
UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN INFORMATICA

# STUDIO E ANALISI DELL'ALGORITMO CIRCLECLUSTERING

RELATORE: Prof. Giuliano Grossi

TESI DI LAUREA DI: Morgan Malavasi

MATRICOLA: 960552

ANNO ACCADEMICO 2022-2023

# Indice


**1 Il Clustering** **3**

1.1 Dati . . . 4
1.2 Caratteristiche di un buon algoritmo . . . 4
1.3 Rassegna degli algoritmi di Clustering . . . 6
1.3.1 Kmeans . . . 6
1.3.2 Dbscan . . . 7
1.3.3 Optics . . . 9
1.3.4 Spectral Clustering . . . 10
1.3.5 Agglomerative Clustering . . . 12
1.3.6 Meanshift . . . 13
1.3.7 Affinity Propagation . . . 15
1.3.8 Birch . . . 17

**2 CircleClustering** **19**

2.1 Introduzione . . . 19
2.2 Riassunto del funzionamento dell'algoritmo . . . 20
2.3 Prima fase . . . 22
2.3.1 Algoritmo . . . 26
2.4 Seconda fase . . . 27
2.4.1 Preprocessing . . . 34
2.4.2 Albero gerarchico pesato . . . 39
2.5 Terza fase . . . 48
2.5.1 Il modello . . . 48
2.5.2 Stima dei parametri . . . 49
2.6 Il problema della matrice di similarità . . . 52
2.6.1 Utilizzo dell'accelerazione hardware . . . 53

**3 Simulazioni numeriche** **55**

3.1 Benchmark Suite . . . 55
3.2 Divisione in classi . . . 61
3.2.1 Adjusted Rand index (ARI) . . . 62

3.2.2 Adjusted Mutual Information Score (AMI) . . . . . 63
3.2.3 Risultati sperimentali . . . . . 66
3.2.4 Risultati globali . . . . . 148
3.3 Scalabilità . . . . . 151
3.4 Numero di cluster corretti . . . . . 157

# Introduzione

In questa tesi viene introdotto e analizzato un nuovo algoritmo di clustering non supervisionato, chiamato Circle-Clustering, nome che deriva dall'uso esclusivo del cerchio unitario impiegato per giungere alla partizione finale dell'insieme di dati in ingresso. I vantaggi che questo approccio mostra sono essenzialmente due: l'uso di uno spazio monodimensionale, come il cerchio unitario, in luogo a generici sottospazi normalmente di grandi dimensioni in cui giacciono i dati stessi e la possibilità di inferire agevolmente il numero di cluster di cui il dataset è composto.

L'algoritmo impiega una matrice delle distanze (o similarità) ottenuta dai dati originali secondo una metrica predefinita e associa ad ogni elemento dell'insieme originale una variabile scalare, identificata unicamente dall'angolo, che assume così valori sul cerchio unitario. A partire da un'assegnazione iniziale a dette variabili (spesso casuale) si attiva un algoritmo di ottimizzazione basato su ricerca locale, cioè un sistema iterativo autonomo a tempo discreto con dinamica asincrona, che dopo un numero finito di iterazioni raggiunge un punto sufficientemente vicino ad un minimo locale di una funzione costo che misura il grado di separazione dei punti sul cerchio combinati opportunamente con la matrice delle distanze dei dati originali. La conseguenza diretta della disposizione dei punti all'equilibrio su archi di cerchio normalmente disgiunti tra loro, consente di determinare una partizione dei dati originali chiamati in campo attraverso le loro distanze reciproche combinate, come detto, nella funzione costo. Questa partizione viene ottenuta con una euristica basata sugli alberi gerarchici il cui scopo è quello di inferire il numero k di cluster presenti nell'insieme. Il processo termina con un'operazione di labeling di gruppi di angoli adiacenti sull'intervallo reale di ampiezza $2\pi$ radianti (isomorfi cioè ai punti distribuiti sul cerchio) mediante una mistura di k gaussiane derivate da fitting della distribuzione degli angoli così ottenuti.

La prima parte del lavoro è consistita nello sviluppo dell'algoritmo con il linguaggio Python sia nella versione procedurale per CPU sia nella versione accelerata/parallela per computazioni su GPU, allo scopo di affrontare anche istanze con un numero elevato dei dati da clusterizzare. Nella seconda parte si è dato vita a una campagna massiccia di test di confronto su benchmark di riferimento con i principali algoritmi di clustering noti in letteratura e tradizionalmente implementati efficientemente in framework di machine learning come scikit-learn sviluppato in Python.

Gli algoritmi utilizzati nella comparazione possono essere divisi in tre macrocategorie: tecniche che richiedono in ingresso il parametro del numero di cluster, come KMeans, Spectral Clustering, Agglomerative Clustering e Birch; tecniche che richiedono altri tipi di parametri, come soglie o distanze, per es. DBscan, Optics e Affinity Propagation; e tecniche che fanno inferenza totale sui dati, come MeanShift. Le metri-

che di scoring quale Adjusted Rand Index (ARI) e Adjusted Mutual Information Score (AMI) hanno permesso di confrontare l'accuratezza dei risultati prodotti dai diversi algoritmi.

Nei risultati sperimentali, la tecnica proposta si è mostrata efficace, superando molto spesso i risultati prodotti da tutti gli altri algoritmi analizzati che non richiedono in ingresso il numero di cluster. In particolare, Circle-Clustering è risultato vincente l'83% delle volte su MeanShift, suo diretto rivale, 72% delle volte rispetto DBscan, 87% delle volte rispetto Optics e 79% delle volte rispetto Affinity Propagation. L'algoritmo ha dato risultati accettabili anche rispetto ad algoritmi che richiedono il numero di cluster in ingresso, con valori in media tra il 30% e il 50% di score vincenti. Per quanto riguarda la predizione del numero di cluster presenti nei dataset, Circle-Clustering ha avuto prestazioni nettamente superiori rispetto a tutte le altre tecniche, in particolare superando MeanShift il 100% delle volte. Infine, sul piano dei tempi computazionali dell'implementazione parallela, la tecnica accelerata ha mantenuto un alto grado di prestazioni in tempo su dataset di grandi dimensioni, dove l'implementazione in GPU ha superato i tempi computazionali della versione CPU di oltre 40 volte (speedup > 40).

# Capitolo 1

# Il Clustering

Dato un insieme di oggetti da analizzare, il processo di divisione in sottoinsiemi di elementi tra loro simili è denominato Clustering. In particolare, con il termine "Clustering" si intende un insieme di tecniche nell'ambito dell'apprendimento non supervisionato, con l'obiettivo di partizionare un insieme di elementi in gruppi omogenei e/o ben separati. L'operazione di clustering, nella vita di tutti i giorni, ha un'infinità di applicazioni e viene messa in pratica dalla nostra mente ogni volta che si realizza un qualche tipo di raggruppamento. Fin dalla tenera età, il nostro sistema impara a distinguere gli oggetti tramite la divisione per gruppi, migliorando continuamente i propri schemi di classificazione e riconoscimento. Perciò, ogni volta che si hanno a disposizione dei dati, risulta naturale come primo approccio quello di aggregare oggetti con le medesime caratteristiche. Questo ci aiuta a comprendere la loro natura e struttura. Se per un essere umano, dividere le informazioni tramite l'osservazione è un compito piuttosto semplice ed intuitivo, questo non lo è per la macchina, la quale deve utilizzare un algoritmo oppure un'euristica per ottenere una buona scomposizione.

Dunque, dato un insieme di dati, descritti da un insieme di attributi, trovare un insieme di cluster vuol dire trovare dei raggruppamenti in cui oggetti appartenenti allo stesso cluster sono simili tra loro, ma dissimili da oggetti appartenenti a cluster differenti.

## 1.1 Dati

Dato un insieme di n oggetti da analizzare, gli algoritmi di clustering solitamente operano su una struttura dati quale:

- Una matrice di dati. Tale matrice rappresenta il dataset iniziale delle osservazioni. E' composta da un numero di righe uguale al numero di campioni presenti nel dataset e un numero di colonne pari al numero di variabili di ciascun dato (chiamate anche attributi, features o dimensioni). Ad esempio, dato un oggetto *fungo*, alcune sue features potrebbero essere *altezza, colore, forma, profumo*.

$$A_{n\times p} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{np} \end{bmatrix}$$

- Una matrice di similarità/dissimilarità. Questa memorizza i livelli di differenza di ciascuna coppia di oggetti nel dataset. Essa è rappresentata il più delle volte da una matrice *n* x *n* dove ogni oggetto (caratterizzato dall'indice di riga) è confrontato con se stesso e con tutti gli altri oggetti dell'insieme iniziale da scomporre. *d(i,j)* rappresenta la differenza (dissimilarità) misurata tra gli oggetti *i* e *j*. Si noti che *d(i,j) = d(j,i)* e che *d(i,i) = 0*

$$A_{n\times n} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ d(2,1) & 0 & 0 & 0 & 0 \\ d(3,1) & d(3,2) & 0 & 0 & 0 \\ ... & ... & ... & ... & ... \\ d(n,1) & d(n,2) & ... & ... & 0 \end{bmatrix}$$

## 1.2 Caratteristiche di un buon algoritmo

- Scalabilità: la maggior parte degli algoritmi di clustering eseguono velocemente su piccoli insiemi; tuttavia, un grande dataset può avere milioni di oggetti. Sono quindi necessari algoritmi di clustering altamente scalabili, che siano in grado di produrre buoni risultati in tempi brevi.

- individuazione di cluster con diverse forme: diversi algoritmi determinano i cluster basandosi sulle misure comuni di distanza, quali ad esempio quella Euclidea. Tuttavia, l'utilizzo di queste metriche portano al riconoscimento o alla costruzione di cluster di forma sferica. Un cluster nel mondo reale può assume-

re qualsiasi forma, perciò è importante pensare a strategie che siano in grado di analizzare cluster di forma arbitraria.

- Conoscenze pregresse: capita spesso che per funzionare, un algoritmo debba essere informato riguardo alcuni parametri del dataset, quali il numero di cluster desiderati. I parametri sono difficili da determinare e molte volte l'obiettivo dell'algoritmo è la ricerca stessa di tali parametri. Ad esempio, l'algoritmo KMeans richiede in ingresso il numero di cluster per effettuare l'operazione di partizionamento, però molte volte, prima ancora della label dei dati, un utente è interessato al numero effettivo di classi presenti nell'insieme.

- Dati rumorosi: i database nel mondo reale contengono quasi sempre dati mancanti o errati. Sono necessari algoritmi in grado di astrarre rispetto queste imperfezioni e che non portino a cluster di scarsa qualità per tal motivo.

- Clustering incrementale: la maggioranza delle tecniche di clustering non possono assimilare nuovi dati nei cluster senza dover effettuare una nuova totale computazione su tutti gli elementi. Tali sistemi sono quindi lenti rispetto a versioni incrementali, dove ogni aggiunta di un dato ha un costo fisso.

- Insensibilità all'ordinamento: Gli algoritmi non devono variare nel loro funzionamento in base all'ordine dei dati in input; lo stesso insieme di dati, in qualsiasi ordine venga esposto, dovrebbe sempre portare allo stesso output.

- Dimensionalità: Un Database può essere composto da decine, centinaia se non a volte migliaia di dimensioni (*caratteristiche*). Molti algoritmi sono in grado di gestire computazionalmente solo dataset con poche dimensioni. Gli algoritmi che sono in grado di ovviare al problema della dimensionalità e riescono a scalare adeguatamente su dataset dove ciascun oggetto presenta molte features sono pochi e difficili da realizzare.

## 1.3 Rassegna degli algoritmi di Clustering

E' facile intuire che non esiste un algoritmo in grado di classificare i dati univocamente in modo migliore. Tuttavia, gli algoritmi presenti in letteratura sono molti, ognuno con le proprie caratteristiche. La qualità dell'output di tali tecniche dipende dal tipo e dalla forma dei dati in ingresso. Alcuni modelli possono essere vantaggiosi in certe situazioni e svantaggiosi in altre. Analizziamo ora alcuni algoritmi utilizzanti tecniche tra loro anche molto differenti. E' curioso notare che le tecniche di clustering prendono molte volte ispirazione dai modelli e dai ragionamenti legati alla vita quotidiana. Gli algoritmi esaminati saranno successivamente utilizzati nel confronto con il CircleClustering.

### 1.3.1 Kmeans

Dato un insieme di $n$ oggetti e un parametro $k$, il KMeans organizza gli oggetti in $k$ partizioni con vincolo $k \leq n$, dove ogni partizione rappresenta un cluster. L'obiettivo del KMeans è ottimizzare un criterio di partizionamento oggettivo, spesso denominato funzione di similarità, in modo tale che oggetti all'interno di un cluster siano "simili" mentre gli oggetti di cluster differenti siano "dissimili".

L'algoritmo KMeans riceve in input un parametro k e partiziona un insieme di n oggetti in k cluster in modo tale che la similarità intra cluster sia alta mentre la similarità inter cluster sia bassa. Per quantificare la similarità di un oggetto rispetto al suo sotto gruppo di appartenenza si misura la distanza dell'oggetto dal valore medio degli oggetti in un cluster; tale valore può essere visto come il centro di gravità del cluster.

L'algoritmo kMeans procede nel seguente modo. Sono selezionati k oggetti appartenenti all'insieme iniziale in modo randomico; ciascuno di essi rappresenta inizialmente la media/centro di un cluster. Ciascuno degli oggetti rimanenti viene associato al cluster più simile seguento la regola della distanza euclidea tra l'oggetto e la media del cluster (possono essere utilizzate anche altre metriche quali la distanza di Manhattan). Una volta che tutti gli elementi sono stati associati al cluster di appartenenza più vicino, sono ricalcolate la medie, ovvero i centri, di ciascun cluster. Tale processo viene iterato fino a quando non si raggiunge la convergenza di una determinata funzione criterio. Tipicamente viene utilizzato l'errore quadratico o l'inerzia.

$$E = \sum_{i=1..k} \sum_{p \in C_i} |p - m_i|^2$$

$E$ è la somma dell'errore quadratico per l'insieme di oggetti del dataset, p è il punto nello spazio che rappresenta il dato elemento e $m_i$ è la media di uno specifico cluster

$C_i$. Tale criterio permette di rendere i $k$ cluster quanto più compatti e separati possibile. L'algoritmo si comporta bene quando i cluster sono originariamente sferici e piuttosto ben separati l'uno dall'altro. La complessità dell'algoritmo è O(nkt), dove n è il numero di elementi, k è il parametro rappresentante il numero di cluster e t è la quantità di iterazioni del loop. Il metodo termina spesso in un ottimo locale, ma la necessità degli utenti di specificare all'inizio k, ovvero il numero dei cluster, è un grande svantaggio. Oltre a ciò KMeans è molto sensibile al rumore e agli outlier, dal momento che un piccolo numero di interferenze possono influenzare sostanzialmente il centro di un gruppo e creare medie errate.

### 1.3.2 Dbscan

Per individuare cluster di forma arbitraria sono stati sviluppati i metodi di clustering basati sulla densità. Tali metodi considerano i cluster come regioni di elementi molto dense nello spazio dei dati separate da regioni a bassa densità che rappresentano rumore. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) è un algoritmo di clustering che costruisce ciascun cluster mettendo insieme regioni con densità sufficientemente alta; esso è in grado di individuare cluster di forma arbitraria e fornisce risultati interessanti anche in presenza di rumore. Per comprendere adeguatamente il DBSCAN sono richieste alcune nozioni fondamentali dell'algoritmo:

*Def*: Un punto $q$ è direttamente raggiungibile da un altro punto $p$ se la distanza fra essi è minore di un $\varepsilon$ fissato (cioè, se $q$ è parte del cosiddetto $\varepsilon$-vicinato di $p$; ovviamente tale relazione è simmetrica). Se nell' $\varepsilon$-vicinato di $p$ è presente un numero minimo M di punti, $p$ è detto un nucleo *(core point)*.

*Def*: Un punto $r$ è raggiungibile per densità da un nucleo $p$ se esiste una sequenza di punti $p_1, \ldots, p_n$ di punti con $p_1 = p$ e $p_n = r$ in cui ogni $p_{i+1}$ è direttamente raggiungibile da $p_i$. Ciò implica che tutti i punti intermedi siano nuclei, mentre $r$ potrebbe non esserlo (cioè potrebbe non avere almeno $M$ punti nel suo $\varepsilon$-vicinato): la relazione di raggiungibilità per densità non è perciò generalmente simmetrica. I punti non raggiungibili da alcun altro punto sono detti "esterni" *(outlier)*.

Si considera dunque come cluster di $p$ lo stesso $p$ e tutti i punti che sono raggiungibili (per densità) da esso. Ogni cluster contiene almeno un punto nucleo, e quelli che non lo sono vengono detti "margini" *(edge)*: ovvero punti da cui non possono esserne raggiunti altri. Far parte dello stesso cluster è una relazione simmetrica che può essere formalizzata tramite la nozione di connessione per densità (density-connectedness): due punti $p$ e $q$ sono connessi per densità se esiste un punto $o$ tale che sia $p$ che $q$ sono raggiungibili (per densità) da $o$.

