-medoids algorithm aims to minimize the absolute error criterion rather than the SSE. Like K-means clustering algorithm, the K-medoids proceeds iteratively until each representative object is actually the medoid of the cluster.

Below some plots.

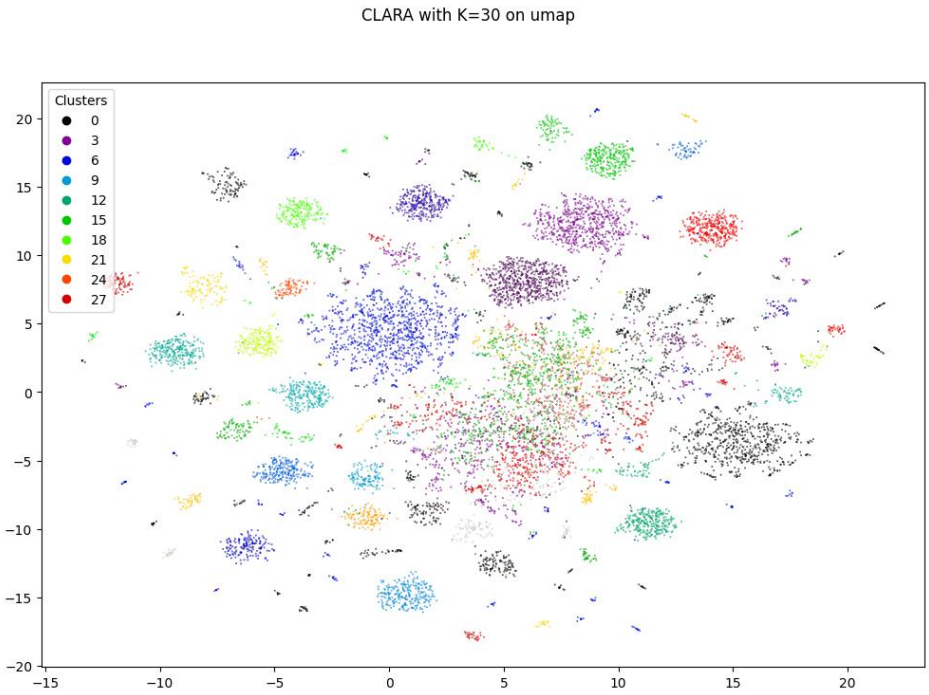


Fig 20 CLARA with K=30, plot on UMAP with Jaccard

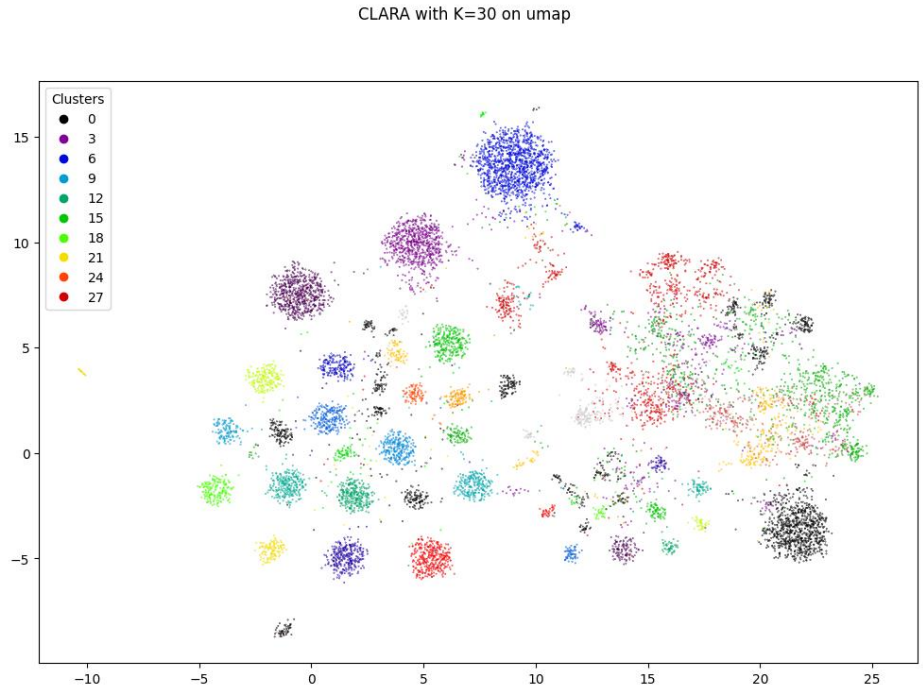


Fig 21 CLARA with k=30, plot on UMAP with hamming

7.2. Density-based Clustering Algorithms

The density-based clustering approach is a methodology that is capable of finding arbitrarily shaped clusters, where clusters are defined as dense regions separated by low-density regions.

A density-based algorithm needs only one scan of the original data set and can handle noise. The number of clusters is not required, since can automatically detect the clusters.

7.2.1. HDBSCAN

HDBSCAN is a hierarchical clustering algorithm based on DBSCAN (Density-Based Spatial Clustering of Applications with Noise).

The algorithm works by following those steps:

1. Transform the space according to the density/sparsity.
2. Build the minimum spanning tree of the distance weighted graph.
3. Construct a cluster hierarchy of connected components.
4. Condense the cluster hierarchy based on minimum cluster size.
5. Extract the stable clusters from the condensed tree.

For further information click here.
An example of result in the plot below:

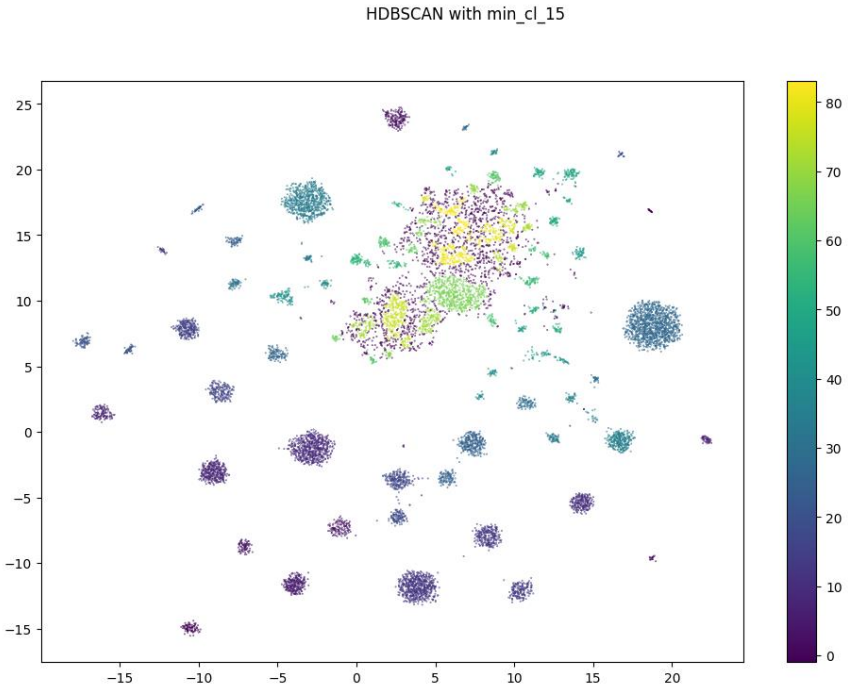


Fig 22 HDBSCAN with min_cluster_size = 15 on boolean data. Plot with UMAP data frame

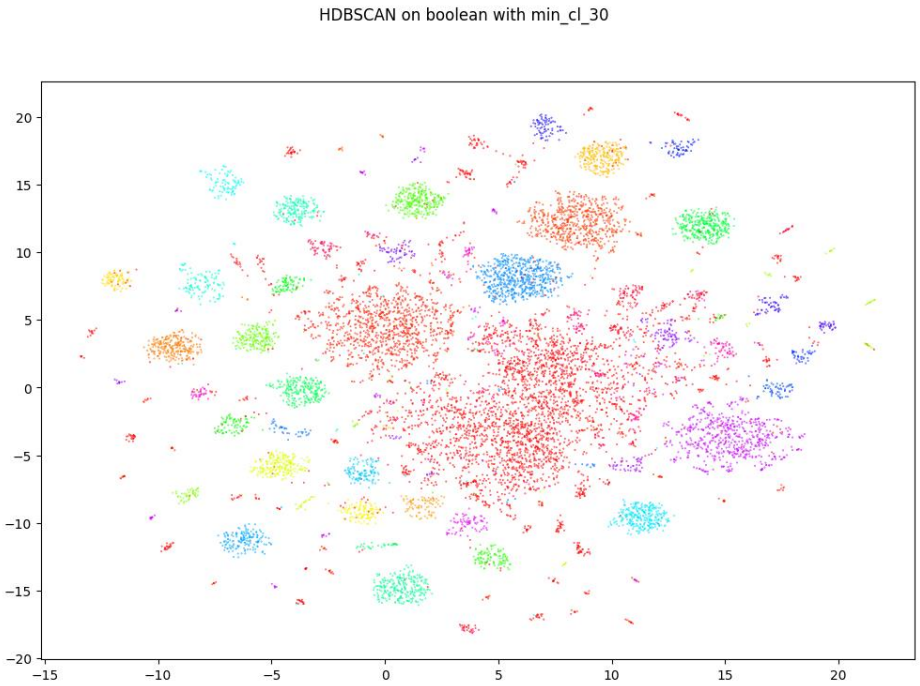


Fig 23 HDBSCAN with min_cluster_size = 30 on boolean data.
Plot with UMAP (jaccard min dist 0.8 e n_neighbors 5).
49 clusters found + 1

7.3. Probabilistic Models for Clustering

Probabilistic model-based clustering approaches attempt to optimize the fit between the observed data and some mathematical model using a probabilistic approach. Such methods are often based on the assumption that the data are generated by a mixture of underlying probability distributions.

In practice, each cluster can be represented mathematically by a parametric probability distribution (e.g. Gaussian distribution). Thus, the clustering problem is transformed into a parameter estimation problem since the entire data can be modeled by a mixture of K component distributions.

Most famous models are Gaussian Mixture Model, for continuous variables, and Bernoulli Mixture Model for boolean variables.

7.3.1. Bernoulli Mixture Model

In the case under study, since we have dichotomous data, each probability distribution is a p-dimensional vector (where p is the number of archives) of Bernoulli distributions, where each value represents the probability of having 1 in that feature.
Therefore, for each partition to analyze (the partitions differ from each other based on the number of clusters, which must be decided beforehand), the following parameters are estimated:

- the success probabilities of the Bernoulli vectors
- the posterior probabilities, for each unit, of belonging to the cluster

The algorithm E-M (Expectation Maximization) estimate these parameters. It is an iterative method based on the likelihood of the parameters (the higher it is, the more likely the estimated parameters are the real ones); the algorithm stops processing as soon as the likelihood no longer increases by a predetermined minimal amount ε. Once the algorithm finishes, the posterior probabilities of belonging to a cluster are observed for each unit and, after identifying the highest probability, the unit is assigned to that cluster. Finally, the partitions obtained are evaluated using various statistical indices existing in the literature (such as AIC and BIC) to determine the best partition.

The advantages of this approach are that it provides a clear definition of clusters, and the issues that arise concerning the determination of the number of clusters and the choice of an appropriate classification method become a single problem of model selection.

Some visual results can be seen in the chapter 9.1 Conclusions

8. Clustering evaluation indexes

In the literature of data clustering, a lot of algorithms have been proposed for different applications and data sizes. But clustering a data set is an unsupervised process; there are no predefined classes and no examples that can show that the clusters found by the clustering algorithms are valid. To compare the clustering results of different clustering algorithms, it is necessary to develop some validity criteria. Also, if the number of clusters is not given in the clustering algorithms, it is a highly nontrivial task to find the optimal number of clusters in the data set. To do this, we need some cluster validity methods.