Un cluster, che è un sotto-insieme dei punti dell'insieme di dati, soddisfa due proprietà:

- Tutti i punti all'interno del cluster sono mutuamente connessi per densità.
- Se un punto è raggiungibile per densità da un altro punto del cluster, anch'esso è parte del cluster.

DBSCAN necessita di due parametri: $\varepsilon(eps)$ e del numero minimo di punti richiesti per formare un cluster *(minPts)*. Si comincia con un punto casuale che non è stato ancora visitato. Viene calcolato il suo $\varepsilon$-vicinato e se contiene un numero sufficiente di punti viene creato un nuovo cluster. Se ciò non avviene il punto viene etichettato come rumore e successivamente potrebbe essere ritrovato in un $\varepsilon$-vicinato sufficientemente grande riconducibile ad un punto differente entrando a far parte di un cluster.

Se un punto è associato ad un cluster anche i punti del suo $\varepsilon$-vicinato sono parte del cluster. Conseguentemente tutti i punti trovati all'interno del suo $\varepsilon$-vicinato sono aggiunti al cluster, così come i loro $\varepsilon$-vicinati. Questo processo itera fino a quando non sono stati visitati tutti i punti.

DBSCAN visita ogni punto del dataset, anche più volte nel caso di punti candidati a cluster differenti, il tempo di esecuzione è pari a $O(n^2)$ nel caso peggiore. Con l'uso di strutture indicizzate, il tempo di esecuzione scende, nel caso medio, a *O(n log(n))*.

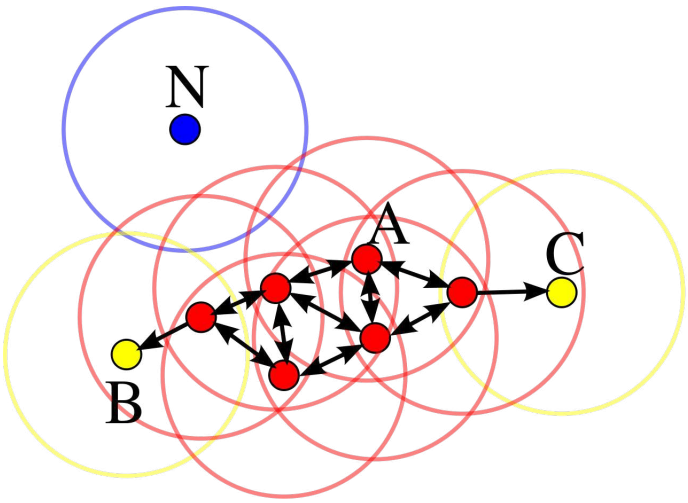



Figura 1.1: Esempio DBSCAN

In figura (1.1), il punto A e gli altri punti rossi sono *core point*. I punti B e C sono raggiungibili per densità da A e quindi essendo connessi per densità appartengono allo stesso cluster. Il punto N è un punto rumoroso, poichè non è nè un *core point*, nè è raggiungibile per densità. (minPts=3 o minPts=4)

### 1.3.3 Optics

OPTICS (Ordering Points To Identify Cluster Structure) è un altro algoritmo basato sul modello di densità. OPTICS cerca di risolvere uno dei principali punti deboli del DBSCAN: il problema di rilevare cluster significativi in dati di densità variabile. Per fare ciò, i punti del database sono ordinati (linearmente) in modo tale che i punti spazialmente più vicini diventino vicini nell'ordinamento. Inoltre, per ogni coppia di punti viene memorizzata una distanza speciale, tale misura sarà utilizzata per determinare se la coppia di punti è appartenente allo stesso cluster oppure no. Lo sviluppo dell'algoritmo e l'unione dei punti tra loro vicini produce un dendogramma, come in figura (1.3).

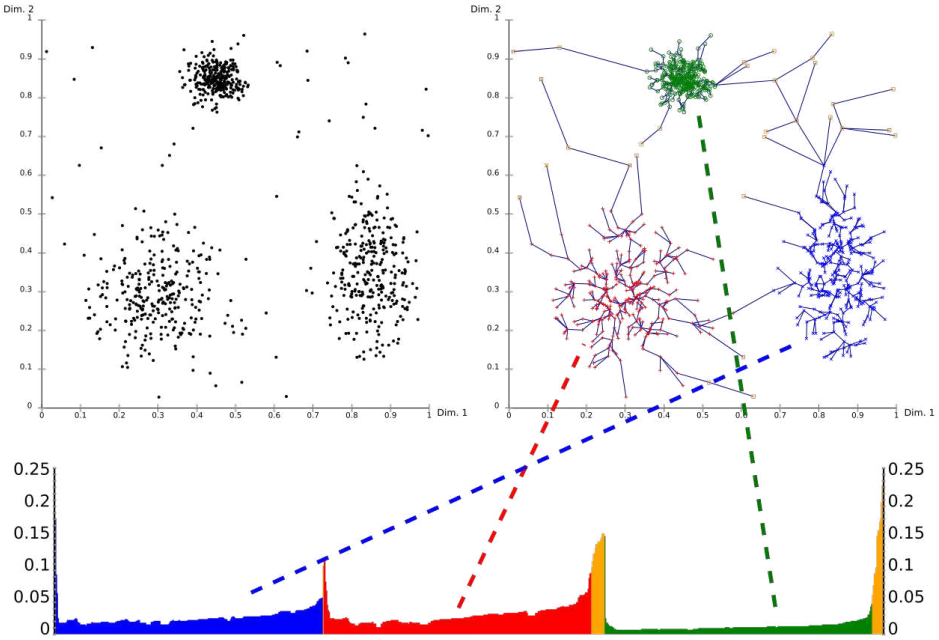



Figura 1.2: Raggio di definizione Core Point

Sono necessarie alcune nozioni oltre quelle introdotte da DBSCAN.

- Core Distance: rappresenta il valore minimo del raggio richiesto per classificare un punto come un *core point*. Se il punto non è un *core point* allora la sua *core distance* sarà "Undefined".
- Distanza di raggiungibilità: dati due punti p e q, la distanza di raggiungibilità è definita come il massimo tra la *core distance* di p e la distanza euclidea tra p e q.

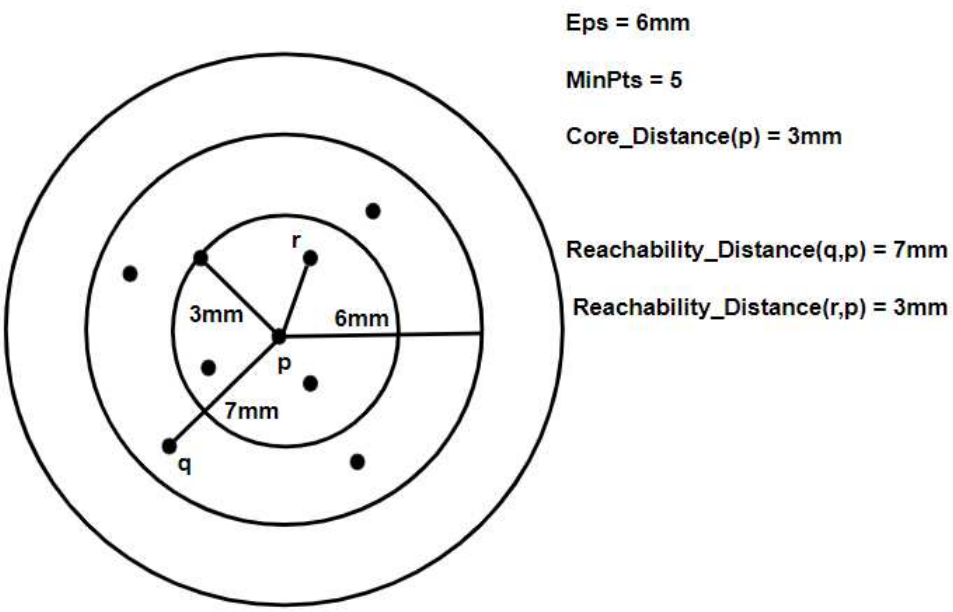



Figura 1.3: Raggio di definizione Core Point

Questa tecnica di clustering è diversa da altre tecniche, nel senso che essa non segmenta in modo esplicito i dati in cluster. Al contrario, produce una visualizzazione delle distanze di raggiungibilità e utilizza questa visualizzazione per raggruppare i dati.

### 1.3.4 Spectral Clustering

La tecnica di clustering spettrale utilizza lo spettro (gli autovalori) della matrice di similarità dei dati per eseguire una prima riduzione della dimensionalità; una volta che la dimensionalità è stata ridotta, è possibile applicare uno dei metodi di clustering, quali ad esempio kmeans o altre tecniche. La matrice di similarità consiste in una valutazione quantitativa della somiglianza relativa di ciascuna coppia di punti nel set di dati.

Dato un insieme di oggetti in input rappresentanti gli elementi di cui si vogliono ottenere i cluster, il processo di clustering spettrale può essere visto come un processo in tre fasi fondamentali:

1. Costruzione della matrice di similarità

   La matrice di similarità di un insieme di punti è una matrice in grado di misurare il grado di (dis)similarità di ciascuna coppia degli oggetti coinvolti. Se essa non è fornita in input all'algoritmo, questo la ricalcolerà.

2. Costruzione del grafo di adiacenza

E' possibile costruire un grafo a partire dai valori delle distanze. Definito un valore per l'intorno di un punto, se la distanza tra tale punto e coloro nel suo intorno è inferiore al valore scelto, significa che i due punti sono "vicini". Definiamo allora un nodo per ciascun punto e tracciamo un arco che connette ogni coppia di nodi che rispetta la regola sopra. Risulta più facile costruire ed esprimere il grafo come una matrice in cui i punti tra loro vicini sono identificati attraverso un arco di valore 1, altrimenti 0. Da tale matrice sarà possibile ricavare il relativo grafo indiretto.

3. Costruzione del grafo laplaciano

   Dato un grafo G con n vertici $v_1, \ldots, v_n$, il suo grafo laplaciano $L_{nxn}$ è una matrice Laplaciana tale per cui, per ogni punto nella matrice:

   $$L_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise,} \end{cases}$$

   O equivalentemente dalla matrice $L$

   $L = D - A$

   dove $D$ è la matrice dei gradi e $A$ è la matrice di adiacenza del grafo. La matrice dei gradi di un grafo non orientato è un particolare tipo di matrice che dispone lungo la diagonale il numero di connessioni di ogni nodo.

4. Si procede con la riduzione della dimensionalità dei campioni attraverso l'utilizzo degli autovettori della matrice Laplaciana sviluppata precedentemente.

5. Infine, ottenuti i campioni sottodimensionati, si procede alla divisione degli elementi con una delle tecniche di clustering comuni, quali ad esempio il KMeans, per ottenere le singole partizioni e le label.

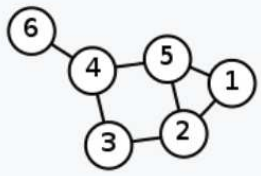
| Labelled graph | Degree matrix | Adjacency matrix | Laplacian matrix |
|---|---|---|---|
|  | $\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$ |

Figura 1.4: Processo di creazione della matrice Laplaciana

### 1.3.5 Agglomerative Clustering

Il clustering agglomerativo ricade sotto una delle famiglie più generali di algoritmi di clustering che prendono il nome di "algoritmi gerarchici". L'idea del clustering gerarchico è di creare cluster nidificati attraverso l'unione o la suddivisione di insiemi più grandi oppure più piccoli, a seconda della logica che si cerca di seguire. Solitamente il risultato di tali algoritmi è un dendogramma, ovvero un albero utilizzato per visualizzare la somiglianza nel processo di raggruppamento.

Il clustering agglomerativo lavora secondo la logica "bottom up", ovvero dal basso verso l'alto. Ogni oggetto (ogni foglia nell'albero) è inizialmente considerato come un cluster a sè stante. Ad ogni step dell'algoritmo, i due cluster che sono più simili tra loro sono combinati in un unico nuovo cluster (nodo). Questa procedura è iterata fino a quando tutti i punti sono membri di un solo grande cluster (radice).

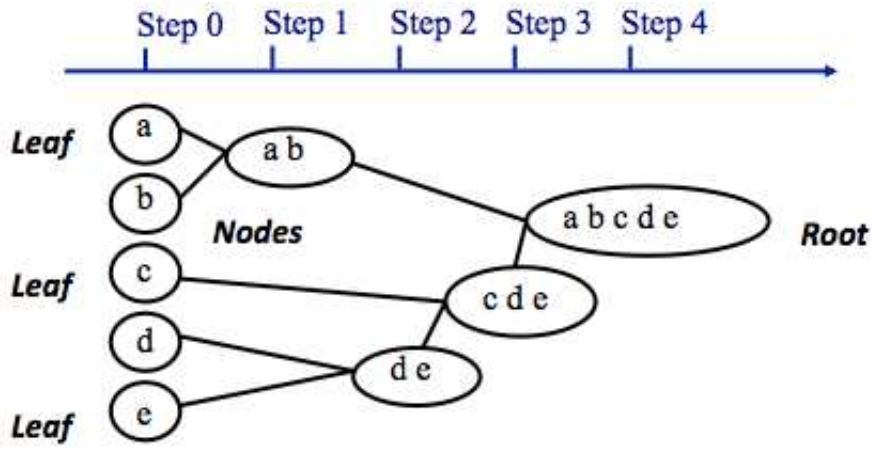



Figura 1.5: Esempio di dendogramma nel clustering agglomerativo

La tecnica può essere suddivisa in tre step principali:

1. Calcolo del valore di (dis)similarità di ciascuna coppia di oggetti all'interno della matrice di (dis)similarità.

2. Unione degli elementi in gruppi di entità sempre maggiore attraverso una funzione *"linkage"* e la rispettiva matrice di (dis)similarità (gli elementi tra loro più simili sono *"linkati"*).

3. Scelta del punto di taglio dell'albero gerarchico per ottenere il numero di cluster e il successivo labeling dei dati.

Tra le varie funzioni di *linkage*, una delle più comuni è il Ward. Diversamente da altre misure di *linkage*, il metodo di Ward afferma che la distanza tra due cluster A e B, è la quantità di aumento della somma dei quadrati, quando questi saranno uniti.

$$\Delta(A,B) = \sum_{i \in A \bigcup B} \|x_i - m_{A \bigcup B}\|^2 - \sum_{i \in A} \|x_i - m_A\|^2 - \sum_{i \in B} \|x_i - m_B\|^2$$

$$\Delta(A,B) = \frac{n_A \, n_B}{n_A + n_B} \|m_A - m_B\|^2$$

$\Delta$ è chiamato "il costo di merging dei cluster A e B". Con il clustering agglomerativo, la somma dei quadrati vale zero (ogni punto ha il proprio cluster), dopodichè cresce man mano che i cluster sono uniti. Il metodo di Ward mantiene questa crescita il più piccolo possibile.

Infine, per definire il numero ottimale di cluster, è spesso utilizzato il Calinski and Harabasz Index (CH). Questo criterio combina la varianza intra-cluster e inter-cluster per valutare la qualità della segmentazione.

$$CH(K) = \frac{B(K)/(K-1)}{W(K)/(n-K)}$$

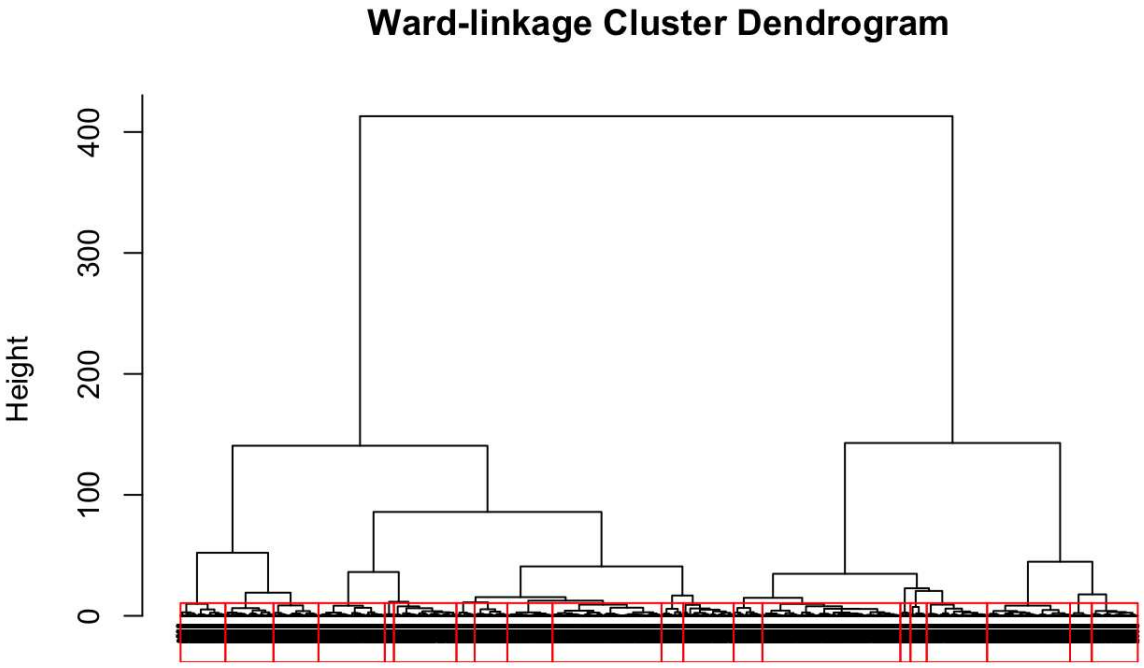



Figura 1.6: Agglomerative Clustering Ward

### 1.3.6 Meanshift

Introdotto nel 1975 da Fukunanga e Hostetler, MeanShift è un algoritmo iterativo utile per determinare i massimi locali di una funzione di densità di probabilità a partire da un dataset di campioni. Dato un insieme di punti, l'algoritmo assegna in modo itera-

tivo ciascun punto alla centroide del cluster più vicino. La direzione, ovvero la scelta della centroide corretta e della sua relativa vicinanza è determinata dalla posizione della maggior parte dei punti vicini. Quando l'algoritmo si interrompe, ogni punto viene assegnato a un cluster.

Data una funzione kernel $K$, ad ogni iterazione viene calcolata, per ogni punto $x$, la media pesata della *stima kernel di densità*.

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

$N(x)$ è l'insieme di campioni $x_i$ per i quali $K(x_i) \neq 0$. Il vettore $m(x) - x$ è detto mean shift. Ogni punto viene aggiornato con uno spostamento verso la media, nella direzione indicata dal vettore mean shift $x_i \leftarrow m(x_i)$ e il procedimento viene iterato fino alla convergenza. La funzione kernel è solitamente una funzione radiale di base. Alcune funzioni comuni sono il cerchio:

$$K(x) = \begin{cases} 1 & \text{if } \|x\| \leq \lambda \\ 0 & \text{if } \|x\| > \lambda \end{cases}$$

e la gaussiana:

$$K(x_i - x) = e^{-c||x_i - x||^2}$$

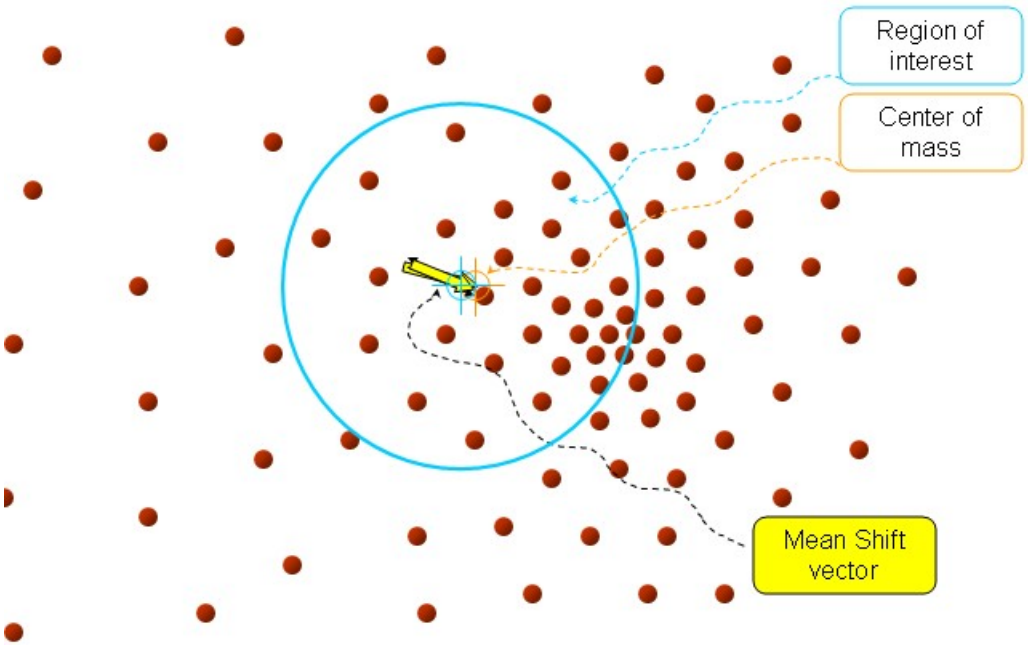



Figura 1.7: MeanShift

La seguente tecnica di clustering può essere molto costosa a livello computazionale, soprattutto per grandi dataset, poichè è necessario iterare molteplici volte ogni

punto. La complessità dell'algoritmo è $O(n^2)$.

### 1.3.7 Affinity Propagation

Affinity Propagation (AP) è un algoritmo di clustering basato sul concetto di *message passing* tra punti. Diversamente da altri algoritmi di clustering, la tecnica AP non richiede in input il numero di cluster per effettuare il calcolo e il successivo labeling. L'idea dell'algoritmo può essere mostrata attraverso un semplice esempio: Si supponga di ritrovarsi su un'isola dispersa con altre persone del tutto sconosciute. All'inizio nessuno conosce nessuno, quindi ognuno è il rappresentante di se stesso. Dopo aver conosciuto più persone e aver comunicato con gli altri, ognuno scopre che alcune persone condividono valori simili ai propri mentre con altre è difficile andare d'accordo. Quindi, ognuno tende ad avere più interazioni con coloro che sono simili a se, ed evita di stare con coloro che non gli stanno simpatici. Con l'aumento dei cicli di comunicazione, si trova finalmente il proprio "gruppo". Nella propria mente ora c'è sicuramente un candidato che può rappresentare perfettamente il proprio gruppo.

La seguente idea spiega perfettamente il concetto che sta alla base della tecnica. L'algoritmo usa le comunicazioni tra punti per trovare esemplari per ogni punto. I punti che condividono lo stesso esemplare saranno assegnati allo stesso gruppo (cluster).

Per descrivere il funzionamento della tecnica sono necessarie tre matrici.

1. Matrice di similarità

   L'algoritmo crea inizialmente la matrice di similarità con cui vengono espresse le vicinanze tra i vari punti. Intuitivamente, più grande sarà il valore di similarità, tanto più vicini saranno tra di loro i punti. Solitamente è utilizzata la negativa della distanza euclidea per tale scopo. Qui, *s(i,k)*, ovvero la similarità tra i punti $i$ e $k$, è definita:

   $$s(i,k) = -\|x_i - x_k\|^2$$

   Nell'algoritmo di affinity propagation, tuttavia, i valori della diagonale della matrice di similarità *s(i,i)* non sono settati semplicemente a zero, ma assumono un significato specifico. I punti con un valore maggiore della diagonale, ovvero con valori più grandi di *s(i,i)* sono destinati ad essere scelti come esemplari. Poichè inizialmente non sono conosciuti i valori delle preferenze dei punti, si assume che tutti i punti dati siano ugualmente possibili come esemplari. Il loro valore diagonale è impostato quindi ad un valore comune per ciascuno di essi.

2. Matrice di responsabilità

La seconda matrice da tenere in considerazione è la matrice di responsabilità, R, dove la "responsabilità" *r(i,k)* riflette quanto il punto k sia adatto a fungere da esempio per il punto i, tenendo conto di altri potenziali esemplari per il punto i.

3. Matrice di disponibilità

   La terza matrice è la matrice di disponibilità, A, dove la "disponibilità" a(i,k), riflette quanto sia appropriato per il punto i scegliere il punto k come suo esemplare, tenendo conto del supporto di altri punti per cui quel punto k dovrebbe diventare un esemplare.

L'algoritmo avvia la procedura settando le due matrici R ed A a zero e prosegue aggiornando in modo iterativo i loro valori.

R è aggiornata usando la seguente equazione:

$$r(i,k) \leftarrow s(i,k) - \max_{k's.t.k' \neq k} \{a(i,k') + s(i,k')\}$$

A è aggiornata usando la seguente equazione:

$$a(i,k) \leftarrow \min \left(0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k))\right) \text{ for i} \neq k$$

$$a(k,k) \leftarrow \sum_{i' \neq k} \max(0, r(i',k))$$

A ed R sono aggiornate continuamente ad ogni iterazione dell'algoritmo, fino a quando non si incontra la convergenza, ovvero non cambia nulla per un cento numero di iterazioni. Quando l'algoritmo viene arrestato, la decisione finale viene presa in base alla somma di R e A. Il nome della colonna corrispondente al valore massimo per ciascuna riga sarà l'esempio per quella riga. Le righe che condividono lo stesso esemplare vengono assegnate allo stesso cluster.

| | Michael | John | Jim | Alex | Clark | Brown |
|---|---|---|---|---|---|---|
| **Michael** | 12 | -3 | 14 | -19 | -4 | 4 |
| **John** | 5 | 1 | 13 | -2 | -3 | 12 |
| **Jim** | 28 | -13 | 30 | -45 | 5 | -2 |
| **Alex** | -13 | -34 | 4 | -10 | 2 | -23 |
| **Clark** | -6 | -15 | -15 | 1 | 3 | -4 |
| **Brown** | 10 | 20 | 6 | -4 | 54 | -3 |

Figura 1.8: Affinity Propagation

### 1.3.8 Birch

Quando si esegue l'operazione di clustering su insiemi di grandi dimensioni, Birch (Balanced Iterative Reducing and Clustering hierarchies) è una delle tecniche pià utilizzate. Birch raggruppa il dataset iniziale in sottogruppi definiti Clustering Feature (CF), detti anche riepiloghi, dopo di che effettua la divisione effettiva in cluster. In particolare, converte i dati in una struttura dati ad albero con le foglie rappresentanti le centroidi del dataset. Tali centroidi possono essere utilizzate per ottenere la divisione totale del dataset, oppure possono fungere da input per altri algoritmi di clustering come ad esempio l'Agglomerative Clustering oppure il KMeans. Per tale motivo, l'algoritmo Birch può essere visto anche come una fase di pre-computazione dei dati prima della vera e propria operazione di clustering.

Birch tenta di minimizzare il requisito di memoria riassumento le informazioni contenute nelle regioni più dense dei dati (CF).

**Theorem 1** *Dati N punti d-dimensionali in un cluster, $X_i$ dove i = 1,2,...,N , la Clustering Feature (CF) del cluster è definita come una tripla:* ***CF*** *= (N, LS, SS), dove N è il numero di punti nel cluster, LS è la somma lineare degli N punti, i.e.$\sum_{i=1}^{N} X_i$, e SS la somma dei quadrati degli N punti, i.e.$\sum_{i=1}^{N} X_i^2$*

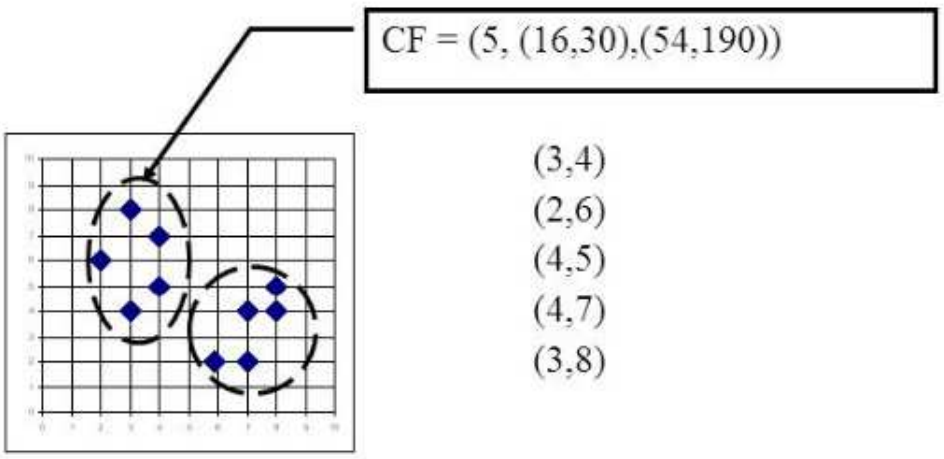



Figura 1.9: CF computing

L'algoritmo comprime i dati in un insieme di nodi CF, fino a formare un albero gerarchico (CF-tree). Quest'ultimo rappresenta una forma molto compatta dei dati, in quanto ogni nodo foglia non è soltanto un singolo punto, ma un sottocluster. Ogni nodo che non è una foglia contiene al massimo B elementi. Il diametro di ogni foglia deve essere pari almeno al valore di una treshold.

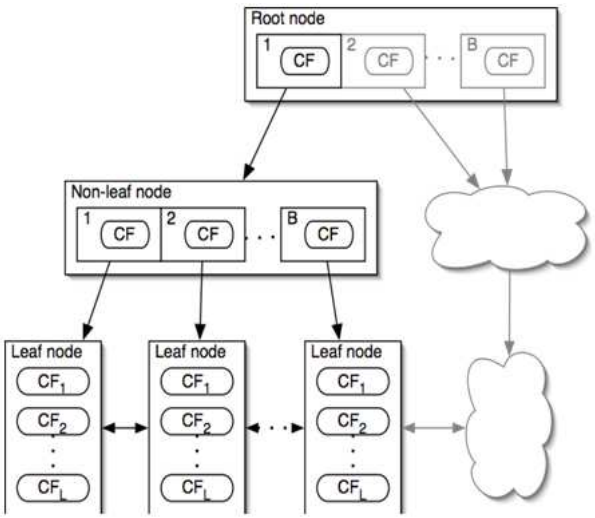



Figura 1.10: CF tree

Se la procedura finisce la memoria, prima di terminare la scansione dei dati, aumenta il valore della treshold e ricostruisce un nuovo CF-tree più piccolo, reinserindo le foglie del vecchio albero. Dopo che tutte le foglie sono state reinserite, la scansione dei dati e l'inserimento nel nuovo albero CF viene ripresa dal punto in cui era stata interrotta. Ottenuto l'albero complessivo nella sua versione finale, e' possibile usare un qualsiasi algoritmo di clustering, adattato per il seguente problema, per categorizzare i dati.

# Capitolo 2

# CircleClustering

## 2.1 Introduzione

Un algoritmo di clustering ideale si caratterizza per una buona scalabilità sui dati, per l'individuazione di cluster con forme arbitrarie, per l'astrazione su dati rumorosi e altri requisiti osservati nel capitolo 1. Nella prima sezione di questo lavoro sono state presentate alcune delle tecniche più famose utilizzate per effettuare l'operazione di clustering. Esistono inoltre tantissime altre metodologie e la letteratura è alquanto ampia riguardo tale materia. Tuttavia, nonostante tali tecniche siano in grado di risolvere discretamente la divisione in sottogruppi, la maggior parte di esse presenta un grave problema, la richiesta iniziale di un parametro k, indicante il numero finale di cluster che si vogliono ottenere dalla scomposizione dell'insieme iniziale.

Se un algoritmo riceve in input un valore numerico indicante il numero di sottogruppi dell'insieme, si avranno conseguenze negative molto importanti. La prima sarà di ordine computazionale. Dato che molte volte non si conosce inizialmente il numero di cluster corretto del dataset, sarà necessario tentare differenti esecuzioni utilizzando parametri differenti.

L'algoritmo computerà quindi molte volte il calcolo sullo stesso dataset e ciò comporterà un aumento del costo computazionale sicuramente non indifferente. Se inoltre si considera che molti algoritmi (vedi KMeans) trovano un ottimo locale, e che quindi devono essere eseguiti molteplici volte, allora il costo cresce in modo drammatico. Un singolo algoritmo invece di eseguire una singola volta, eseguirà centinaia di volte.

Infine, come valutare se i gruppi e il labeling prodotti da una esecuzione sono migliori o peggiori rispetto un'altra esecuzione? Seppur è vero che una persona manualmente è in grado di determinare, su dataset di piccole dimensioni, se un algoritmo ha prodotto buoni risultati oppure no, si può dire lo stesso su dataset di elevate dimensioni? Si rendono necessarie in questi casi delle strategie in grado di determinare quale sia stata l'esecuzione migliore; tali metodologie prendono spesso il nome di in-

dici, si citano ad esempio l'indice di Silhouette, l'indice di Calinski Harabasz, oppure ancora l'indice aic/bic. Tali indici sfruttano dei concetti simili agli algoritmi di clustering, come l'omogeneità intra-cluster o la lontananza dei diversi sottogruppi, per produrre delle stime sui dati e permettere di decidere quale sia il raggruppamento migliore. Tali algoritmi ovviamente presentano il fondamentale problema di essere molto imprecisi e hanno un costo computazionale a volte anche superiore alle stesse tecniche di clustering.

I motivi computazionali mostrano immediatamente che gli algoritmi di clustering, cui deve essere fornito il numero $k$ di cluster, propendono per una perdita di risorse di calcolo enorme dovuta ai tentativi e ai confronti che devono essere eseguiti.

Oltre al costo esponenziale, si potrebbe dire esserci anche una motivazione etico/strutturale legata agli algoritmi che richiedono il parametro $k$ in ingresso. Il clustering è una disciplina che ricade sotto le strategie dei metodi non supervisionati (unsupervised learning method). Questi algoritmi scoprono modelli nascosti o raggruppamenti di dati senza la necessità dell'intervento umano. La capacità di scoprire somiglianze e differenze nelle informazioni li rende la soluzione ideale per l'analisi esplorativa dei dati, la segmentazione in parti e il riconoscimento delle immagini. Tuttavia, un algoritmo che in ingresso assume il numero finale di cluster, è un algoritmo poco esplorativo, il cui unico obiettivo è quello di definire una label per i singoli punti dell'insieme. Ancora una volta è necessario ripetere che nel mondo reale è molto difficile, se non quasi impossibile, disporre di informazioni a priori riguardo un insieme di dati osservati. Molte volte i dati sono raccolti e accumulati senza nessuna conoscenza pregressa e ci si aspetta che sia proprio l'algoritmo in esame a definire il numero di sottogruppi presenti nell'insieme.