8.1. Elbow

Elbow method is a poorly technique since rarely shows an optimal point where **diminishing results are no longer worth the additional cost**. Below an example

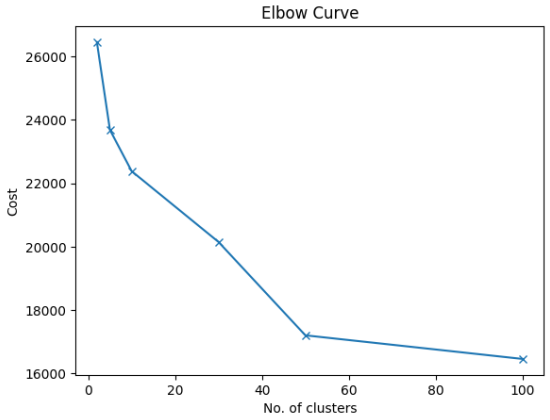


Fig 24 Elbow curve with KModes on boolean data

8.2. Silhouette

Silhouette is another well-known technique. It is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a **high value** indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

A clustering with an average silhouette width of over 0.7 is "strong", a value over 0.5 "reasonable" and over 0.25 "weak", but with increasing dimensionality of the data, it becomes difficult to achieve such high values because of the curse of dimensionality, as the distances become more similar. The silhouette score is specialized for measuring cluster quality when the clusters are convex-shaped and may not perform well if the data clusters have irregular shapes or are of varying sizes. Below some examples:

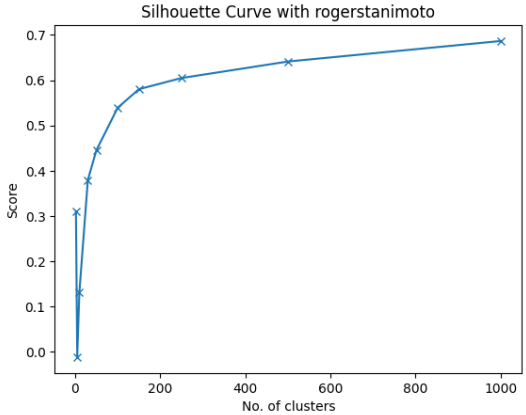


Fig 25 Silhouette (metric Rogerstanimoto) on CLARA's clusters

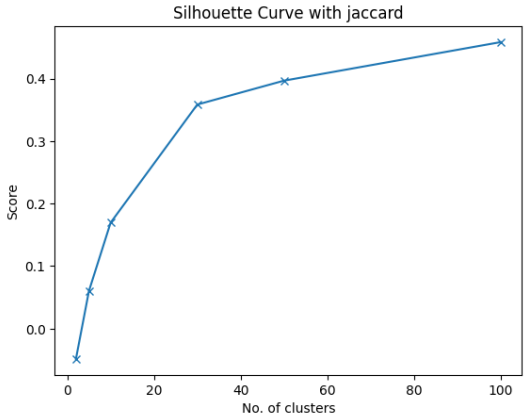


Fig 26 Silhouette (metric jaccard) on Kmodes's clusters (input dataframe cd_archivi)

8.3. Calinski-Harabasz Index (Pseudo-F)

Calinski–Harabasz (CH) Index is defined as the ratio of the between-cluster separation (BCSS) to the within-cluster dispersion (WCSS), normalized by their number of degrees of freedom. A **higher** Calinski-Harabasz score relates to a model with better defined clusters.

Example:

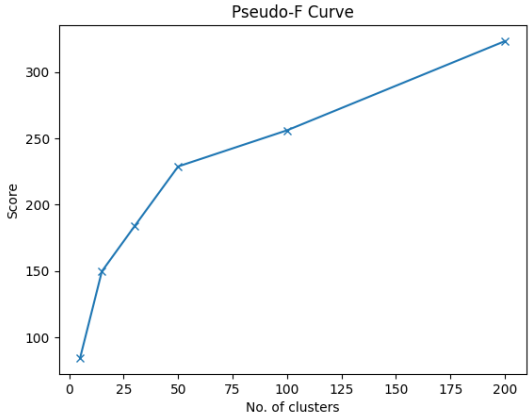


Fig 27 Pseudo-F on HDBSCAN (raw data)

8.4. Davies-Bouldin Index

The score is defined as the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Thus, clusters which are farther apart and less dispersed will result in a better score.

The **minimum score** is zero, with lower values indicating better clustering.

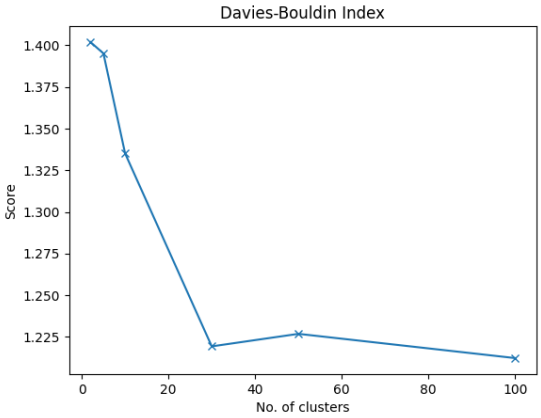


Fig 28 Davies-Bouldin on KModes's clusters (input cd_archivi)

8.5. Violin Index

Due to limited time and focus on new indexes, violins are not included in this study.

9. Practical Applications

This bigger chapter it is aimed to code and show comparisons between approaches.

First of all, the study is splitted into four main "paths" reliant to several **input data frames**:

1. **Categorical data**: Boolean (raw) and categorical (cd_archivi)
   - KModes on cd_archivi

- KModes on boolean
- CLARA on boolean
- HDBSCAN on boolean
- Bernoulli Mixture Model on Boolean

2. **UMAP** with **low** minimum **distance** (2D)
   - K-Means
   - CLARA
   - HDBSCAN
3. **UMAP** with **high** minimum **distance** (2D)
   - K-Means
   - CLARA
   - HDBSCAN
4. **PCA** with **5 PCs** (around 50% explained variance)
   - K-Means
   - CLARA
   - HDBSCAN

For each input, algorithms are compared based on Clustering evaluation indexes and scatter plots.

Except for Categorical Data, the standard distance metric is Euclidean.

9.1. Categorical Data

Elbow

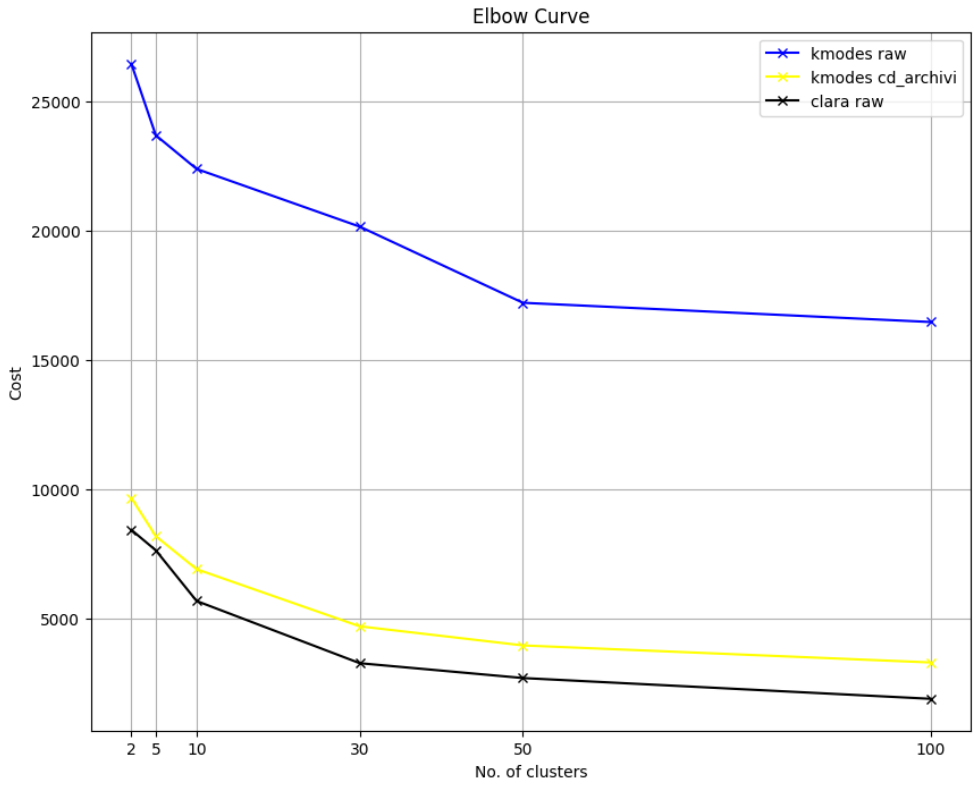


Fig 29 Elbow curve for KModes (raw and cd_archivi) and CLARA

KModes on Boolean (blue) seems to have two elbows at K = 5 and 50
KModes on cd_archivi (yellow) does not have a clear elbow
CLARA has elbows at K = 10 and 30

Silhouette

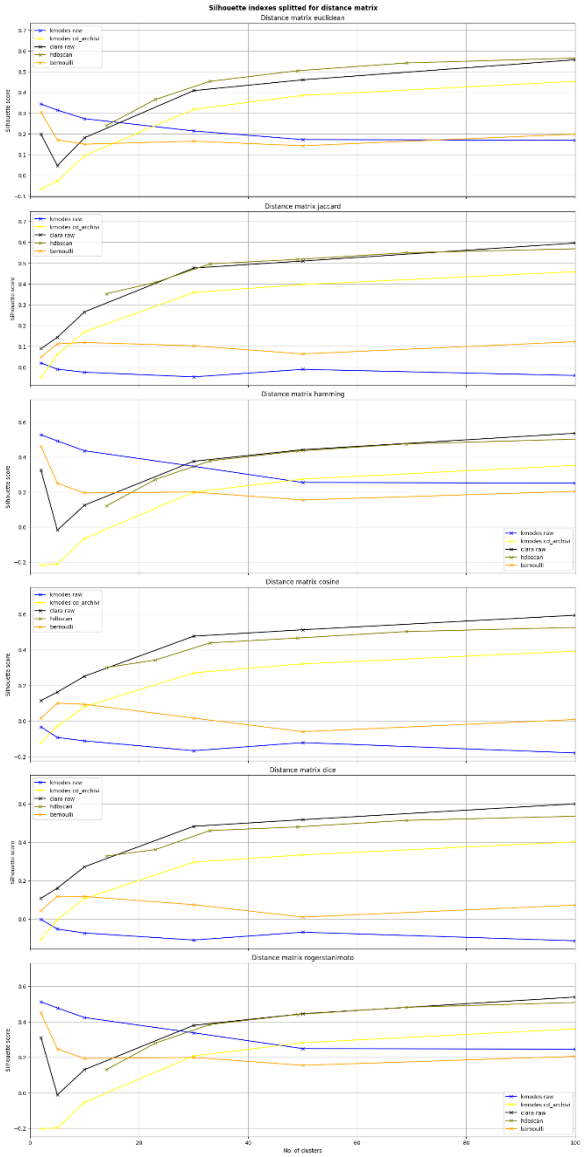


Fig 30 Silhouette with different distance matrixes

For a better view **Ctrl + click** on the image.

Algorithms are not clearly comparable since HDBSCAN requires min_ dimension as (unique) parameter instead of cluster numbers. Based on min_dimension intervals, below the clusters obtained (min_ dimension - N. clusters):

```
min_cl_5 has n. clusters = 153
min_cl_15 has n. clusters = 69
min_cl_30 has n. clusters = 49
min_cl_50 has n. clusters = 33
min_cl_100 has n. clusters = 23
min_cl_200 has n. clusters = 14
```

With different distance matrix techniques, lines change a bit but, in a general way, there is not a best one. Only **Bernoulli** seems to have a spike with Jaccard, Cosine and Dice at K = **5** and **10**.

Silhouette on boolean data frame with models' outputs (trained on boolean data frame) is pretty useless.

Pseudo-F - The Calinski-Harabasz Index

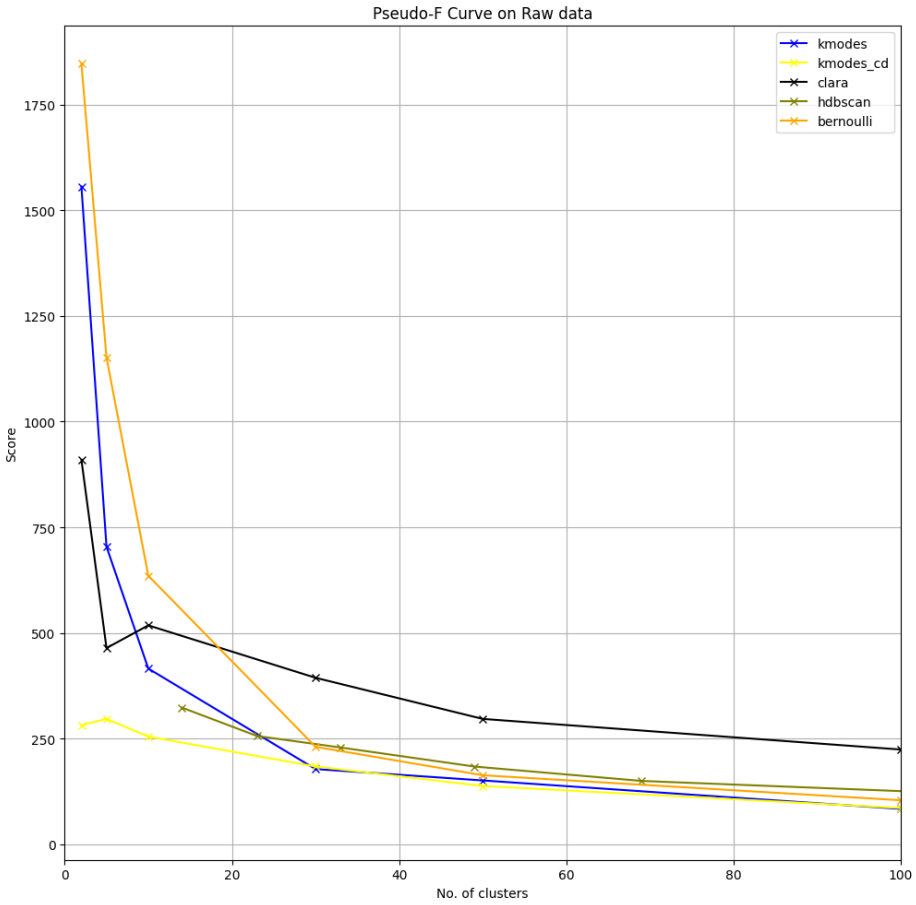


Fig 31 Pseudo-F on different algorithms

By keeping in mind that to higher values correspond a better clustering, **kmodes cd_archivi** and **clara** have a pike at, respectively, 5 and 10 clusters.

Davies-Bouldin Index

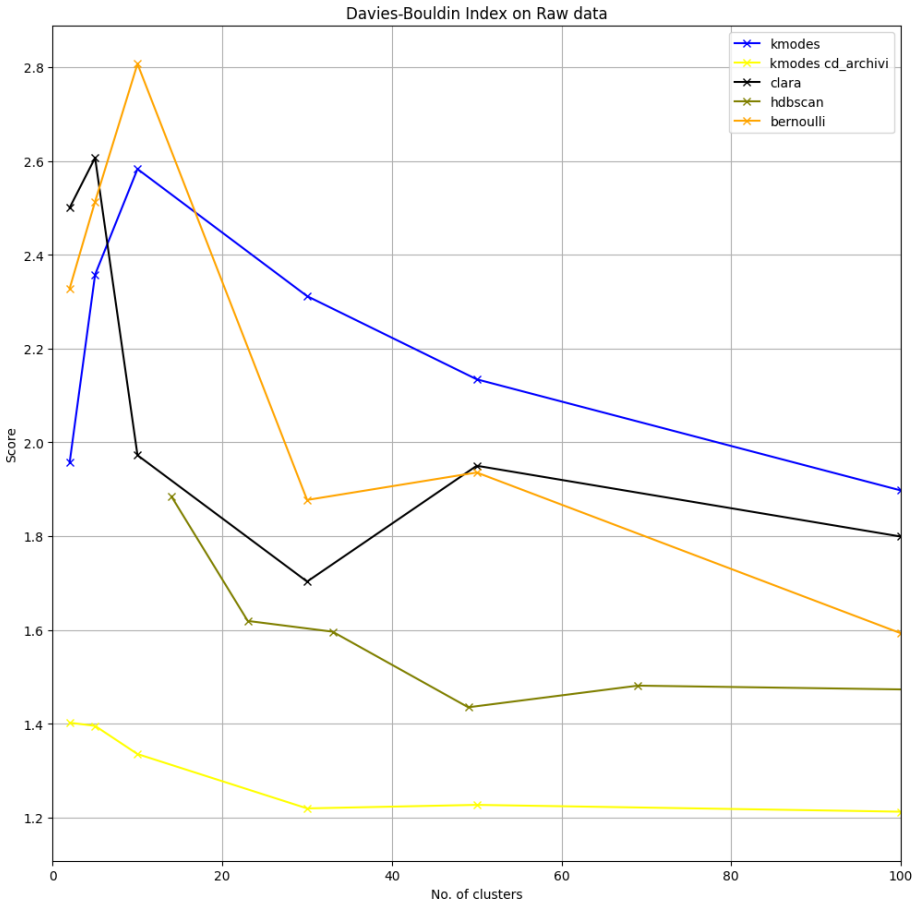


Fig 32 Davies-Bouldin Index on several algorithms

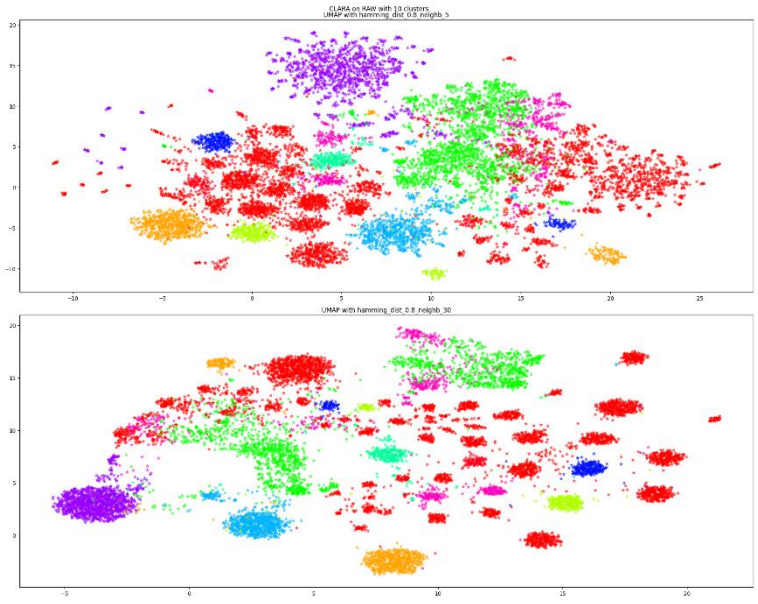
As opposite to Pseudo-F graph, a better clustering is pointed out by lower values.

Clara, **Bernoulli** and **Kmodes cd_archivi** have a very good point at K=30 and **HDBSCAN** at 33 clusters (min_dimension = 30)

Conclusions

Indexes	KModes	KModes cd_archivi	CLARA	HDBSCAN	Bernoulli MM
Elbow	5, 50		10, 30		
Silhouette					5, 10
Pseudo-F		5	10		
Davies-Bouldin		30	30	33	30

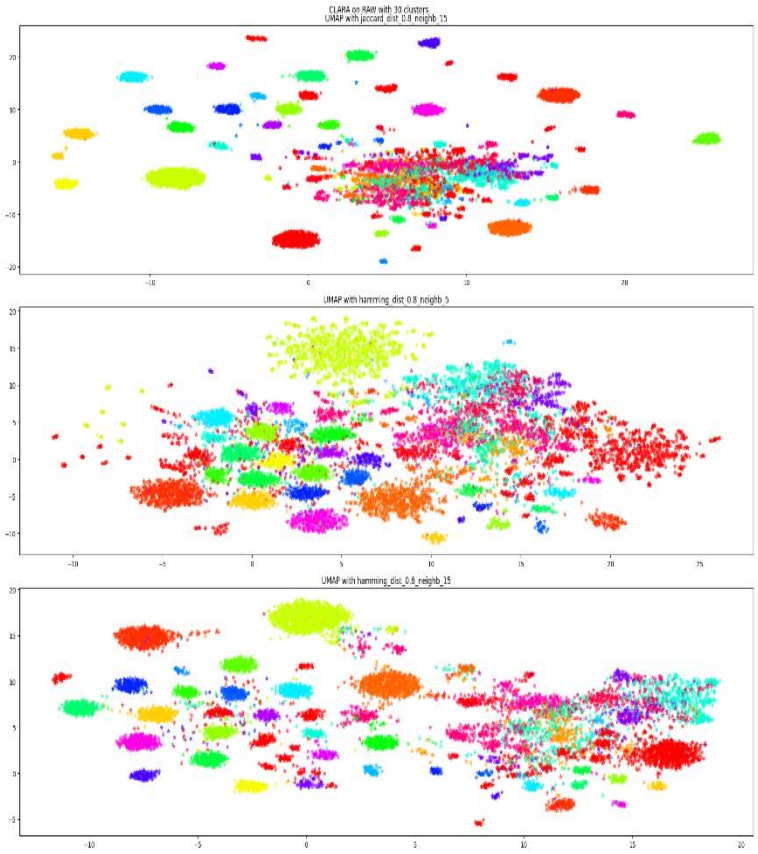
Those are the best numbers of clusters for each algorithm.
Choosing CLARA, below some of the best plots on UMAP data frame.



10 clusters with CLARA

For a better view **Ctrl + click** on the image.

This plot is made with UMAP 2D and Hamming as distance metrics. There is clearly an abundance of red points- a lot of vectors are in the same cluster-.