L'obiettivo di questo lavoro sarà quello di mostrare una nuova tecnica di clustering in grado di dividere i dati in ingresso senza l'ausilio di nessun parametro aggiuntivo al dataset stesso. L'algoritmo tenterà, tramite l'osservazione autonoma dei dati, di determinare i singoli parametri di cui necessita, quali appunto il numero di cluster.

## 2.2 Riassunto del funzionamento dell'algoritmo

L'algoritmo si divide essenzialmente in tre parti sequenziali. La prima parte riguarda la scomposizione dei dati in ingresso e la loro disposizione su un cerchio unitario. La procedura tenta di spostare i dati in input sul perimetro del cerchio unitario e fa in modo di avvicinare i dati che presentano delle similitudini, al tempo stesso tenta di allontanare coloro che non ne presentano o comunque in minor forma. Al termine di questa fase, i dati saranno stati raggruppati sul cerchio unitario e sarà possibile osservare quanti gruppi sono presenti nel dominio iniziale.

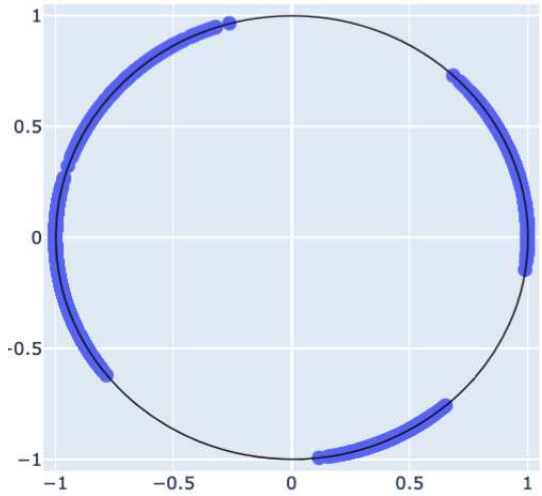



Figura 2.1: Cerchio Unitario

Se da un lato tale osservazione è facile o comunque intuitiva per l'occhio umano, ben più difficile è per un computer, il quale necessita di una euristica ben definita per essere in grado di adempiere a tale scopo. La seconda fase sarà infatti proprio questa, ovvero il tentativo di determinare con un processo algoritmico, il numero di cluster presenti sul cerchio. Per fare ciò si muoverà il dominio del cerchio verso un segmento di retta dove si ricaverà l'istogramma della densità.

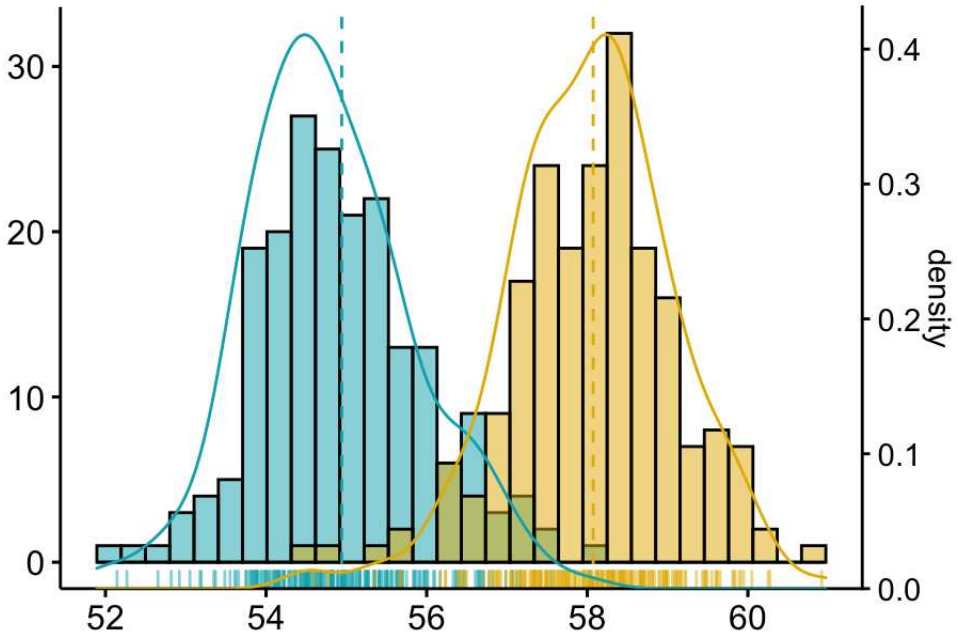



Figura 2.2: Istogramma della densità

Ottenuto l'istogramma sarà necessario cercare di determinare con un ulteriore algoritmo, il numero corretto di k classi presenti nel dataset. In figura 2.2 si possono osservare due macrogruppi; il cervello umano risolve questo compito facilmente e os-

serva che vi sono due differenti cluster, ma per un computer si tratta di un task molto complesso che può essere risolto in numerosi modi. Uno dei tentativi che ha portato a risultati di maggior successo è stata la costruzione dell'albero gerarchico delle aree. Un albero di ricerca gerarchico delle aree è una struttura dati molto simile ad un albero, dove tuttavia i nodi e gli archi riprendono il valore delle aree accumulate presenti nell'istogramma di densità. Il processo è molto veloce ed è in grado di portare a buoni risultati in diversi contesti. Ottenuto il numero di k cluster presenti nel dataset, ciò che resterà da fare sarà individuare la label di appartenenza di ciascun campione dell'insieme. Questa sarà la terza fase. Si mostrerà che per tale scopo vi è un algoritmo di clustering in letteratura in grado di svolgere adeguatamente questo compito, chiamato "EM (Expectation Maximization)".

## 2.3 Prima fase

Di seguito presentiamo una nuova tecnica che mira a condensare gli elementi simili sul perimetro di un cerchio unitario ed allontanare quelli dissimili. Dato un insieme $\mathcal{S}$, definiamo per ogni coppia di valori $\mathcal{S}_i, \mathcal{S}_j \in \mathcal{S}$ la pair-wise distance e costruiamo la matrice di dissimilarità $W = (w_{ij})$ dove $w_{ij} = \text{distance}(\mathcal{S}_i, \mathcal{S}_j)$ fornisce un grado di (dis)similarità tra due vettori $\mathcal{S}_i$ e $\mathcal{S}_j$. Per ragioni pratiche che diverranno chiare successivamente, associamo ad ogni vettore $\mathcal{S}_i \in \mathcal{S}$ un vettore sul cerchio unitario $\xi_i \in \mathrm{S}_2$. Facciamo ciò per scalare la dimensionalità dei vettori iniziali $\mathcal{S}_i$ e perchè il prodotto interno tra due vettori $\xi_i = (\cos\theta_i, \sin\theta_i)$ e $\xi_j = (\cos\theta_j, \sin\theta_j)$ verifica $\xi_i \cdot \xi_j = \cos(\theta_i - \theta_j)$, la similarità relativa di due vettori in $\mathrm{S}_2$ dipende dalle variabili scalari $\theta_i$ e $\theta_j$ e può essere scritta come una funzione che (con abuso di notazione) assume la seguente forma:

$$d(\xi_i, \xi_j) = \frac{1 - \xi_i \cdot \xi_j}{2} = d(\theta_i, \theta_j) = \frac{1 - \cos(\theta_i - \theta_j)}{2} \tag{2.1}$$

(2.1) prende valori nel range unitario reale, dove 0 indica massima similarità e 1 massima dissimilarità; valori intermedi forniscono un grado di similarità proporzionale al coseno dell'angolo tra due vettori $\xi_i$ e $\xi_j$. Combinando i pesi $w_{ij}$ e le funzioni definite per ogni coppia di indici distinti $\{i, j\}$ in (2.1), il problema di trovare delle partizioni in S, può essere formulato come un problema di ottimizzazione sull'ipercubo

$\mathcal{D} = [0, 2\pi]^L$ con una funzione di costo da massimizzare data da:

$$\begin{aligned} J(\theta_1, \ldots, \theta_L) &= \sum_{i<j} w_{ij}\, d_{i,j}(\xi_i, \xi_j) \\ &= \sum_{i<j} w_{ij} \frac{1 - \cos(\theta_i - \theta_j)}{2} \\ &= c - \frac{1}{2} \sum_{i<j} w_{ij} \cos(\theta_i - \theta_j), \end{aligned}$$

dove $c = \frac{1}{2} \sum_{i<j} w_{ij}$. In questa equazione il termine da massimizzare appare con il segno negativo, questo equivale a minimizzare la stessa con i segni invertiti:

$$J(\theta_1, \ldots, \theta_L) = \frac{1}{2} \sum_{i<j} w_{ij} \cos(\theta_i - \theta_j). \tag{2.2}$$

Intuitivamente, ogni termine $w_{ij} \cos(\theta_i - \theta_j)$ nella precedente somma contribuisce alla magnitudine della funzione $J$ in modo pesantemente dipendente sul valore del peso $w_{ij}$. Infatti quando $w_{ij} \approx 0$, il suo prodotto con $\cos(\theta_i - \theta_j)$ rende libera la scelta di $\theta_i$ e $\theta_j$, vice versa quando $w_{ij} > 0$, l'ampiezza di $\cos(\theta_i - \theta_j)$ deve essere forzata per essere vicina quanto più possibile a zero e conseguentemente per interporre un angolo $\pi$ tra $\xi_i$ e $\xi_j$. La funzione (2.2) è in generale non convessa e trovare l'esatta soluzione globale di un problema non convesso è NP-hard. Un modo per trattare con la non convessità è rilassare l'obiettivo dal trovare un minimo globale al cercare un minimo locale usando, ad esempio, una tecnica di ricerca locale basata su un sistema dinamico a tempo discreto con regola di aggiornamento asincrona.

Associamo alla funzione 2.2 una regola di aggiornamento locale $T = (T_k)_{k=1}^L$ su ipercubo $D$, i.e. $T(\theta) = (T_1(\theta_1), \ldots, T_L(\theta_L))$ tale che

$$T_k(\theta_k) =_{\theta_k \in [0,2\pi]} \{C_k \cos\theta_k + S_k \sin\theta_k\}, \tag{2.3}$$

dove

$$C_k = \sum_{k \neq j} w_{kj} \cos\theta_j \quad \text{and} \quad S_k = \sum_{k \neq j} w_{kj} \sin\theta_j. \tag{2.4}$$

Per il sistema nell'equazione 2.3, il punto $\theta_k^* \in [0, 2\pi]$ è un punto di equilibrio se e solo se

$$T_k(\theta_k^*) = \theta_k^*, \quad k = 1, \ldots, L,$$

Ciò implica

$$T(\theta^*) = \theta^*, \quad \theta^* \in D. \tag{2.5}$$

Per gestire (2.5) introduciamo il sistema autonomo non lineare a tempo discreto

definito dalla relazione ricorsiva

$$\theta(t+1) = T(\theta(t)), \quad \theta(0) = \theta_0, \quad t \in \mathbb{N} \tag{2.6}$$

dove $\theta(t) \in D$ ad ogni momento $t$. Su operazione asincrona, le componenti del vettore corrente $\theta(t)$ sono aggiornate una alla volta in accordo con (2.3) per produrre un nuovo stato del vettore. Per mostrare che il sistema (2.6) converge ad un punto fissato stabile asintoticamente $\theta^* \in D$ che minimizza localmente la funzione $J$ su $D$ facciamo le seguenti considerazioni. Il punto d'equilibrio $\theta^* \in D$ è detto essere (localmente) asintoticamente stabile nel senso di Lyapunov se per ogni $\epsilon > 0$, esiste qualche $\delta > 0$ tale che $||\theta(0) - \theta^*|| < \delta$ che implica $||\theta(t) - \theta^*|| < \epsilon$ per tutti $t \in \mathbb{N}$, allora $\lim_{t\to+\infty} \theta(t) = \theta^*$. Cioè, se lo stato di un sistema è vicino all'equilibrio iniziale, esso sta sempre vicino all'equilibrio. Se, in aggiunta, lo stato converge all'equilibrio, $\theta^*$ è detta essere asintoticamente stabile nel senso di Lyapunov. Per provare la proprietà di stabilità per (2.6) il seguente lemma è definito.

Lemma: Assumiamo $W = (w_{ij})$ sia una $L \times L$ matrice simmetrica, e $\theta_j$, per $j = 1, \ldots, L$, siano fissate, allora la differenza

$$\Delta_k J = J(\theta_1, \ldots, \theta'_k, \ldots, \theta_L) - J(\theta_1 \ldots, \theta_k, \ldots \theta_L) \leq 0$$

se e solo se

$$\begin{cases} \pi - \alpha \leq \theta'_k \leq 2\pi, & \text{if } \alpha \leq \pi \\ 0 \leq \theta'_k \leq 2\pi - \alpha, & \text{if } \alpha > \pi \end{cases}$$

dove $\cos\alpha = C_k / \sqrt{C_k^2 + S_k^2}$, e $C_k$, $S_k$ sono definite in (2.4). Inoltre:

$$_{\theta'_k \in [0,2\pi]}\Delta_k J = \begin{cases} \alpha + \pi, & \text{if } \alpha \leq \pi \\ \alpha - \pi, & \text{if } \alpha > \pi \end{cases}.$$

Dim.: A causa della simmetria di $W$ cominciamo riscrivendo la funzione $J$ in (2.2) nella forma:

$$\begin{aligned} J(\theta_1, \ldots, \theta_L) &= \frac{1}{4} \sum_{i,j=1}^{L} w_{ij} \cos(\theta_i - \theta_j) \\ &= \frac{1}{4} \sum_{j=1}^{L} w_{kj} \cos(\theta_k - \theta_j) + c, \end{aligned}$$

dove $c = \frac{1}{4} \sum_{i,j\neq k} w_{ij} \cos(\theta_i - \theta_j)$.

Usando le variabili $C_k$ e $S_k$ definite in (2.4), possiamo scrivere la differenza

$$\begin{aligned}\Delta_k J &= J(\theta_1, \dots, \theta'_k, \dots, \theta_L) - J(\theta_1 \dots, \theta_k, \dots \theta_L) \\ &= \frac{1}{4}\sum_j w_{kj}\big(\cos\theta'_k \cos\theta_j + \sin\theta'_k \sin\theta_j\big) \\ &\qquad - \frac{1}{4}\sum_j w_{kj}\big(\cos\theta_k \cos\theta_j + \sin\theta_k \sin\theta_j\big) \\ &= \frac{1}{4}\big(C_k \cos\theta'_k + S_k \sin\theta'_k - C_k \cos\theta_k - S_k \sin\theta_k\big).\end{aligned}$$

Moltiplicando e dividendo $\Delta_k J$ per $Z_k = 4\sqrt{C_k^2 + S_k^2}$ e impostando $\cos\alpha_k = C_k/Z_k$, $\sin\alpha_k = S_k/Z_k$, abbiamo

$$\begin{aligned}\Delta_k J &= Z_k(\cos\alpha_k \cos\theta'_k + \sin\alpha_k \sin\theta'_k \\ &\qquad - \cos\alpha_k \cos\theta_k - \sin\alpha_k \sin\theta_k) \\ &= Z_k\big(\cos(\theta'_k - \alpha_k) - \cos(\theta_k - \alpha_k)\big).\end{aligned}$$

Perciò, assumendo w.l.o.g che $\alpha_k \in [0, 2\pi]$, poichè $Z_k > 0$, allora $\Delta_k J \leq 0$ se e solo se

$$\begin{cases}\pi - \alpha_k \leq \theta'_k \leq 2\pi, & \text{if } \alpha \leq \pi \\ 0 \leq \theta'_k \leq 2\pi - \alpha_k, & \text{if } \alpha > \pi\end{cases}$$

In particolare, il minimo della differenza $\Delta_k J$ vale:

$$\begin{aligned}\hat{\theta}_k &= {}_{\theta'_k \in [0,2\pi]}\ Z_k\big(\cos(\theta'_k - \alpha_k) - \cos(\theta_k - \alpha_k)\big) \\ &= -{}_{\theta'_k \in [0,2\pi]} Z_k\big(1 + \cos(\theta_k - \alpha_k)\big)\end{aligned}$$

i.e.,

$$\hat{\theta}_k = \begin{cases}\alpha_k + \pi, & \text{if } \alpha_k \leq \pi \\ \alpha_k - \pi, & \text{if } \alpha_k > \pi\end{cases}.$$

Per utilizzare i risultati fondamentali della teoria della stabilità di Lyapunov per sistemi dinamici autonomi non lineari (2.6) osserviamo che $J : \mathcal{D} \to \mathbb{R}$ è Lipschitz continua su $\mathcal{D}$ poiché lo stesso vale per la funzione trigonometrica, $\cos : [0, 2\pi] \to [-1, 1]$, essendo la sua costante di Lipschitz 1.