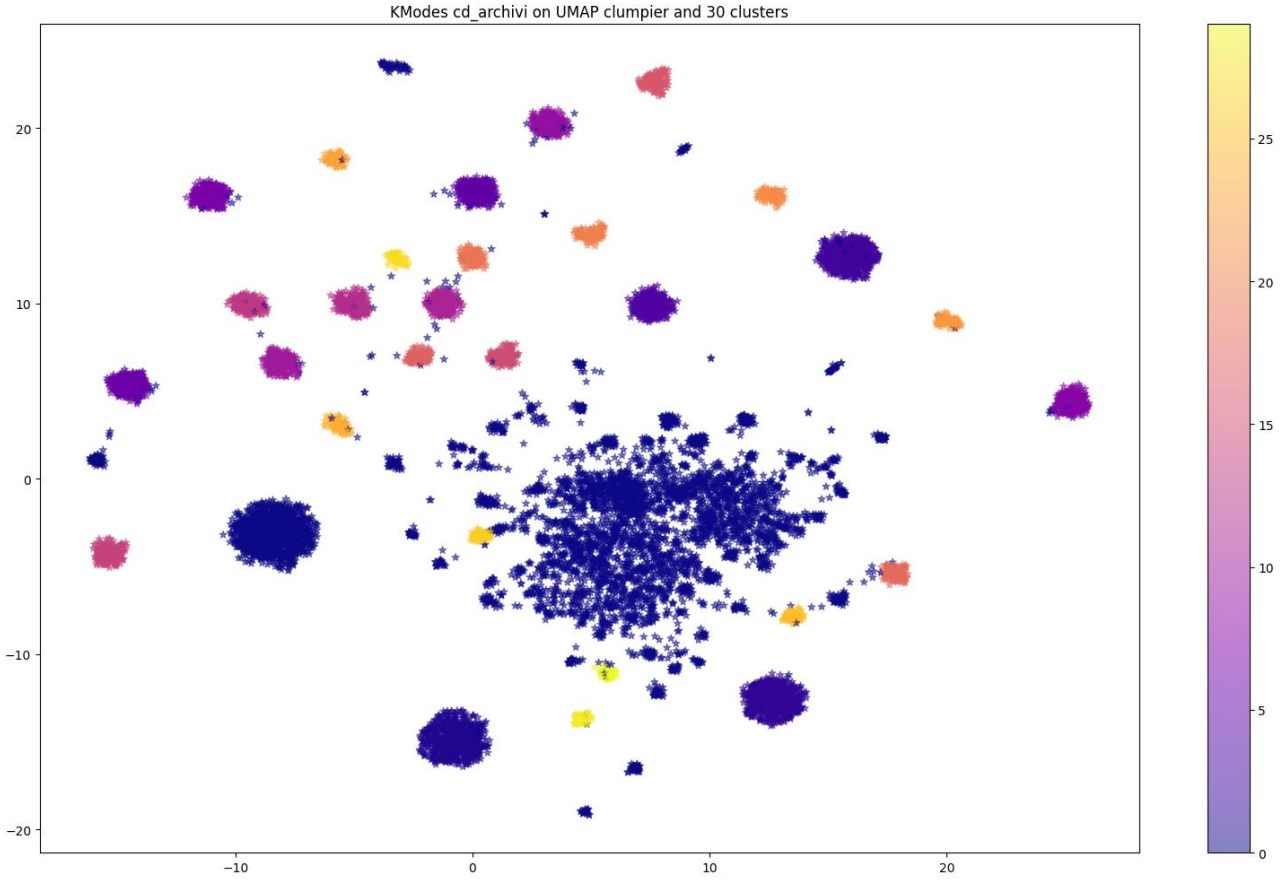
30 clusters with CLARA

For a better view **Ctrl + click** on the image.

This plot, as the one before, is still made with UMAP 2D but distance metrics are either Jaccard and Hamming.

As always, CLARA recognize high-density areas very well and cluster them right. Where things get mess is in "not constant" density-areas, clearly visible in the center of the first plot of three.

Another well-looking clustering is done by KModes (30 clusters) that, based on evaluation indexes, does not perform so well but, in concrete, here it is:

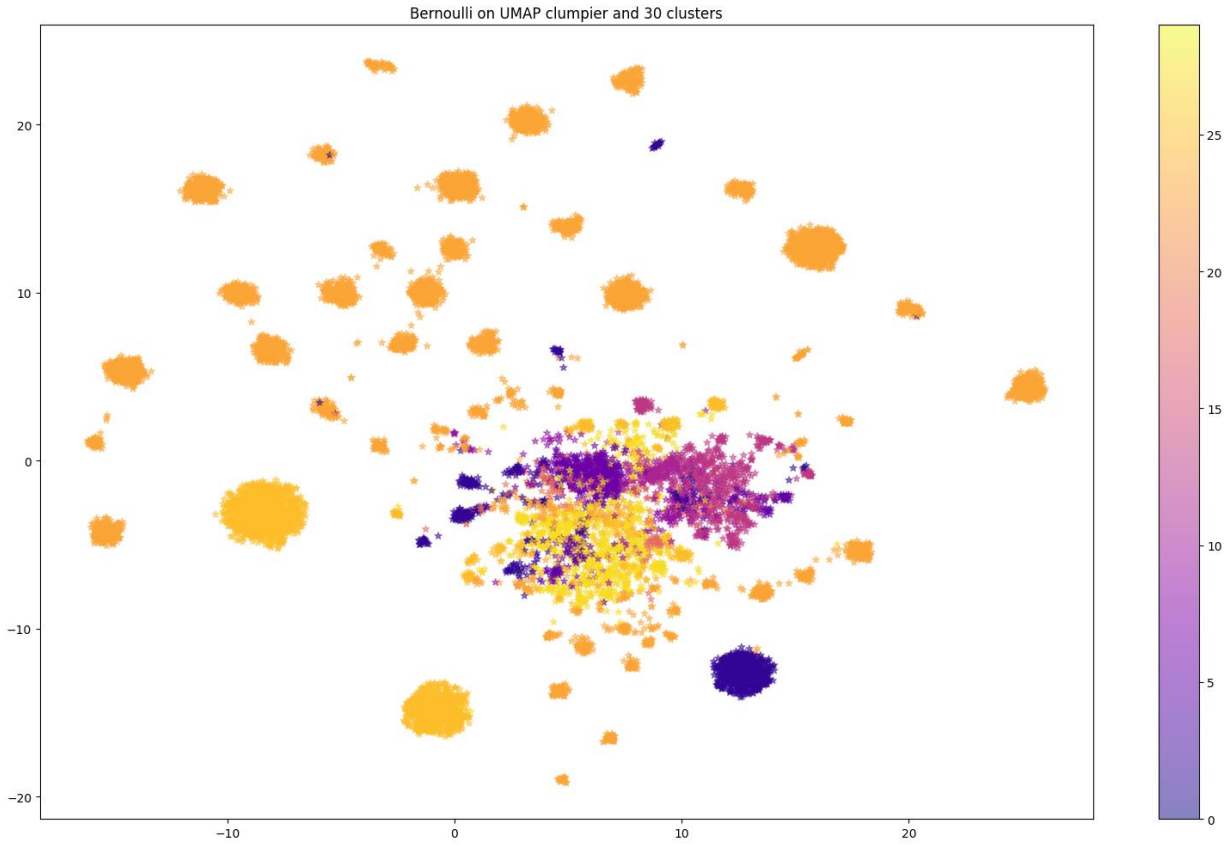


By a visual analysis of clusters plotted on UMAP with Jaccard (dist 0.8, neighbors 15), the model has achieved very good results since each clump is a well define by a color. Only blue cluster is too much present but, it could be brought back to poor K range.

Where other models struggle, KModes seems to choose the simplest and cleanest way by putting everybody into one cluster. The center of the plot has some density chaos with points near but not enough tight to define a group or, some very small groups. In general, the blue cluster seems a waste-container where, every point in doubt, is put in.

Just for information, must be keep in mind that KModes, here, is trained on categorical data (column cd_archivi) and has never seen the UMAP data frame.

For fairness, a plot of clusters from Bernoulli Mixture Model:



Bernoulli Mixture model clustering with k=30

Clustering performances for this technique are not excellent, but some considerations are due.

Bernoulli Mixture Model requires a pre-defined matrix $m \cdot n$ of probabilities where each row is a cluster and on columns the features (archives). Each value into the matrix is generated randomly (example table below).

		archive				
		2	73	...	156	800
cluster	1	0.01	0.89	...	0.56	0.11
	2	0.73	0.25	...	0.4	0.15
	3	0.12	0.04	...	0.9	0.02
	...	...	...	...	...	...
	n	p(2)	p(73)	...	p(156)	p(800)

Together, another matrix is generated with users on rows and clusters on columns. Each value into the matrix is the probability of a user to belong to a cluster (example table below).

		cluster				
		1	2	3	...	n
user	RSS12345	0.78	0.02	0.56	...	p(n)
	NRI67890	0.73	0.25	0.01	...	p(n)
	DSD15645	0.12	0.56	0.9	...	p(n)
	RCC13243	0.1	0	0.05	...	p(n)
	...	...	...	...	...	p(n)

This algorithm could perform better if combined with centroids (clustering). The problem resides in random numbers (0<x<1) generated for clusters probabilities. If, instead of randomness, the matrix is created whit centroids (float) or medoids (original boolean), Bernoulli could show more interesting conclusions.

9.2. UMAP Low Min Distance

Choosing a good UMAP data frame is not so easy since there is not a clear graphical representation of original data (raw / boolean data frame) from which to compare.

In this chapter, the focus is on data frames who caught clumpier clusters instead of broad structure (explained here), so with **n_neighbors** and **min_dist** with **lower** values.

UMAP, in this abstract, is executed with Hamming and Jaccard.

The data frame chosen is this one:

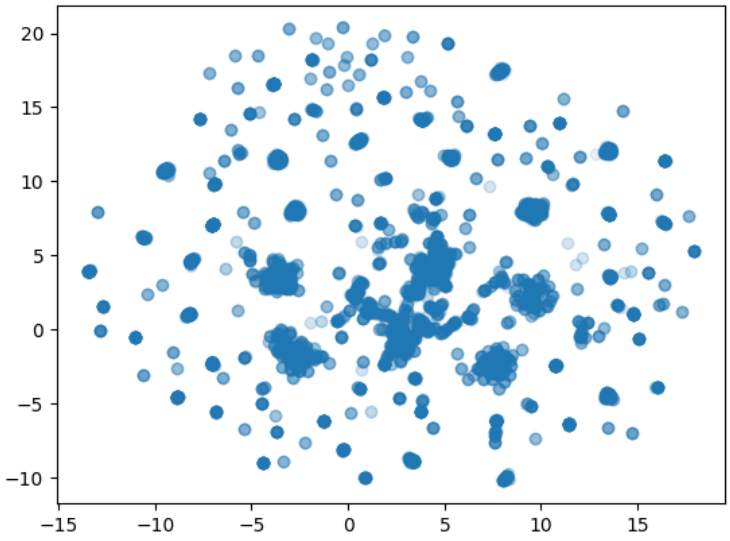


Fig 33 UMAP 2D with Jaccard (min_dist 0.0125 and n_neighbors 5)

Elbow

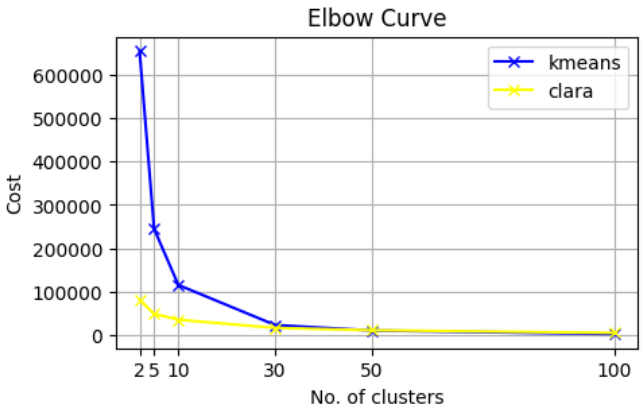


Fig 34 Elbow for kmeans and clara

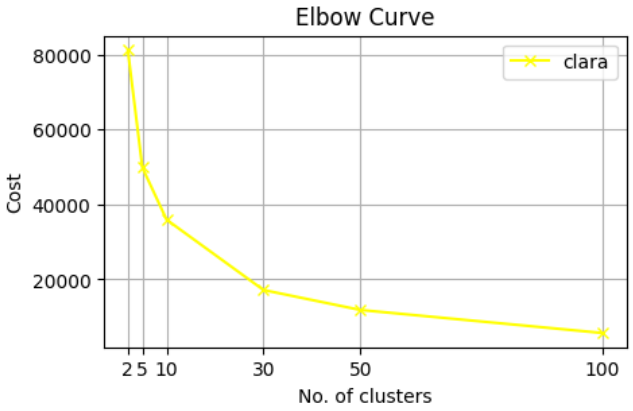


Fig 35 Elbow with focus on clara

In conclusion, for clara no clear elbow, for **kmeans** elbow at **10** and **30**