Per il Lemma precedente e la continuità della funzione (2.2) il teorema principale sul sistema dinamico non lineare a tempo discreto afferma che J è una funzione di

Lyapunov per il sistema (2.6). Infatti, per $J(\theta) - c$, si ha:

$$\begin{aligned} J(0) &= 0, \\ J(\theta) &> 0, \quad \theta \in \mathcal{D} - \{0\}, \\ J(T(\theta)) - J(\theta) &\leq 0, \quad \theta \in \mathcal{D}. \end{aligned}$$

### 2.3.1 Algoritmo

Per trovare la soluzione di (2.6) in pratica applichiamo lo schema ricorsivo sotto:

**Algorithm 1** CircleClustering

**Data**: La matrice pesata $W$ e un reale $\varepsilon > 0$
**Result**: il vettore $\theta$
**for** $k := 1$ **to** $L$ **do**
    $\theta_k \leftarrow$ un numero casuale in [0,2$\pi$]
**end for**
**for** $i := 1$ **to** $L$ **do**
    $C_k \leftarrow \sum_j w_{kj}\, cos\theta_j$
    $S_k \leftarrow \sum_j w_{kj}\, sin\theta_j$
**end for**
cond $\leftarrow$ True
**while** (cond) **do**
    cond $\leftarrow$ False
    **for** $k := 1$ **to** $L$ **do**
        $\hat{\theta}_k \leftarrow argmin_{\gamma \in [0,2\pi]} \{C_k \cos\gamma + S_k \sin\gamma\}$
        **if** $|\hat{\theta}_k - \theta_k| > \varepsilon$ **then**
            cond $\leftarrow$ True
            **for** $j := 1$ **to** $L$ **do**
                $C_j \leftarrow C_j + w_{kj}\left(\cos\hat{\theta}_k - \cos\theta_k\right)$
                $S_j \leftarrow S_j + w_{kj}\left(\sin\hat{\theta}_k - \sin\theta_k\right)$
            **end for**
            $\theta_k \leftarrow \hat{\theta}_k$
        **end if**
    **end for**
**end while**

La procedura involve un'iterazione che calcola una soluzione approssimata per (2.6) generando una traiettoria finita $\theta(0), \theta(1), \theta(2), \ldots, \theta(\hat{t})$, partendo da un punto arbitrario $\theta(0) = \theta_0$, fino a quando non si verifica la condizione $|J(T(\theta(\hat{t}))) - J(\theta(\hat{t}))| < \varepsilon$, con un $\varepsilon > 0$ fissato. Per quanto riguarda la complessità dell'algoritmo, la ripetizione del **while** loop rappresenta il blocco che richiede più tempo all'interno dell'algoritmo, in particolare, data un'istanza di dimensione $n$ e un valore reale $\varepsilon > 0$, il tempo computazionale del caso peggiore è $\mathcal{O}(n^2/\varepsilon)$.

## 2.4 Seconda fase

Ad ogni punto del dataset originale abbiamo associato un valore theta, posizionato sul perimetro del cerchio. Allo stato corrente tuttavia, i valori in output dalla fase 1 sono difficili da utilizzare. Si possono già "vedere" dei cluster, ma l'identificazione manuale del numero di gruppi corretto e soprattutto l'operazione di labeling rimangono operazioni quasi impossibili, soprattutto se si considerano dataset di grandi dimensioni, con decine di migliaia di punti (angoli) distribuiti nell'intervallo $[0, 2\pi)$ o punti sul cerchio. Per capire come risolvere il problema, può risultare utile osservare i dati da un altro punto di vista. Ciò a cui siamo interessati è la densità dei theta sul cerchio; il risultato della prima fase evidenzia infatti che ci sono archi di cerchio con addensamenti, dove i theta si sono accumulati e archi completamente liberi. Osservare questo tipo di caratteristica è piuttosto semplice, basta infatti costruire l'istogramma della densità del vettore di valori theta ricavato dalla prima fase.

Definito un range fisso $[0, x]$, l'istogramma della densità è un comune istogramma dove il numero di *bins* (rettangoli) corrisponde al numero di classi in cui è stato suddiviso tale tange. La larghezza della base dei rettangoli dipende dall'ampiezza di tali classi e, complessivamente, tutte le basi dei rettangoli affiancate devono coprire l'intera gamma del range. L'altezza di ogni rettangolo indica invece la frequenza (densità) dei casi presenti in ogni classe; la frequenza rappresenta quante unità statistiche sono presenti all'interno della classe.

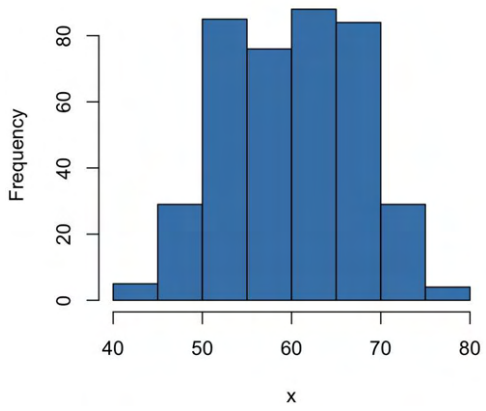



Figura 2.3: Esempio istogramma della densità

Rimane ovviamente una variabile fondamentale da gestire: il numero di classi con cui suddividere il segmento di retta, ovvero il numero di rettangoli che si vogliono ottenere. Si osservi l'esempio in figura 2.4.

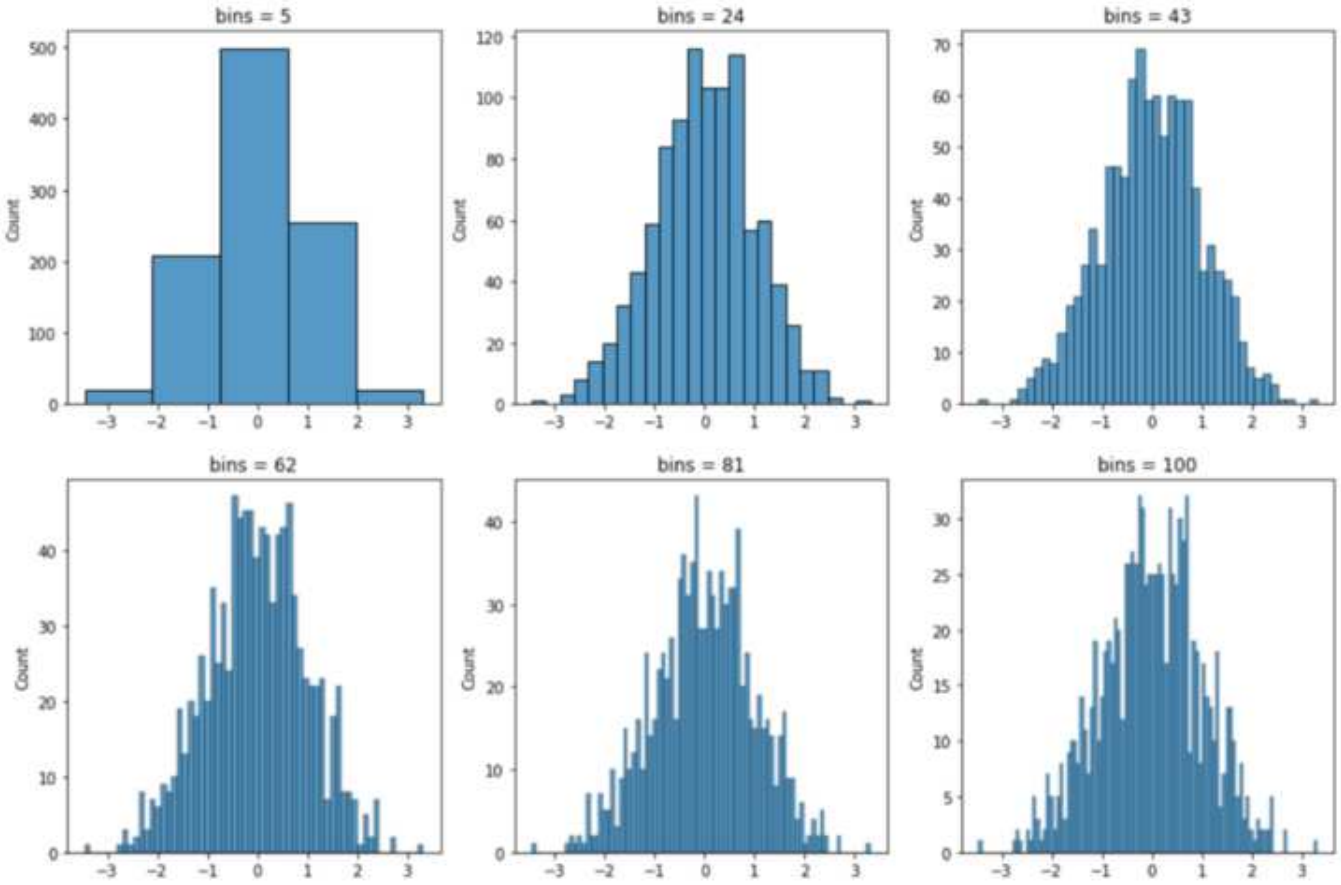



Figura 2.4: Esempio scelta differente del numero di bins

Risulta subito chiaro che la scelta del numero di bins produrrà risultati significativamente differenti. Con un numero di bins troppo alto, saremo in grado di estrarre più informazioni, ma al contempo sarà impossibile astrarre sui dati, ovvero sarà complesso capire il numero di cluster presenti a causa del rumore. Con un numero troppo basso, al contrario, rischieremmo di eliminare informazioni fondamentali e di ottenere un numero di cluster troppo basso rispetto al numero corretto.

Per quanto riguarda invece la scelta di un parametro statico o dinamico, un parametro statico potrebbe anche portare a risultati discreti, dopo uno studio sperimentale su dataset con features e forme differenti, ma della stessa misura. Tuttavia, un dataset con milioni di dati avrà sicuramente bisogno di un numero di bins molto superiore rispetto un dataset composto da centinaia di campioni, per catturare tutte le sfumature dei dati. La scelta di questo parametro deve essere quindi dinamica. Un metodo comunemente utilizzato per risolvere tale problema è la regola di Freedman-Diaconis. Per un insieme di misurazioni empiriche campionate da una distribuzione di probabilità, la regola di Freedman-Diaconis è progettata approssimativamente per minimizzare l'integrale della differenza al quadrato tra l'istogramma (cioè la densità

di frequenza relativa) e la densità della distribuzione di probabilità teorica.

$$bin\ width = 2\frac{IQR(x)}{\sqrt[3]{n}}$$

dove IQR$(x)$ è l'intervallo interquantile dei dati e n è il numero di osservazioni nel campione $x$. L'intervallo interquartile è una misura della dispersione statistica, ovvero la diffusione dei dati. È definito come la differenza tra il 75° e il 25° percentile dei dati. Per calcolare l'IQR, il set di dati è diviso in quartili, ordinate per rango tramite interpolazione lineare. Questi quartili sono indicati con Q1 (chiamato anche quartile inferiore), Q2 (la mediana) e Q3 (chiamato anche quartile superiore). Il quartile inferiore corrisponde al 25° percentile e il quartile superiore corrisponde al 75° percentile, quindi $IQR = Q3 - Q1$. Una volta ottenuto il valore della larghezza di una cella, ovvero di un bin, possiamo trovare il numero corretto di bins attraverso la semplice equazione

$$bins = ceil(\frac{max(x) - min(x)}{bin\ width})$$

*ceil* è un'operatore che arrotonda un numero per eccesso all'intero più vicino e restituisce il risultato. Ora che è stata definita una metodologia per il calcolo del numero dei bins, è possibile ottenere l'istogramma della densità a partire dai valori di theta precedentemente calcolati, il risultato sarà simile al seguente:

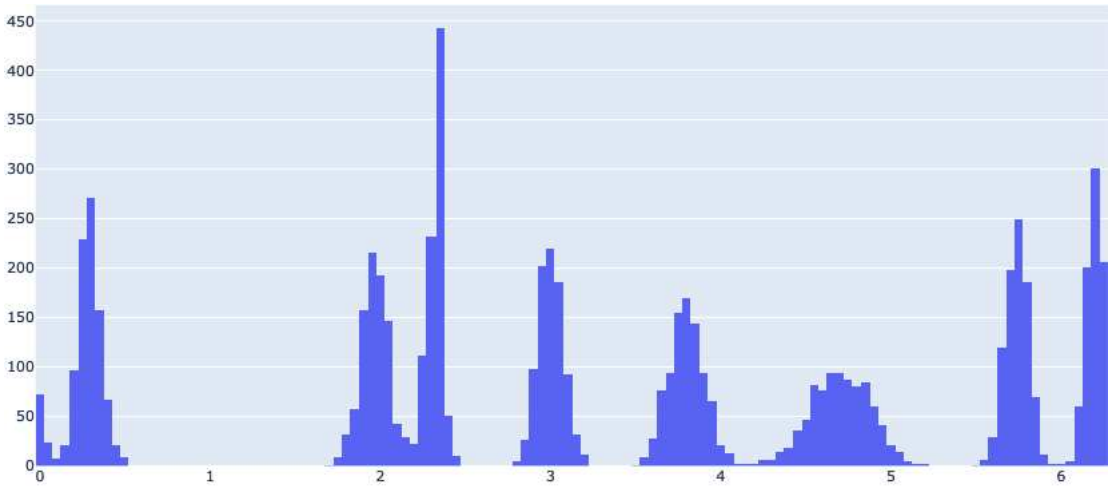



Figura 2.5: visualizzazione per istogramma

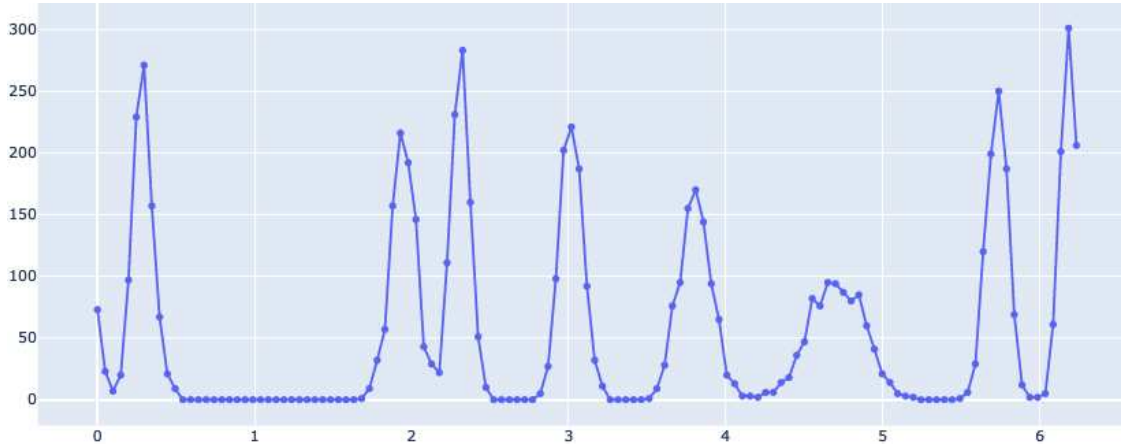



Figura 2.6: visualizzazione simil curva

I picchi, che rappresentano essenzialmente i cluster, ovvero zone ad alta densità, sono simili a delle montagne, mentre le regioni più sparse o assenti sono simili a delle valli. Purtroppo, anche con la definizione di "montagne e valli" per i cluster, può risultare difficile sapere se qualcosa è o meno un cluster. Si veda l'esempio in figura 2.7. Qui, abbiamo ottenuto la PDF (probability density function), ovvero una funzione più omogenea e curvilinea dell'andamento dell'istogramma di probabilità. La PDF è utilizzata solo a titolo di esempio, poichè mostra in modo migliore come si può lavorare con l'istogramma della densità.

Come si può vedere ci sono tre picchi, ma questi corrispondono anche a tre cluster? "Intuitivamente", diciamo che ci sono solo 2 cluster. Come facciamo quindi a decidere?

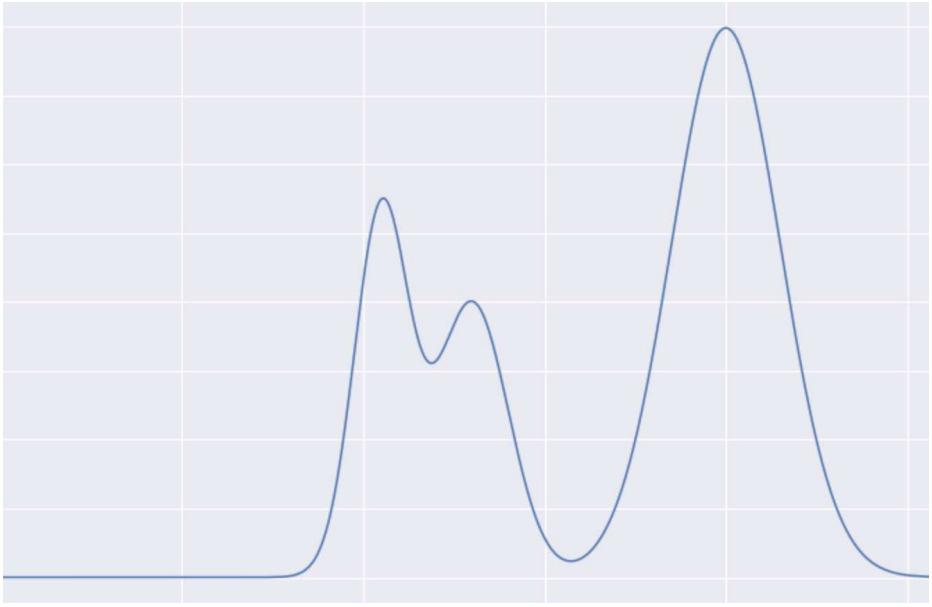


Figura 2.7: Esempio PDF

Un'euristica che ha portato a risultati interessanti è quella legata alle aree accumulate dell'istogramma di densità. Osserviamo attentamente l'immagine 2.7, risulta

subito chiaro che ogni picco, per essere tale, dipende dall'area sottesa. Un'area più grande comporterà un picco più alto o più esteso di un'area più piccola.