Silhouette

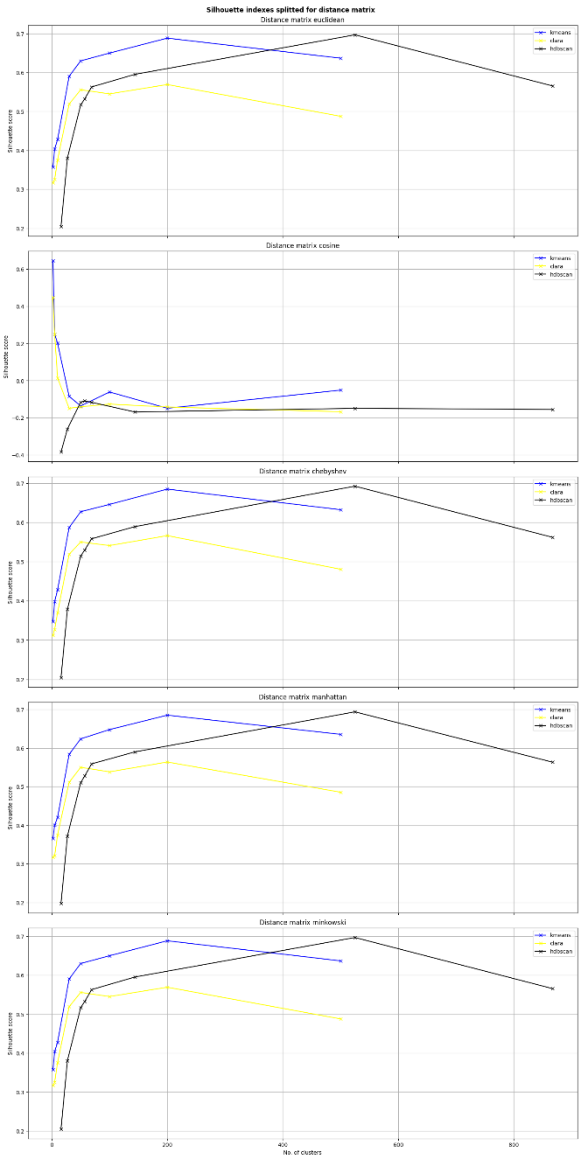


Fig 36 Silhouette with various distance metrics

For a better view **Ctrl + click** on the image.

The only different plot is the second one: cosine distance, all the others are the same. Probably due to limits of the scikit-learn version.

By analyzing first two plots, **HDBSCAN** has optimal k around **525**, **kmeans** around **200** and **clara** at **50** and **200**.
While silhouette with cosine as distance metrics seems to work bad because with increasing clusters numerosity scores decrease, except for **HDBSCAN** with maximum around **50/57** groups.

Pseudo-F - The Calinski-Harabasz Index

For this index there are very different scores, a single plot for model is required.

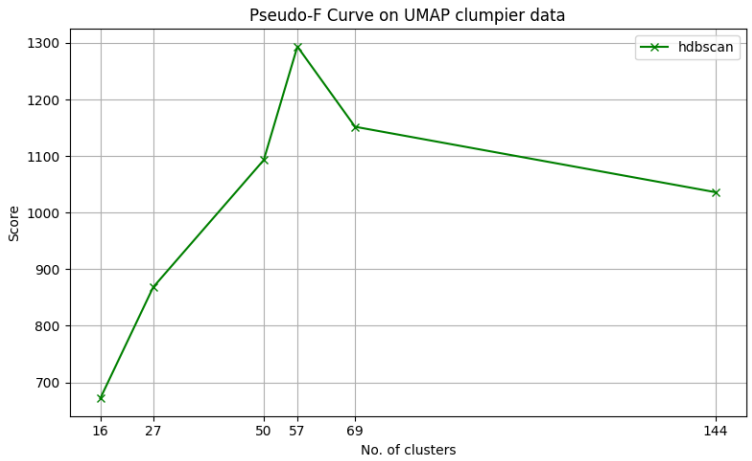


Fig 37 Pseudo-F HDBSCAN

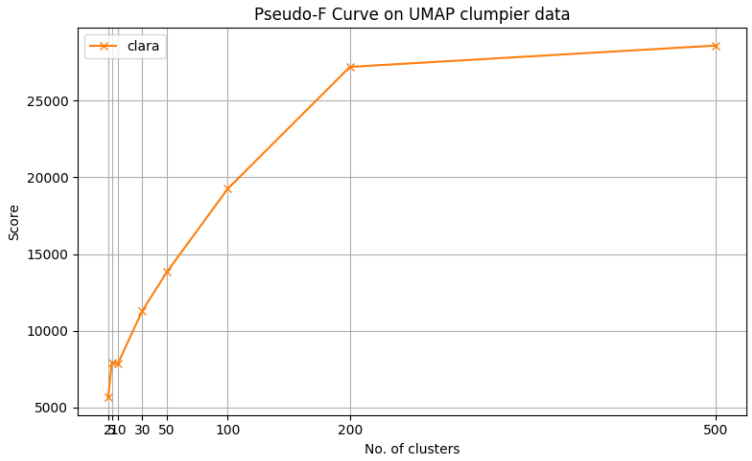


Fig 38 Pseudo-F CLARA

Pseudo-F for Kmeans grows for each bigger K so it is useless.

From those two graphs, **CLARA** as a 'good' point at **200** and **HDBSCAN** a clear maximum at **57**

Davies-Bouldin Index

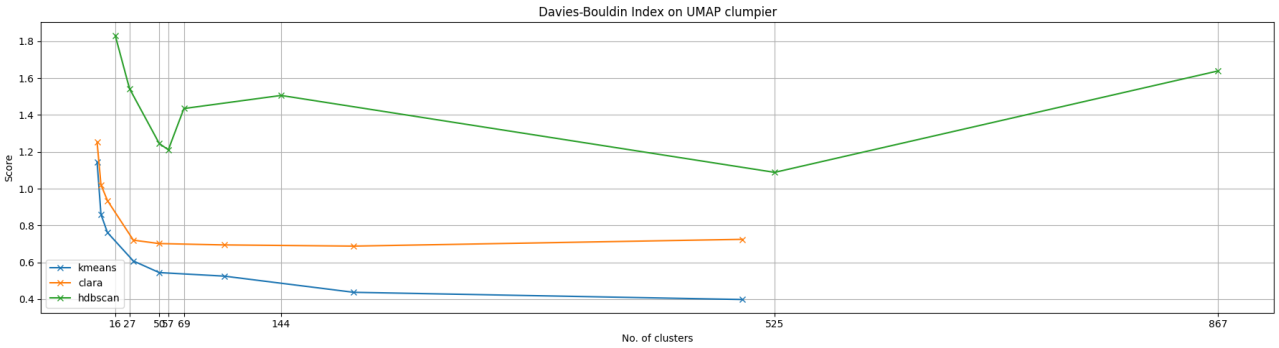


Fig 39 Davies-Bouldin index on UMAP clumpier

HDBSCAN is the only one with two local minimums at **57** and **525.**

Conclusions

Indexes	KMeans	CLARA	HDBSCAN
Elbow	10, 30		
Silhouette	200	50, 200	50, 57, 525
Pseudo-F		200	57
Davies-Bouldin			57, 525

There is clearly a hint about HDBSCAN with 57 cluster, obtained by setting min_dimension = 40. By working with tighter range around 40 can be found the best one but, for the purposes of this study (i.e. showing potentiality of algorithms and evaluations methods), deeper studies cannot be done.

For clarity, this is how of HDBSCAN's clusters appears on input data frame:

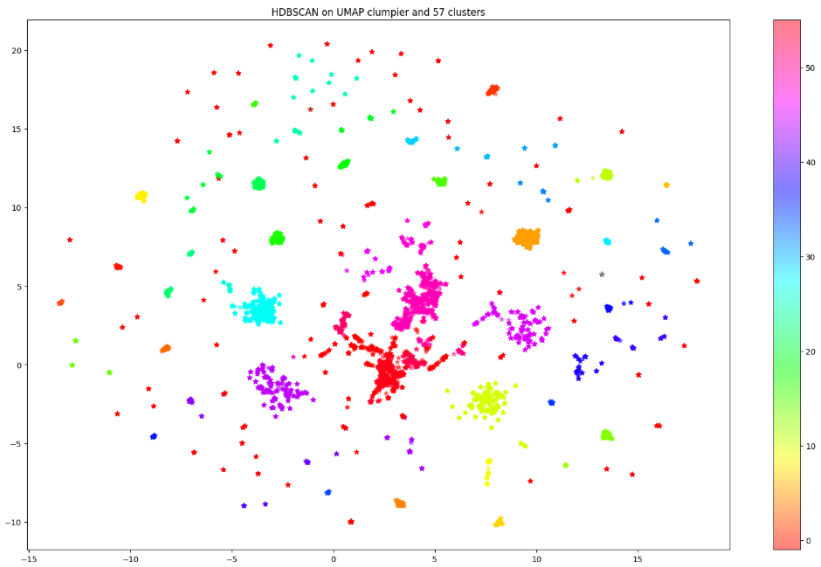


Fig 40 Best performing HDBSCAN model

For a better view **Ctrl + click** on the image.

9.3. UMAP High Min Distance

As mentioned in previous chapter, also in this section choosing a good 'simplification' of original data is not so simple. To be coherent, UMAP with Jaccard distance metrics is the focus. Below the 2D representation of chosen one:

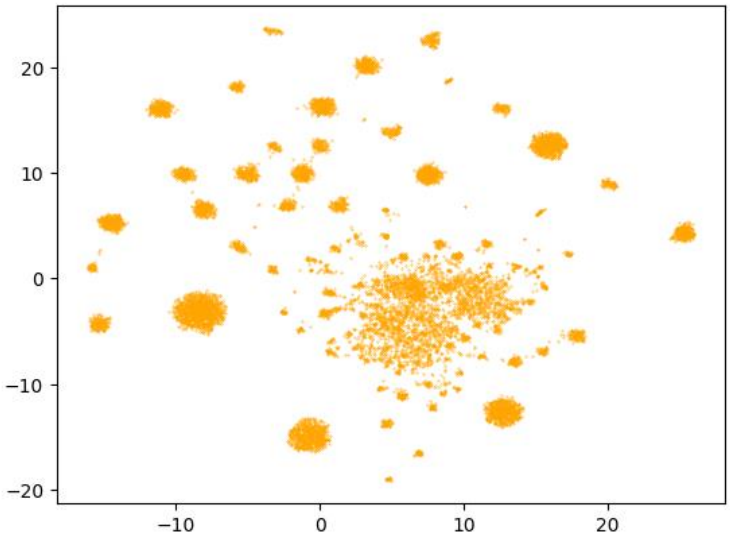


Fig 41 UMAP with Jaccard. Min_dist 0.8 and n_neighbors 15

Elbow

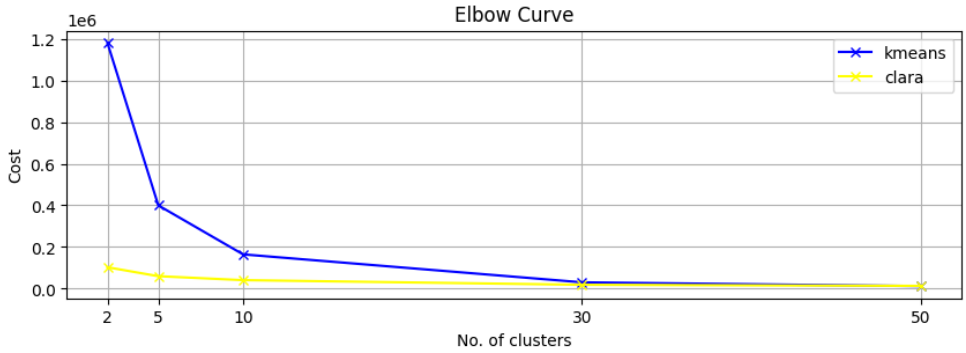


Fig 42 Elbow with kmeans and clara on UMAP broader

About **KMeans** there is an elbow at K = **10**, instead nothing useful about CLARA.