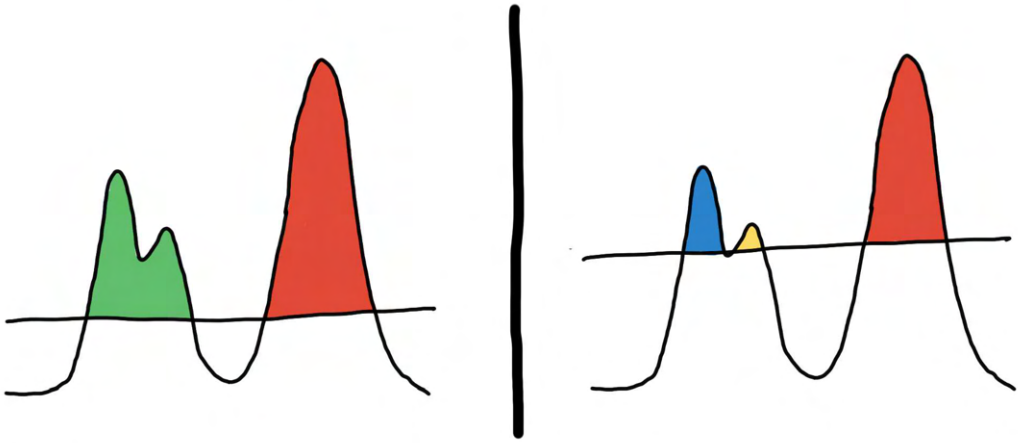


Figura 2.8

A seconda del punto in cui iniziamo a calcolare la nostra area, ovvero "tagliamo" la PDF, otteniamo un numero di montagne e di valli differente. Se ad esempio, come nell'immagine 2.8, tracciamo una linea a metà della curva, si verranno a creare due aree (verde - rosso). Se invece scegliamo di tracciare la linea più in alto, otterremo tre aree ben distinte (blu - giallo - rosso). Supponiamo ora di tagliare la PDF ogni qualvolta si incontra una valle, ovvero un minimo locale della funzione, e calcolare l'area in ciascun intervallo venutosi a creare, come in figura 2.9.

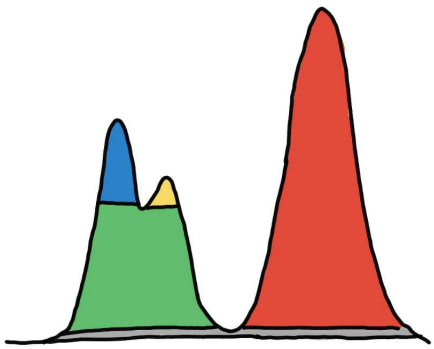


Figura 2.9: Esempio del taglio nei minimi locali

Come dimostrato precedentemente, ogni valle è sinonimo di divisione di due o più picchi. Quello che vogliamo scoprire è se l'area di un taglio sottostante supera oppure meno i tagli superiori. In altre parole, se il colore sottostante, supera la somma dei colori nelle aree superiori. In caso positivo, potremo dire che si tratta di un unico cluster, altrimenti dovremo ripetere in modo ricorsivo il calcolo per le aree superiori.

Questa euristica, applicata ad una funzione oppure un istogramma di densità, permette di definire in modo chiaro e gerarchico quanti cluster sono presenti. Possiamo sviluppare una matematica basata sulle relazioni di grandezza tra le diverse aree.

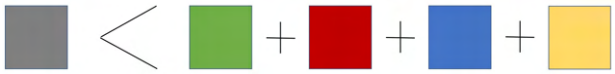


Figura 2.10

Nel caso in figura 2.9, come viene evidenziato in figura 2.10, la prima area, l'area grigia, non supera la somma della aree superiori, ovvero la somma delle aree verde, rossa, blu e gialla. Per scoprire il numero di cluster è quindi necessario ripetere l'osservazione in modo ricorsivo nelle aree superiori che sono state scoperte attraverso il primo taglio dei minimi locali. Allora in questo caso noteremo che la matematica dei colori è la seguente:

Figura 2.11

L'area rossa non ha bisogno di essere ricalcolata, in quanto non presenta ulteriori tagli. L'area verde invece presenta un taglio in due sottogruppi la cui area è superiore alle due successive. Per tal motivo possiamo interrompere la nostra analisi e affermare che ci sono due cluster nella funzione di densità analizzata.

Osserviamo ora un ulteriore esempio:

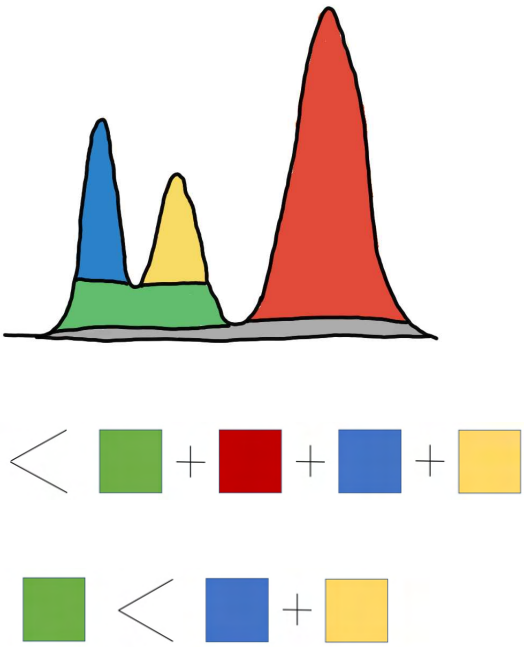
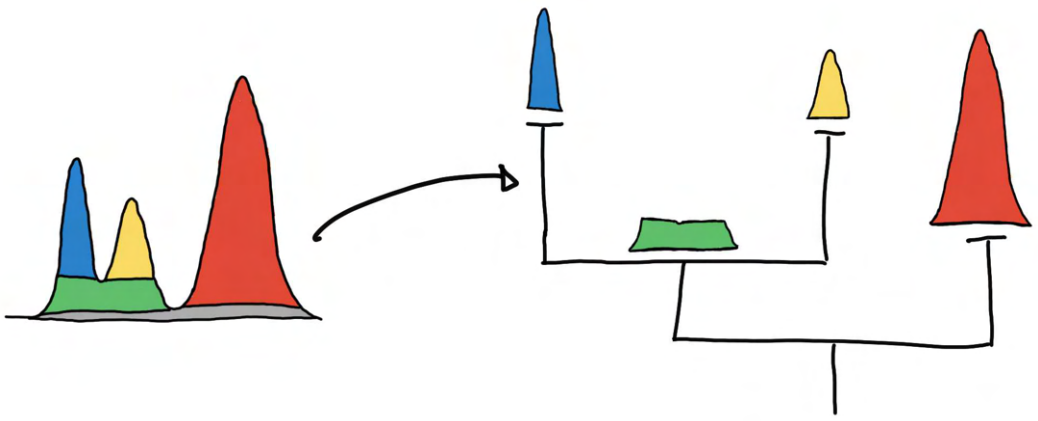


Figura 2.12

Qui ancora una volta l'area grigia non permette di notare che è presente un singolo cluster in quanto inferiore alle aree superiori. Ripetendo il problema sulle sottoparti notiamo però che anche l'area verde è inferiore alle relative aree successive. Perciò è necessario reiterare il calcolo sui due sottogruppi blu e giallo. Essendo essi la parte conclusiva, possiamo affermare che sono presenti nella nostra funzione tre cluster. Utilizzeremo questa euristica per cercare di scoprire il numero di cluster presenti nell' istogramma della densità.

Per poter catturare le relazioni tra le aree con un algoritmo, possiamo basarci sulla teoria degli alberi, in particolare sugli alberi gerarchici pesati. Questa struttura dati, che siamo in grado di attraversare e manipolare, rappresenta un'astrazione naturale del problema che stiamo affrontando. L'albero si compone di nodi e archi i quali possono rappresentare la densità empirica definita sopra. I nodi rappresenteranno i minimi locali della funzione, ovvero i punti di taglio, mentre gli archi dell'albero rappresenteranno le connessioni tra le diverse aree, e avranno dei pesi, indicanti l'area del pezzo di funzione e l'area accumulata fino a quel momento (la somma delle aree cumulative). La costruzione dell'albero comincia dal nodo radice, nodo vuoto raffigurante il livello zero, dove ancora non è presente nessuna area da calcolare. Dal nodo radice si proseguirà con il nodo dell'area grigia, il quale si dividerà in due macrogruppi, il gruppo composto da verde - blu - giallo e il gruppo composto dal rosso, e cosi via (Figura 2.13). Per convenzione, gli alberi sono disegnati dall'alto verso il basso, dove la radice è in cima e l'albero cresce verso il basso.



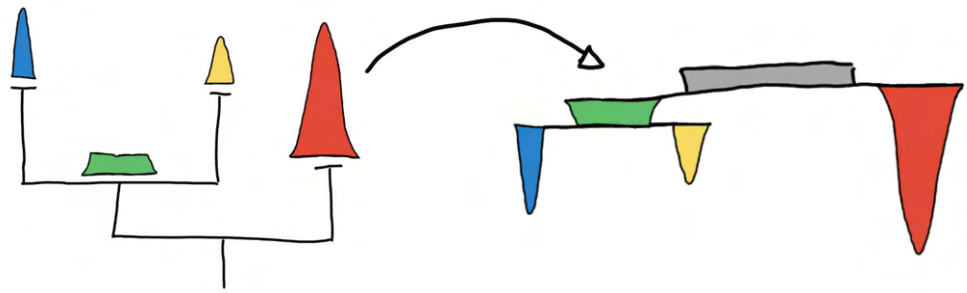


Figura 2.13

Abbiamo definito un'euristica in grado di stimare il numero di cluster e abbiamo definito una struttura dati tale da permetterci di manipolare e risolvere il problema.

### 2.4.1 Preprocessing

Prima di passare direttamente alla costruzione dell'albero gerarchico pesato, si devono affrontare alcuni problemi legati all'input attuale, ovvero l'istogramma dei theta. Il primo problema da risolvere necessariamente è il problema della rotazione. Poiché i valori sono distribuiti sul cerchio, come in figura (2.14), può accadere che la fase 1 abbia accumulato alcuni angoli nell'intorno di 0 o $2\pi$. Il picco creatosi in $[0, 2\pi)$ è stato spezzato perché stiamo lavorando non più su un cerchio continuo, ma su un segmento. Quando lanceremo l'algoritmo descritto precedentemente di ricerca del numero di cluster, incorreremo nel problema di valutare il picco spezzatosi in due differenti picchi.

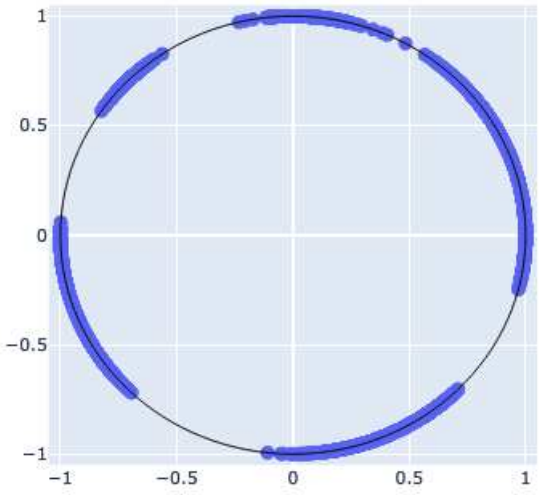



Figura 2.14

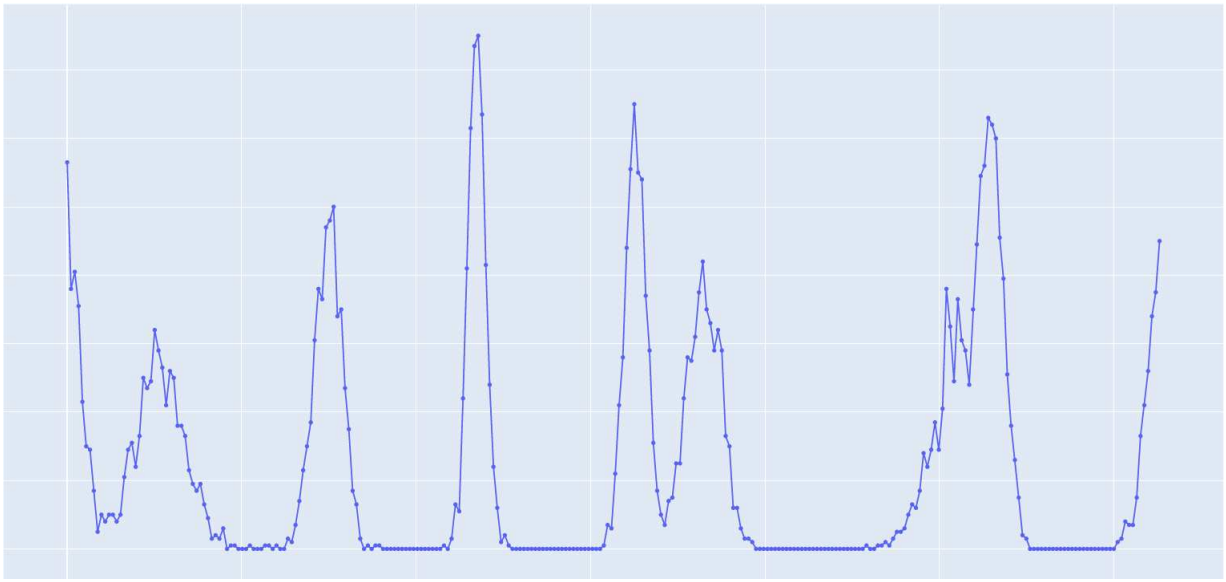


Figura 2.15: Problema rotazione su istogramma della densità

Per ovviare al fenomeno è possibile trovare un punto di minimo dell'intero istogramma e ruotare di conseguenza tutti i *bins*.

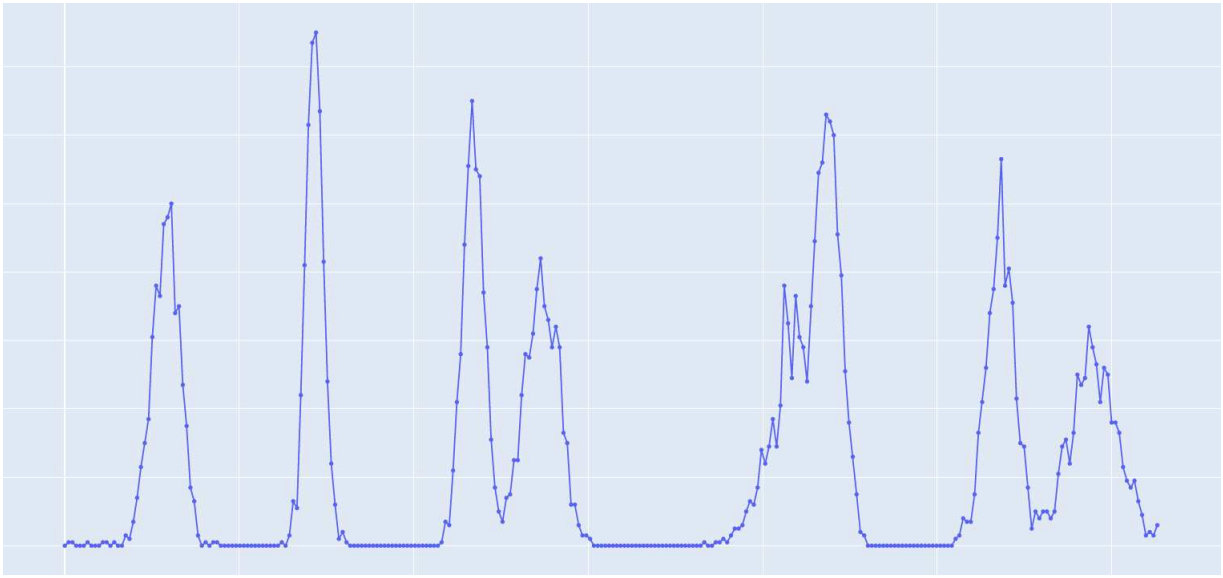


Figura 2.16: Istogramma della densità ruotato

Il secondo problema che necessita una soluzione è quello relativo ai cluster creati dal primo taglio, ovvero l'asse su cui poggia l'istogramma. Se applichiamo ora l'algoritmo di costruzione dell'albero, non otterremo un numero cluster corretti, ma bensi un numero di cluster ben più alto, in quanto tutti i lievi picchi presenti alla base dell'istogramma (che nell'intorno presentano delle valli) saranno considerati come dei cluster. Inoltre ha senso rimuovere questi piccolissimi centri di densità, in quanto rappresentano la quasi totalità delle volte solo rumore (sono infatti per la maggior parte singoli punti sparsi). Una tecnica che si può utilizzare è calcolare la media di tutte le differenze sequenziali tra i bins. Dopo di che, è rimosso tutto ciò che è inferiore a tale valore. L'algoritmo prende in input l'istogramma, che altro non è il vettore delle densità, e calcola il valore della soglia sotto cui eliminare i dati. Il risultato della sogliatura sarà come in figura 2.17

**Algorithm 2** removeLowPercentage

```
1: Data: (hist) -> istogramma della densità
2: Result: istogramma sogliato dal valore di sogliatura ricavato nello step 1
3: /* STEP 1
4: newHist ← array [hist.size()]
5: sumOfDifferences ← 0
6: count ← 0
7: for i := 0 . . . hist.size() − 1 do
8:     sumOfDifferences ← sumOfDifferences + abs(hist[i+1] - hist[i])
9:     count ← count + 1
10: end for
11: /* STEP 2
12: threshold ← 2 (sumOfDifferences / count)
13: for j := 0 . . . hist.size() do
14:     if hist[j] ≥ threshold then newHist[j] ← hist[j]
15:     else newHist[j] ← 0
16: end for
17: return newHist
```

Figura 2.17: Istogramma densità dopo rimozione del rumore a bassa percentuale

L'ultima operazione che andremo ad effettuare sarà anche l'operazione fondamentale di questa fase di preprocessing: lo smoothing dei dati. Analizzando meglio l'istogramma infatti, risulta evidente che i picchi non sono ancora molto puliti. L'operazione considerata è quella di smoothing, che è tuttavia un'operazione molto rischiosa in quanto può portare all'eliminazione di informazione fondamentale. Cercheremo

quindi di effettuare uno smoothing tale da rendere i dati più smooth, ma senza una grave perdita di informazione.