Silhouette

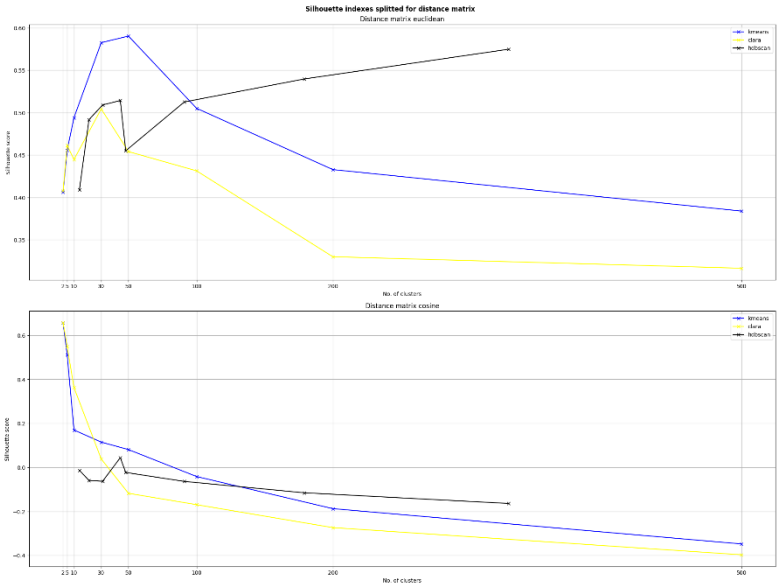


Fig 43 Silhouette for KMeans, CLARA and HDBSCAN on UMAP data frame (with Jaccard). HDBSCAN cluster list (not showed in x-axis) is [14, 21, 31, 44, 48, 91, 179, 329]

For a better view **Ctrl + click** on the image.

With Euclidean distance metric for Silhouette score, **KMeans** has a higher point at K=**50**, **CLARA** at **30** and **HDBSCAN** at **44**.

While, with Cosine distance metric, KMeans and CLARA poorly perform but **HDBSCAN** seems to have (even here) a maximum at **44**.

Pseudo-F - The Calinski-Harabasz Index

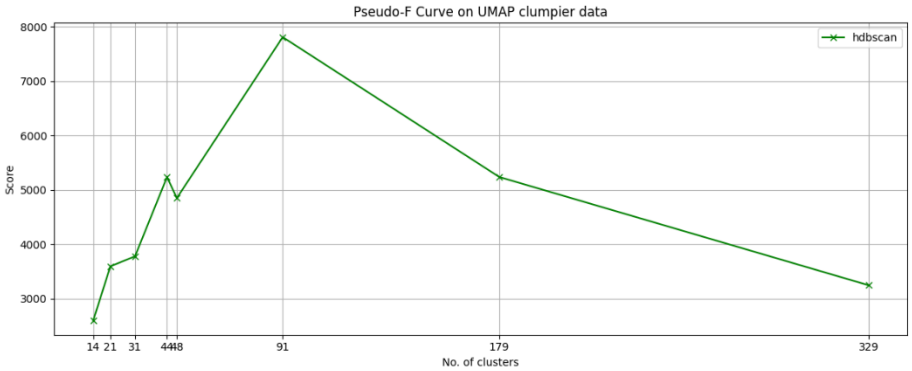


Fig 44 CH index for HDBSCAN

The Pseudo-F index did not perform well for KMeans and CLARA. Instead, with **HDBSCAN** there is a very clear maximum at K=**91**.
Just for info, note how, at K=44, there is a local maximum.

Davies-Bouldin Index

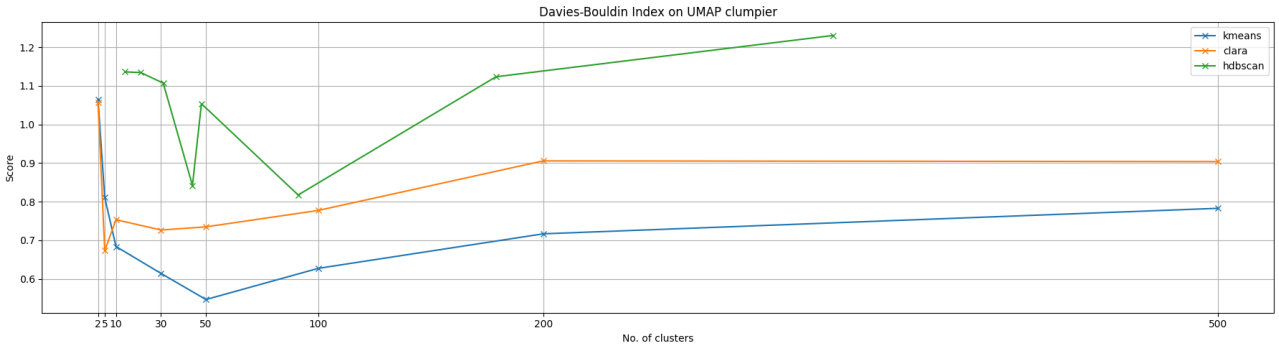


Fig 45 Davies-Bouldin index for KMeans, CLARA and HDBSCAN on UMAP broad. HDBSCAN cluster list (not showed in x-axis) is [14, 21, 31, 44, 48, 91, 179, 329]

Keeping in mind that lower score means better clustering, **KMeans** performs very well with K=**50**, **CLARA** with K=**5** and **HDBSCAN** has two local minimum points at K=**44** and **91**.

Conclusions

Indexes	KMeans	CLARA	HDBSCAN
Elbow	10		
Silhouette	50	30	44
Pseudo-F			(44), 91
Davies-Bouldin	50	5	44, 91

By studying this final tab, KMeans seems to have a good fit with K=50 and HDBSCAN two good fit with K=44 and 91 (they respectively correspond to min_dimension 40 and 15). Below a list of plots of best performing algorithms.

For all of them **Ctrl + click** on the image for a better view.

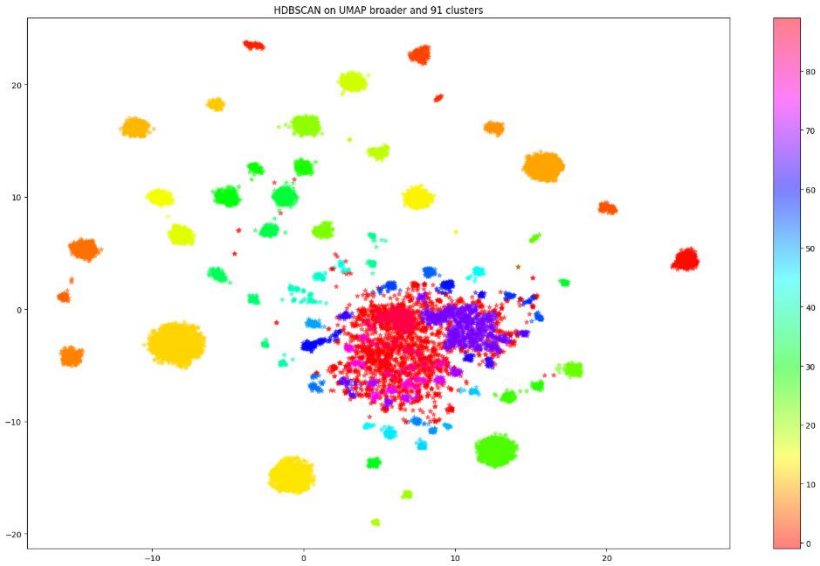


Fig 46 HDBSCAN on UMAP broad with 91 clusters

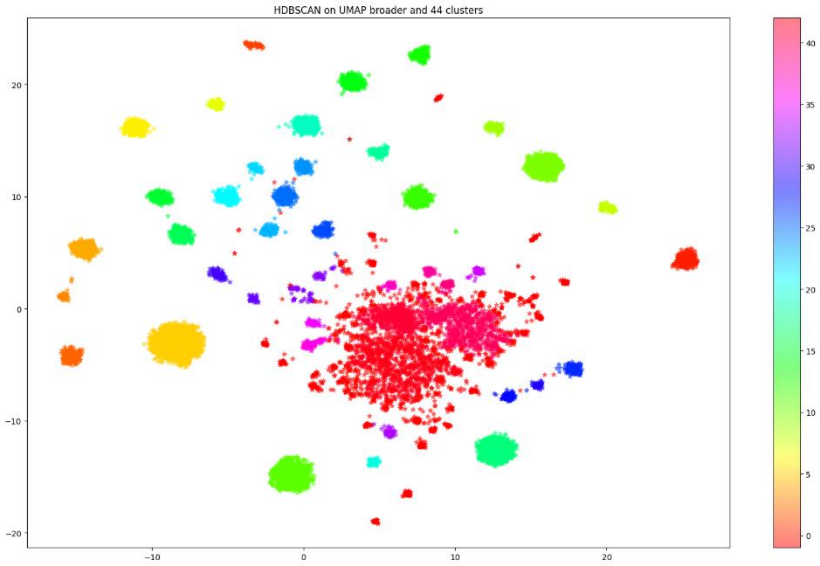


Fig 47 HDBSCAN on UMAP broad with 44 clusters

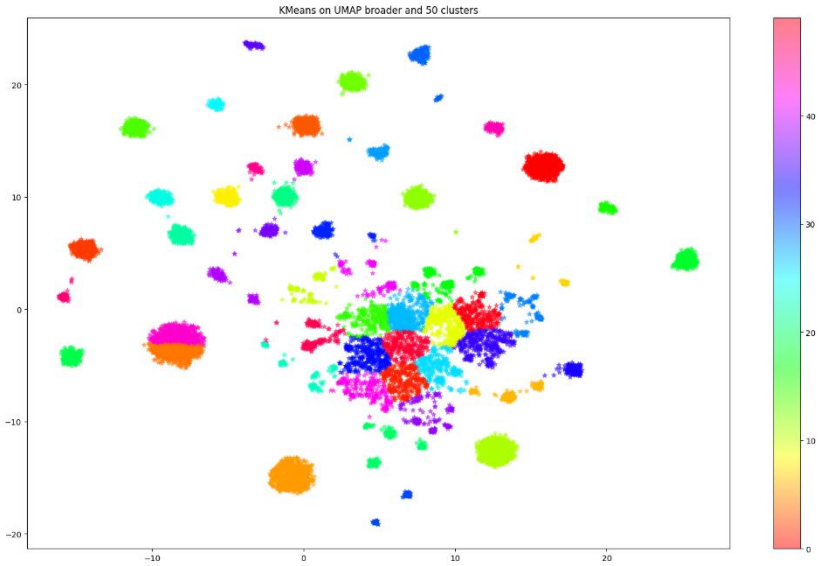


Fig 48 KMeans on UMAP broad and 50 clusters

KMeans's flaws are clearly shown here. Clusters tend to have spherical shape and centroids – or means- are not real value and (sometime) too abstract (far away from observed values) like orange cluster in south-west part of graph.

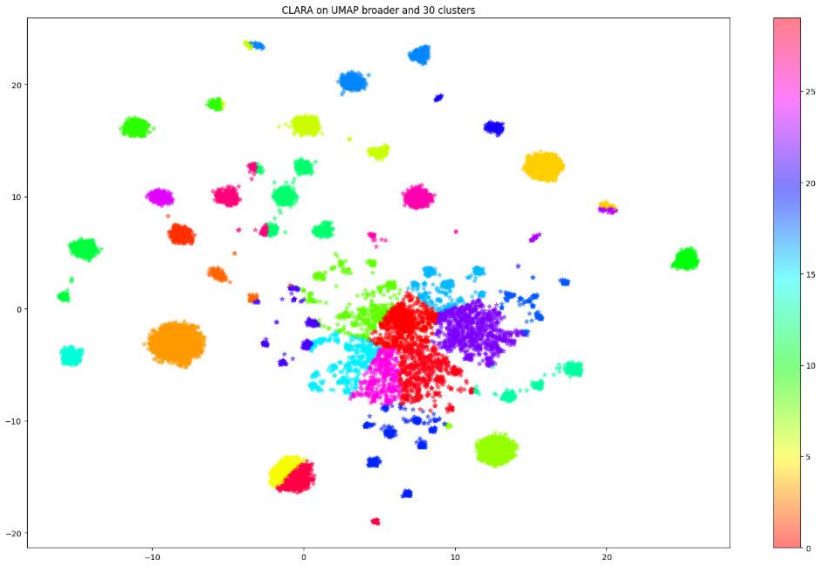


Fig 49 CLARA on UMAP broad with 30 clusters

Last two cents about CLARA algorithm. As shown in graph above (Fig 49) -and others not shown here-, there's chaos in understanding clusters from clearly visible grouped points (high density areas), since sometimes are splitted into 2 or more separated clusters when it is not right. Some examples are the clump in the south area of the plot splitted in red and yellow and more sparse and biggest clump in the center of the plot (splitted in ~7 clusters).