Per ottenere ciò una tecnica ben nota è la Kernel Density Estimation (KDE). La KDE fa utilizzo di un kernel, passante sopra ciascun elemento del vettore della densità; a seconda del tipo di kernel implementato, si possono ottenere risultati differenti di smoothing. In figura 2.18 sono mostrati alcuni kernel utilizzabili.

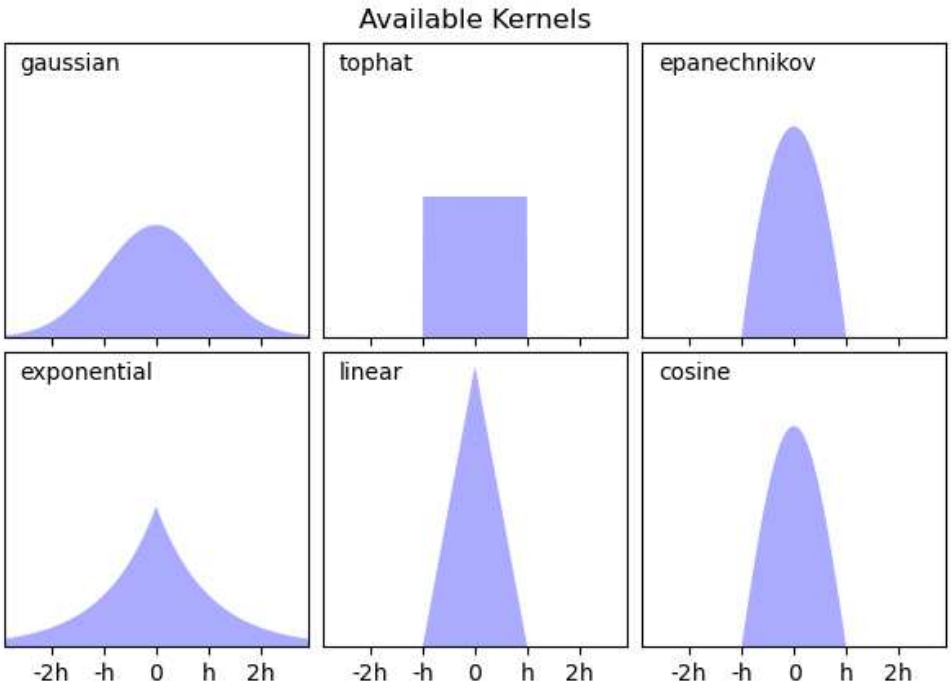



Figura 2.18: Kernel disponibili

Il kernel che ha prodotto i risultati migliori è stato il kernel basato sulla campana gaussiana. La Curva a campana di Gauss è la distribuzione normale delle probabilità ed è la modalità normale del manifestarsi di qualsiasi fenomeno naturale. Utilizzare una campana di gauss come kernel per levigare i dati, significa effettuare l'operazione di convoluzione sul vettore mono dimensionale della densità, ottenuto fino a questo momento. Segue il risultato in figura 2.19

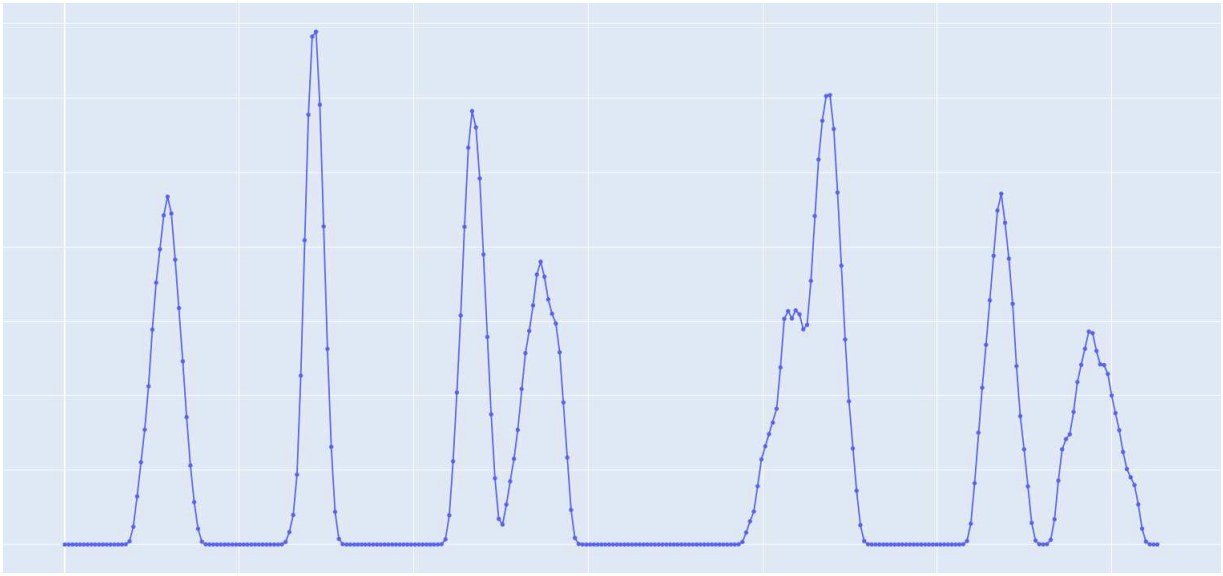


Figura 2.19: Istogramma densità dopo smoothing

### 2.4.2 Albero gerarchico pesato

Giunti a questo punto, è finalmente possibile implementare l'albero gerarchico pesato, ovvero la struttura dati che permetterà di rilevare il numero di cluster presenti nell'insieme iniziale dei dati. Nell'istogramma è complesso trovare dei minimi locali; ciò che abbiamo a disposizione infatti, non è nient'altro che la serie di bins dell'istogramma, identificata attraverso il vettore delle densità. Invece che effettuare un taglio ad ogni minimo locale, realizzeremo un taglio ogni volta che è incontrata una nuova altezza. Analizziamo l'esempio in figura:

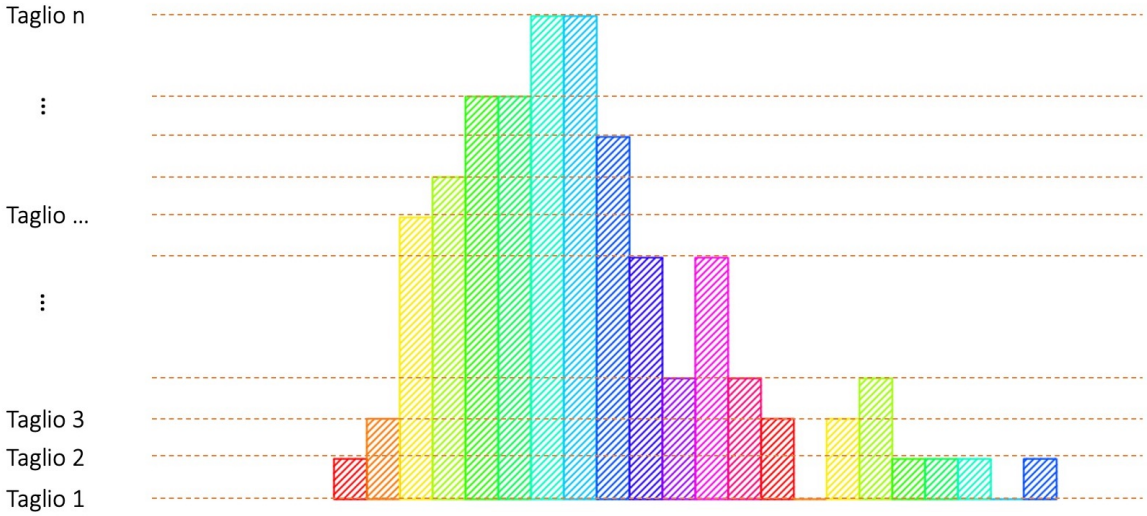



Figura 2.20: Istogramma tagliato ad ogni altezza

Come si vede, ad ogni altezza è stato effettuato un taglio, se due o più altezze sono uguali, allora esse condivideranno ovviamente lo stesso taglio. Costruiamo ora l'albero in modo che ogni nodo rappresenti l'incontro del taglio con una parte dell'istogramma. Ogni volta che il taglio incontra una parte di istogramma in senso orizzontale, è aggiunto un nodo all'albero. Sul nodo è salvato anche l'intervallo di taglio, ovvero dove il taglio ha incontrato il primo bin e l'ultimo bin; tale informazione è fondamentale in quanto solo attraverso l'intervallo siamo in grado di inserire i nodi cercando il loro padre corretto (figura 2.21).

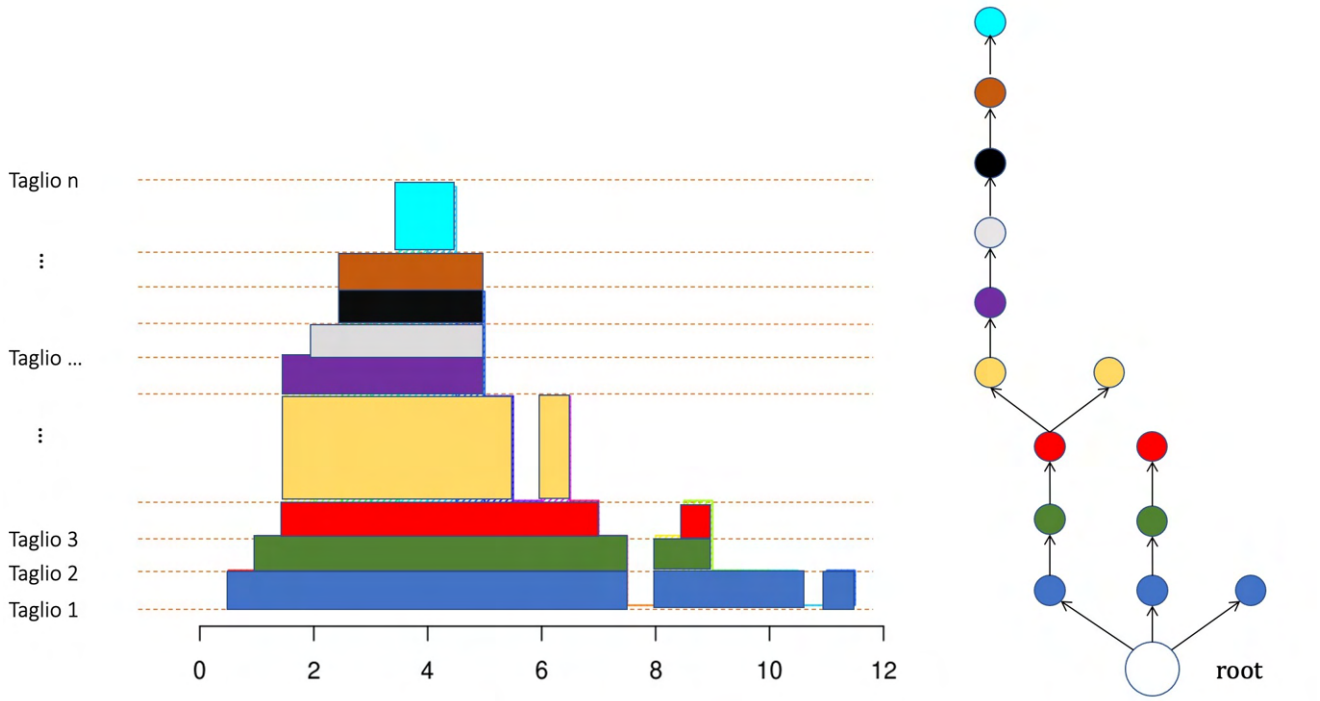



Figura 2.21: costruzione dell'albero

**Algorithm 3** createHierarchicalTree

```
1: Input: heights -> vettore delle densità, nbins -> numero di bins
2: Result: albero gerarchico pesato
3: tree ← new Node(name ="0", interval = (0, nbins-1), area = 0, clusters = 0)
4: parents ← new empty array
5: parents.append(tree)
6: listOfheightsToCheck ← sort(new Set(heights))
7: lastHeight ← = 0
8: for i:=0. . . len(listOfheightsToCheck) do
9:     if listOfheightsToCheck[i] != 0: then
10:        checkHeight ← listOfheightsToCheck[i]
11:        j ← 0
12:        while j < (nbins-1) do
13:            if heights[j] ≥ checkHeight then
14:                start ← j
15:                end ← nbins-1
16:                while j < (nbins-1) and heights[j] ≥ checkHeight do
17:                    j ← j+1
18:                    end ← j
19:                end while
20:                interval ← (start, end)
21:                area ← (end - start) * (listOfheightsToCheck[i] - lastHeight)
22:                addToParent(parents, interval, area, nbins)
23:            else
24:                j ← j+1
25:            end if
26:        end while
27:        lastHeight ← listOfheightsToCheck[i]
28:    end if
29:    sumOfDifferences ← sumOfDifferences + abs(hist[i+1] - hist[i])
30:    count ← count + 1
31: end for
32: return tree
```

**Algorithm 4** addToParent

```
1: Input: parents -> vettore dei nodi inseriti, interval -> tupla dell'intervallo del nodo
   che si vuole inserire, area -> valore dell'area sul nodo, nBins -> numero di bins
2: intervalChild ← interval
3: areaChild ← area
4: _parent_ ← correctParent(parents, intervalChild, nBins)
5: id ← _parent_.name
6: n_children ← len(_parent_.children)
7: newId ← id + "_" + str(n_children)
8: newNode ← new Tree(name=newId, interval = intervalChild, area = areaChild,
   clusters = 0, parent=_parent_)
9: parents.append(newNode)
```

**Algorithm 5** correctParent

```
1: Input: parents -> vettore dei nodi inseriti, intervalOfNode -> tupla dell'intervallo
   del nodo che si vuole inserire, nbins -> numero di bins
2: start ← 0
3: end ← nBins
4: parent ← Null
5: int_n_st ← intervalOfNode[0]
6: int_n_end ← intervalOfNode[1]
7: for i:=0 . . . len(parents) do
8:     intervalParent ← parents[i].interval
9:     intervalParentStart ← intervalParent[0]
10:    intervalParentEnd ← intervalParent[1]
11:    if int_n_st ≥ intervalParentStart and int_n_end ≤ intervalParentEnd then
12:        if int_n_st ≥ start and int_n_end ≤ end then
13:            start ← int_n_st
14:            end ← int_n_end
15:            parent ← parents[i]
16:        end if
17:    end if
18: end for
19: return parent
```