This phenomenon happens when pre-defined number of clusters (K) is wrong, causing algorithm to forcedly split high-density clumps into two or more clusters just to obtain the number of clusters you set it to.

In this study , the number of clusters tested for CLARA (and KMeans and KModes) are these:

```
{'clara_2': CLARA(n_clusters=2, random_state=0),
 'clara_5': CLARA(n_clusters=5, random_state=0),
 'clara_10': CLARA(n_clusters=10, random_state=0),
 'clara_30': CLARA(n_clusters=30, random_state=0),
 'clara_50': CLARA(n_clusters=50, random_state=0),
 'clara_100': CLARA(n_clusters=100, random_state=0),
 'clara_200': CLARA(n_clusters=200, random_state=0),
 'clara_500': CLARA(n_clusters=500, random_state=0)}
```

There's confidence that, with a range of K (number of clusters) incrementing of 1, could be obtained better scores.

Based on CLARA's best scores (Silhouette = 30 and Davies-Bouldin = 5), a range of K between 5 and 30 must be set with step of 1 to find the perfect tuning.

9.4. PCA with 5 PCs

Elbow

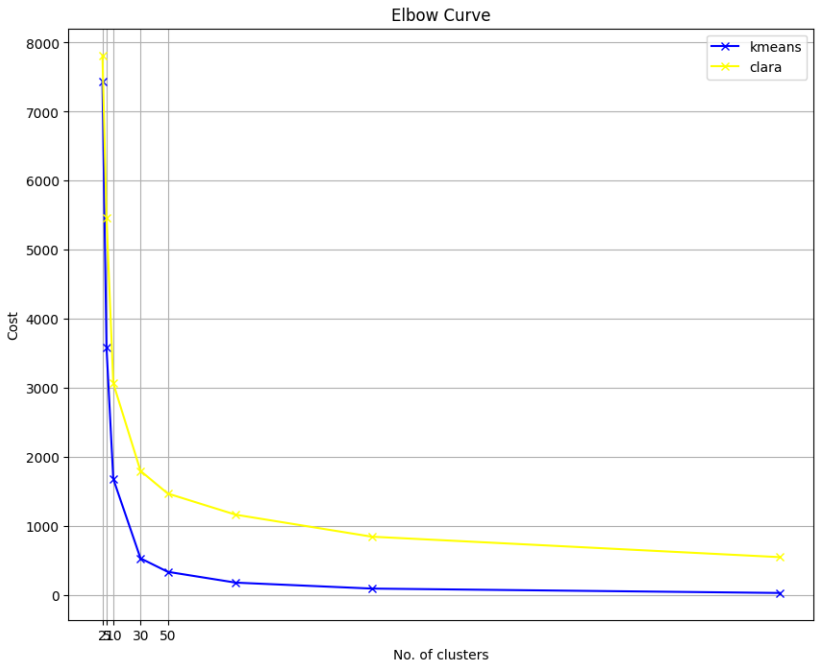


Fig 50 Elbow for KMeans and CLARA on PCA (5 principal components)

In general, there are not clear elbow points. For **KMeans** seems to be a change of gradient at K=**30**

Silhouette

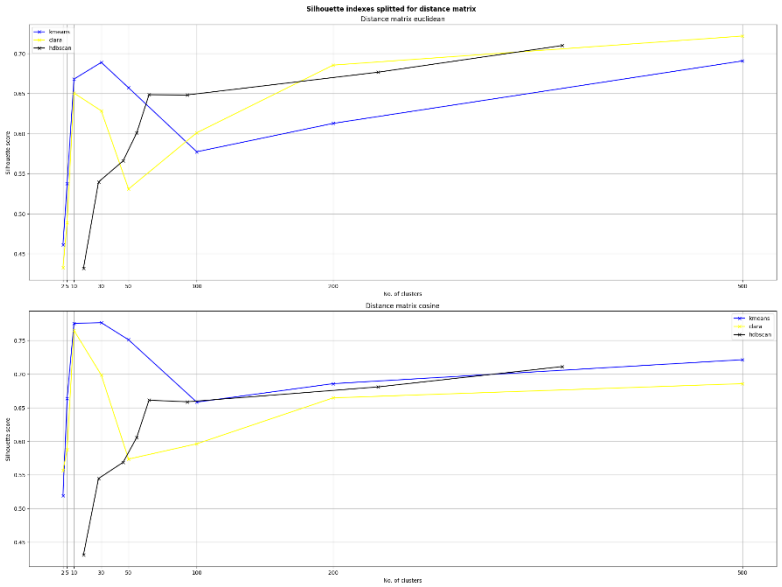


Fig 51 Silhouette with KMeans, CLARA and HDBSCAN on PCA with 5 components

For a better view **Ctrl + click** on the image.

For each algorithm for each distance metric their optimal points.

With Euclidean, **KMeans** has the best K in **30**, **CLARA** has K = **10** and **HDBSCAN** has K = **65**.

With cosine, **KMeans** and **CLARA** have the best K in **10**, and **HDBSCAN** has K = **65**.

Pseudo-F - The Calinski-Harabasz Index

Pseudo-F presents varying intervals so, a separate plot is required.

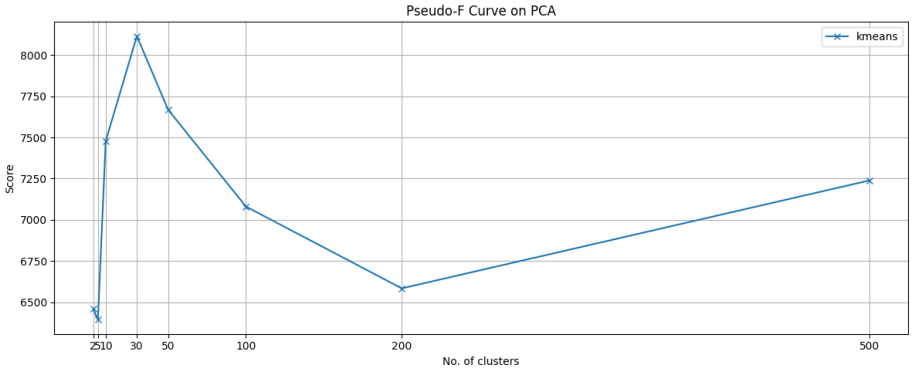


Fig 52 Pseudo-F with Kmeans

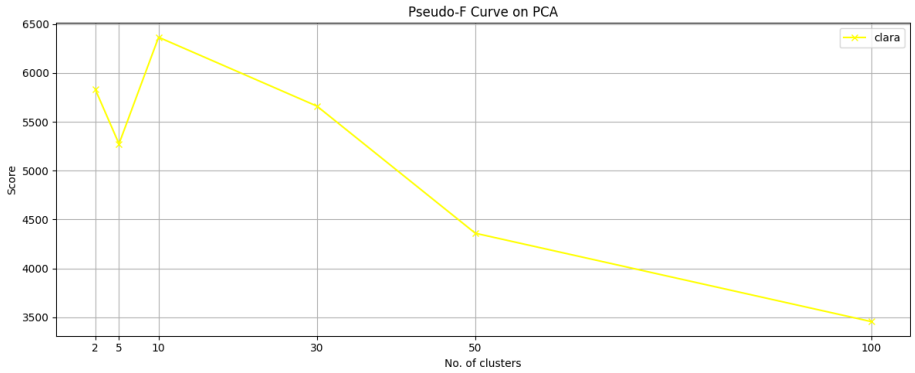


Fig 53 Pseudo-F on CLARA

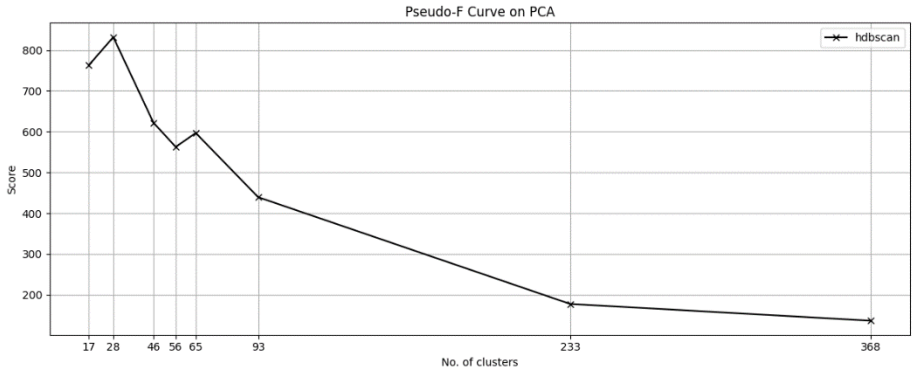


Fig 54 Pseudo-F on HDBSCAN

In conclusion, optimal Ks are **KMeans** = **30**, **CLARA** = **10**, **HDBSCAN** = **28** and **65**

Davies-Bouldin Index

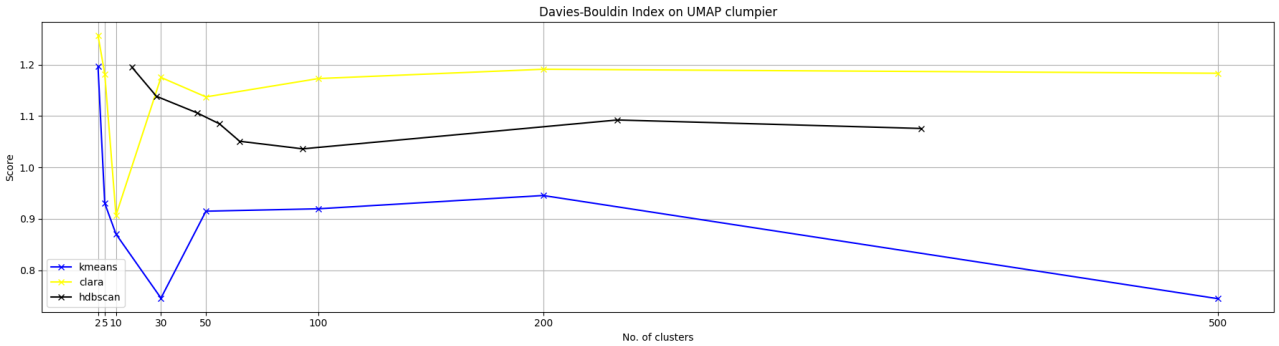


Fig 55 Davies-Bouldin index

With the last index best Ks are: **KMeans** = **30**, **CLARA** = **10** and **50**, **HDBSCAN** = **91**.

Conclusions

Indexes	KMeans	CLARA	HDBSCAN
Elbow	30		
Silhouette	10, 30	10	65
Pseudo-F	30	10	28, 65
Davies-Bouldin	30	10, 50	91

KMeans seems to have a better fit at K=30 since each index scores have enlightened it. Even CLARA seems to perform good with K=10. HDBSCAN is not really clear, especially if compared to the other best Ks, 65 is way bigger than 10 or 30.

Let us see how they appear. Below, two plots of PCA data frame on which algorithms have been trained.

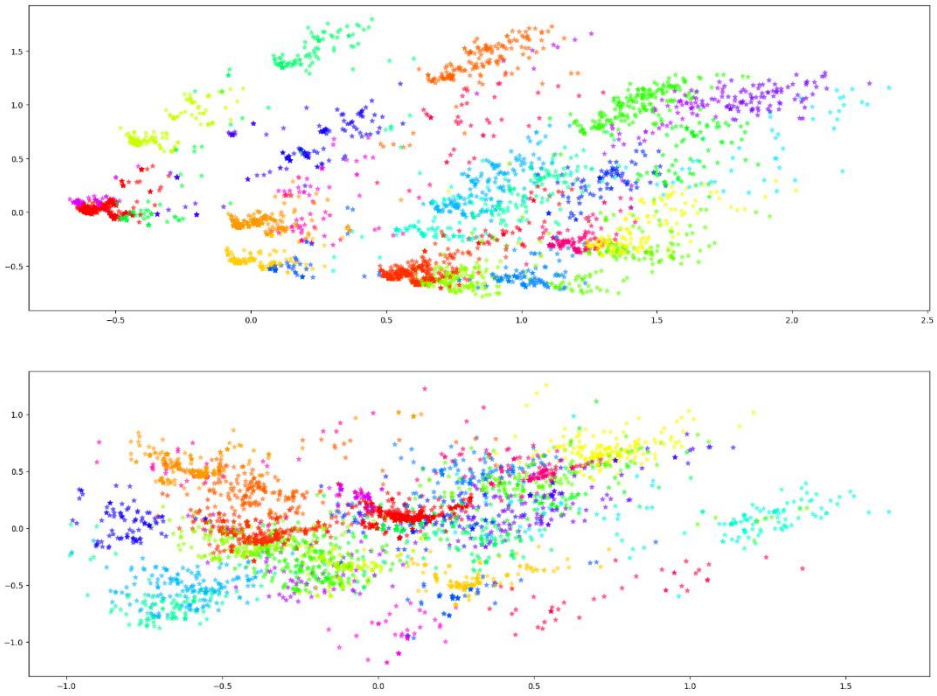


Fig 56 Scatter plot of KMeans K=30 on PCA components.
First plot is PC 0 and PC 1,
second plot is PC 2 and PC 3

Let us see how plots look on UMAP transformation:

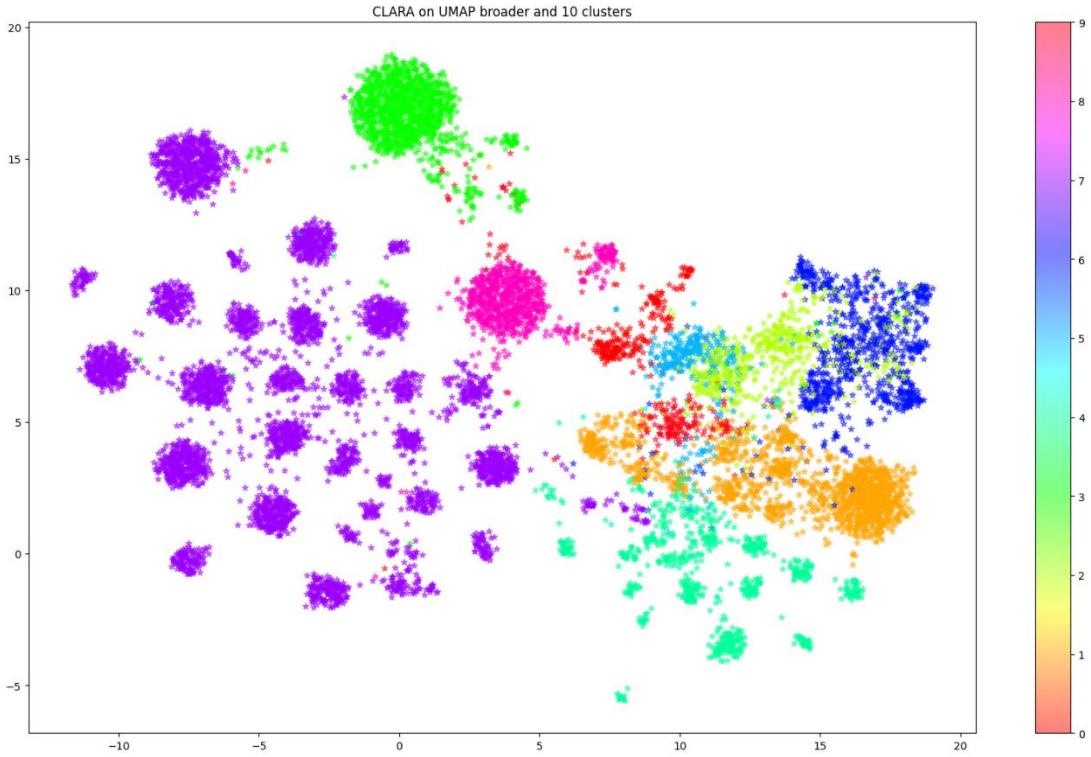


Fig 57 CLARA K=10 on UMAP

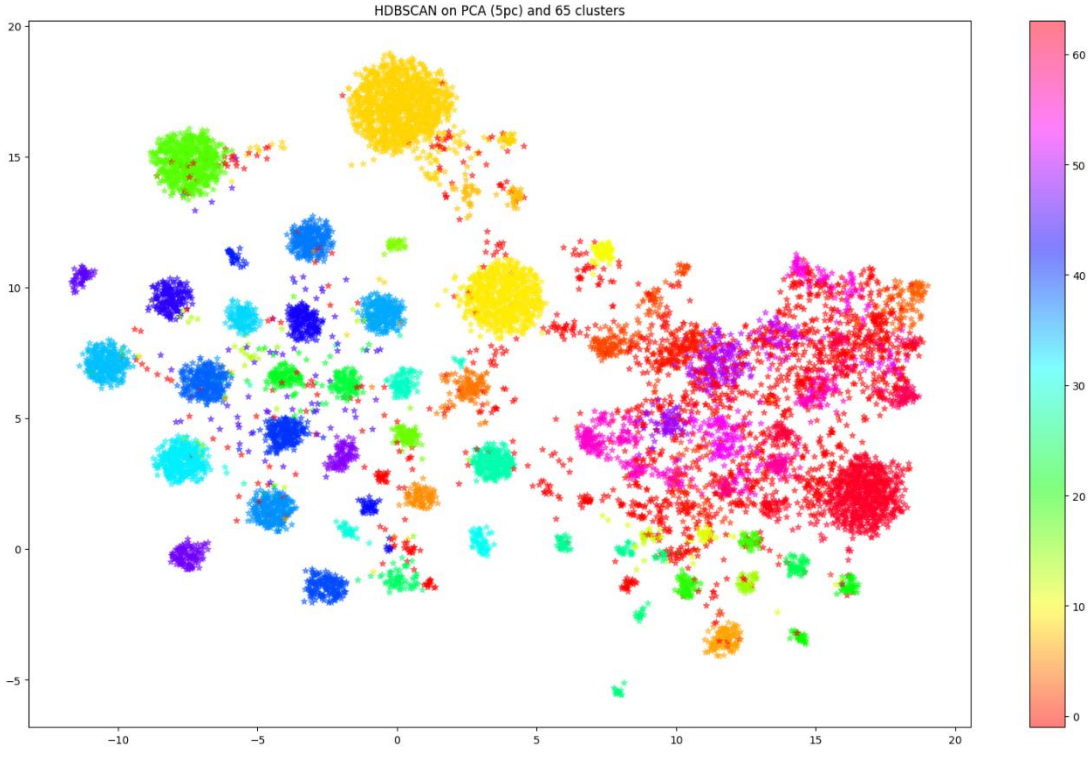


Fig 58 HDBSCAN K=65 (min_dist=30) on UMAP

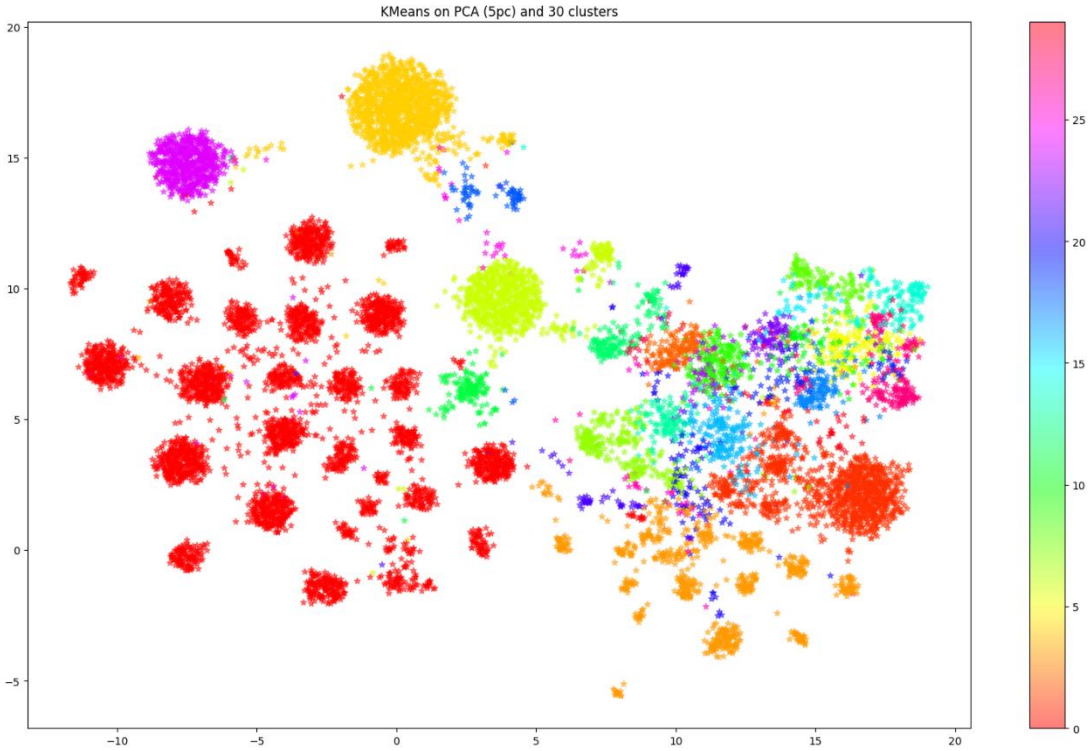


Fig 159 KMeans on PCA with 30 clusters

10. Conclusions

Being in an unsupervised learning, the desired result is unknown. Furthermore, the input dataset is boolean type with more than eighty dimensions so, a visual representation is impossible.

In chapter 5 are presented some techniques to reduce dimensions and, more importantly, to show data on a two/three dimensions plot thus, to see how vectors are.
Several approaches have been tried and almost each one has at least one well performing model.

In summary:

- On Boolean (raw) data, CLARA has best scores.
- On UMAP 2D low minimum distance (clumpier embeddings), HDBSCAN with its 57 clusters has a very good fit.
- On UMAP 2D high minimum distance (broad topological structure), HDBSCAN- again- has best results.
- On PCA with ~50% of explained variance, KMeans and CLARA work very good.

Defining the best Data Transformation technique is not feasible with dichotomic variables. The unique selection doable is by looking at plots on cartesian axes and choosing the well-looking one based on how "clusterizable" it seems.

Another (possible) hint to find and define how good is the match between data transformation technique and clustering model, is by looking at scores of the various indexes proposed in chapter 8.
For example, with original dataset (boolean/raw), KModes (on raw data) has very poor results with silhouette, pseudo-F and Davies-Bouldin; it does the opposite of what is desired. This means that KModes and boolean data is not a "good duo". The problem is resolved if, instead of boolean data, KModes is fitted on categorical data (cd_archivi).