Il risultato ottenuto sarà simile al seguente

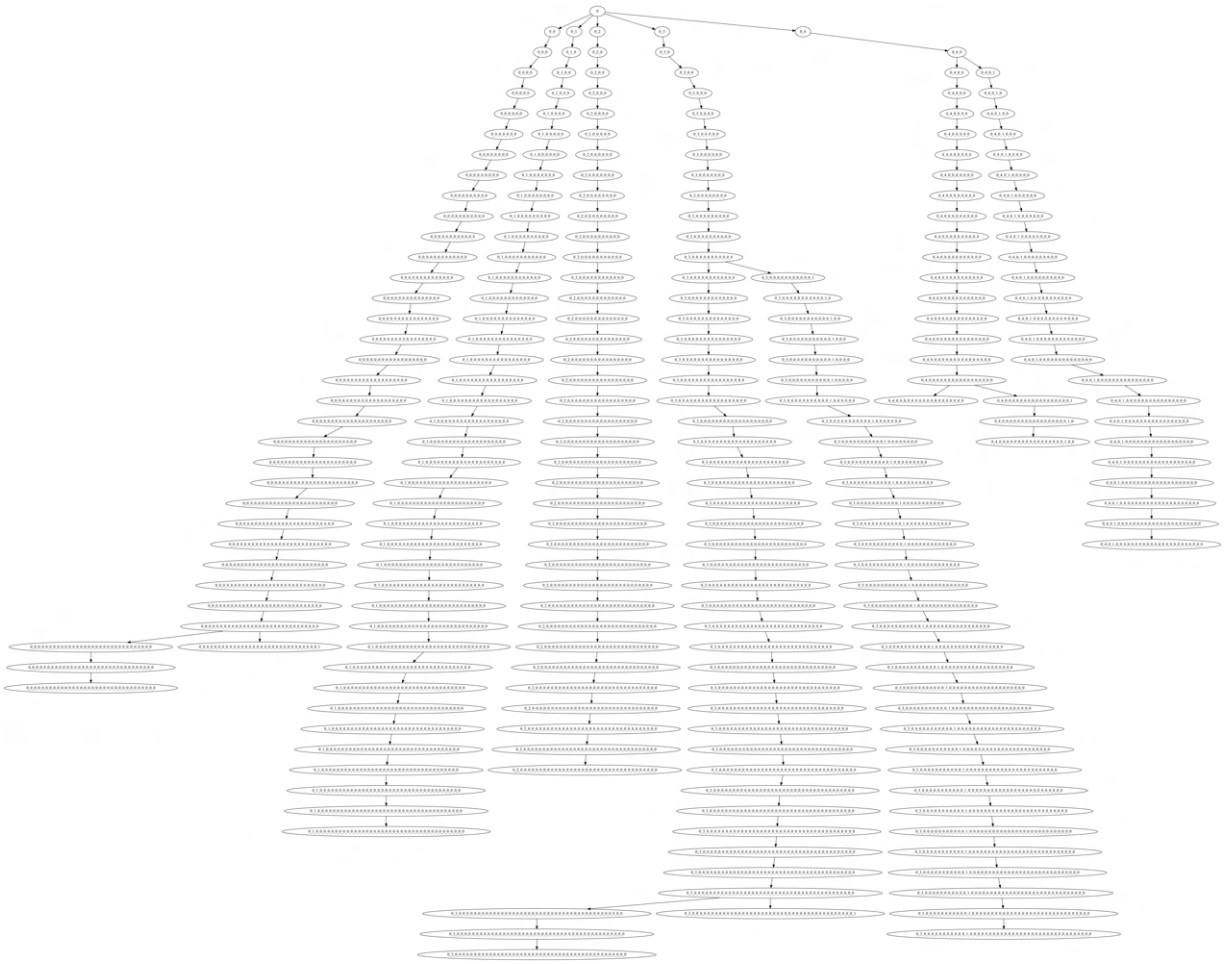


Figura 2.22: Risultato dell'algoritmo

Ogni nodo contiene al suo interno l'intervallo di appartenenza e l'area calcolata dall'incontro del taglio con l'istogramma. Se vogliamo trovare il numero di cluster presenti nell'albero appena costruito, è necessario implementare una logica che cerchi di estrapolare tale informazione attraverso la somma delle aree. Come già evidenziato precedentemente, un cluster è determinato dal fatto di avere un'area sull'istogramma, che termina con un picco. Se vi è una biforcazione e vogliamo sapere se si tratta di un cluster singolo oppure di più cluster, è necessario osservare le aree antecedenti al nodo e conseguenti. Se il totale della somma delle aree successive al nodo è superiore all'area tra le ultime due biforcazioni, allora avrà senso continuare la ricerca, altrimenti potremo già affermare che si tratta di un singolo cluster. Per risolvere il problema possiamo salvare su ogni nodo che presenta una biforcazione in due o più rami, il numero di cluster che potrebbero essere trovati. L'algoritmo seguente svolge questo compito.

**Algorithm 6** detectClusters

```
1: Input: tree -> nodo radice dell'albero
2: node ← tree
3: stack ← []
4: stack.append(node)
5: while len(stack) > 0 do
6:     currentNode ← stack.pop()
7:     if len(currentNode.children) > 1 then
8:         areaOfTheUpperPart ← currentNode.area
9:         parentNode ← currentNode.parent
10:        while parentNode != Null and len(parentNode.children) ≤ 1 do
11:            areaOfTheUpperPart ← areaOfTheUpperPart + parentNode.area
12:            parentNode ← parentNode.parent
13:        end while
14:        childArea ← 0.0
15:        for i:=0... len(currentNode.children) do
16:            childArea ← childArea + returnArea(currentNode.children[i])
17:        end for
18:        if childArea > areaOfTheUpperPart then
19:            currentNode.clusters ← len(currentNode.children)
20:        end if
21:    end if
22:    children ← currentNode.children
23:    for i:=0... len(children) do
24:        stack.append(children[len(children)-1 - i])
25:    end for
26: end while
27: return parent
```

**Algorithm 7** returnArea

```
1: Input: root -> nodo dell'albero dove partire a calcolare l'area
2: stack ← []
3: stack.append(root)
4: totArea ← 0.0
5: while len(stack) > 0 do
6:     currentNode ← stack.pop()
7:     totArea ← totArea + currentNode.area
8:     children ← currentNode.children
9:     for i:=0... len(children) do
10:        stack.append(children[len(children)-1 - i])
11:    end for
12: end while
13: return totArea
```

Computiamo per ogni nodo dell'albero la somma dell'area superiore e l'area inferiore. Per fare ciò utilizziamo l'algoritmo dfs (depth first search), che permette una ricerca in profondità all'interno di un albero. Ottenute le due aree le si confronta, se l'area superiore (quella dopo un taglio) è maggiore dell'area inferiore (la base dell'istogramma) allora si segna sul nodo il numero di biforcazioni (il numero di figli) che esso possiede. Facciamo questo poiché nella seconda parte utilizziamo il valore appena calcolato sui nodi per determinare il numero finale di cluster.

L'albero ora è composto da due tipi di nodi, i nodi con il parametro *cluster* settato a 0 e i nodi con il parametro *cluster* settato ad un valore superiore a 1, che sono quelli che hanno una biforcazione. Non resta altro che compattare l'albero in una forma compressa, dove i nodi sequenziali che non hanno nessuna biforcazione utile sono uniti tra loro, come nelle figure (2.23, 2.24).

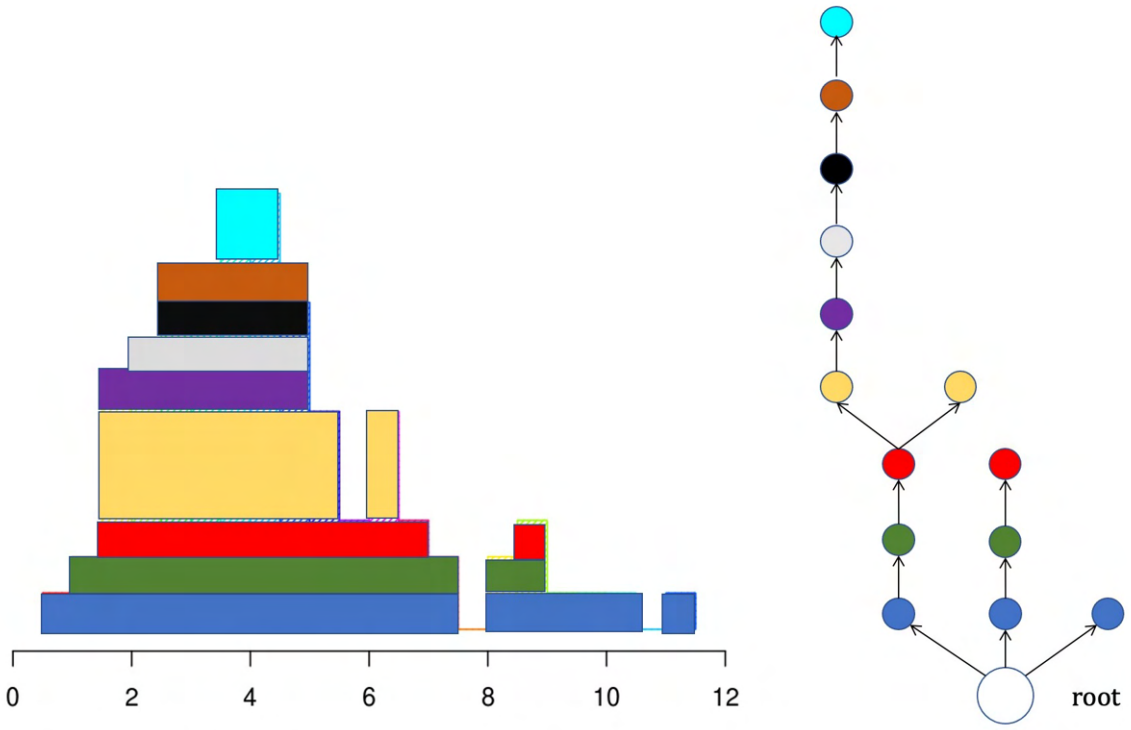



Figura 2.23: Albero dell'istogramma

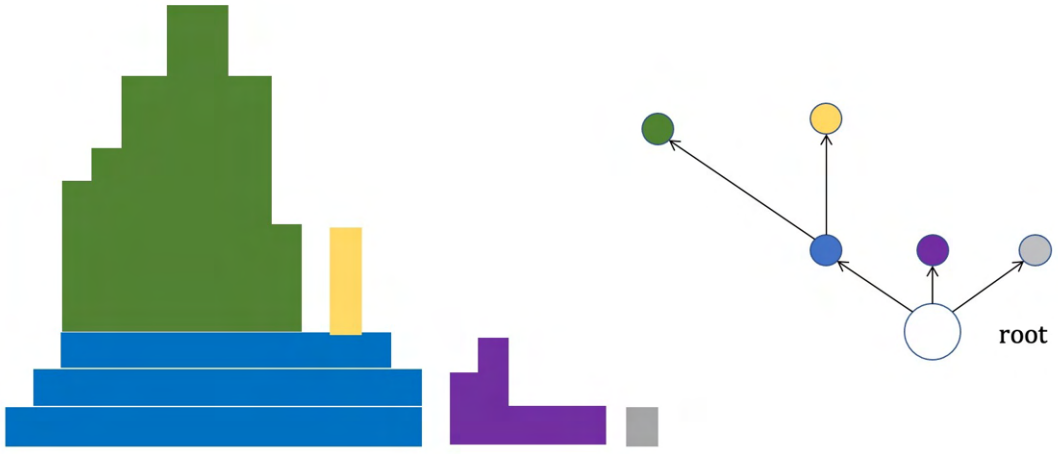



Figura 2.24: Albero compresso dell'istogramma

**Algorithm 8** createNewTreeOfClusters

```
1: Input: tree -> nodo radice dell'albero, nbins -> numero di bins
2: size_bins ← nbins-1
3: interval ← (0, size_bins)
4: newTree ← new Node(name="[" + str(0) + " - " + "2pi" + "]", interval = interval)
5: stackOfParents ← []
6: stackOfParents.append(newTree)
7: node ← tree
8: stack ← []
9: stack.append(node)
10: while len(stack) > 0 do
11:     currentNode ← stack.pop()
12:     if currentNode.clusters > 0 then
13:         children ← currentNode.children
14:         for i:=0... len(children) do
15:             nChildren ← children[i]
16:             _parent_ ← correctParent(stackOfParents, nChildren.interval, size_bins)
17:             newNode = new Node(name = newName, interval = nChildren.interval, parent = _parent_)
18:             stackOfParents.append(newNode)
19:         end for
20:     end if
21:     children = currentNode.children
22:     for i:=0... len(children) do stack.append(children[len(children) - 1 - i])
23:     end for
24: end while
25: return totArea
```

L'albero ottenuto è simile al seguente

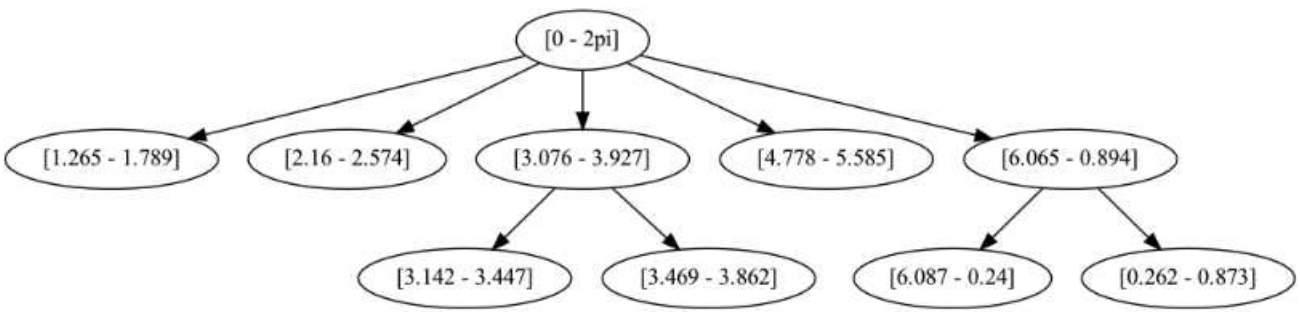



Figura 2.25: Risultato dell'algoritmo

Il risultato in forma compressa permette di capire immediatamente la struttura del'istogramma iniziale, in questo specifico caso sono stati computati 2 cluster con partenza nella base, rispettivamente con i valori di theta compresi tra [1.265, 1.789] e [2.16, 2.574]. L'albero ci mostra inoltre che sono stati trovati due cluster a partire dalla stessa origine [3.076 - 3.927]; stesso risultato in [6.065 - 0.894]. Infine un ulteriore cluster unico in [4.778 - 5.585]. Se esaminiamo nuovamente l'istogramma in figura (2.5) noteremo che le partizioni sono corrette e che l'algoritmo ha disposto attraverso un albero le informazioni di cui avevamo bisogno. Per trovare il numero $k$ di cluster presenti in modo automatico è sufficiente contare il numero di foglie dell'albero, che rappresentano esattamente i cluster presenti nell'istogramma, sia per numero che per forma che posizione. Per fare ciò è possibile eseguire nuovamente l'algoritmo di ricerca in profondità dfs, aumentando di un'unità un contatore ogni qualvolta è incontrato un nodo senza figli(una foglia dell'albero).

La fase 2 ritorna il numero $k$, il numero di cluster presenti nell'istogramma di densità e quindi dell'insieme di dati.

## 2.5 Terza fase

Siamo a conoscenza del numero di cluster presenti nell'insieme grazie alla procedura sviluppata nelle due sezioni precedenti. Non resta altro che cercare di etichettare i punti per dargli un significato semantico e poter sapere quali appartengono ad un determinato sottogruppo piuttosto che un altro. Cercare di definire una label per un gruppo di dati in un cerchio unitario oppure in un istogramma delle densità è un compito piuttosto complesso che tuttavia può essere risolto attraverso una tecnica ben nota dal nome Gaussian Mixture (GM). Il modello GM richiede in input i valori di theta e il valore k del numero di cluster presenti nei dati. Attraverso questi due elementi, è in grado di determinare il labeling dei dati e come questi sono posizionati, attraverso l'uso di una miscela di funzioni gaussiane.

### 2.5.1 Il modello

I modelli di miscela gaussiana sono un modello probabilistico per rappresentare sotto popolazioni normalmente distribuite all'interno di una popolazione complessiva. Questo tipo di tecnica è specialmente utilizzata quando si affrontano dati multimodali, che presentano quindi più di un "picco" nella distribuzione dei dati.

Come indica il nome, tale modello fa utilizzo ancora una volta delle distribuzioni gaussiane. Per definire una gaussiana si ha bisogno della media di tutti i dati, che rappresenta il centro della curva, e la deviazione standard, che descrive come sono

distribuiti i dati. La formula della distribuzione gaussiana è la seguente

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

dove $\mu$ è la media, $\sigma$ la deviazione standard. Grazie a tale formula, per un punto $x$ siamo in grado di calcolare $y$, la sua probabilità.

Nel caso in esame, abbiamo più di una distribuzione gaussiana, in particolare una per ciascun cluster, ovvero per ciascun picco nell'istogramma di densità. In altre parole possiamo affermare che, se abbiamo tre distribuzioni gaussiane GD1 GD2 GD3, aventi medie $\mu_1$ $\mu_2$ $\mu_3$ e varianza $\sigma_1$, $\sigma_2$, $\sigma_3$, allora per un dato insieme di punti, il modello GMM identificherà la probabilità di ciascun punto di appartenere a ciascuna di queste distribuzioni.

Poichè stiamo lavorando con i valori dell'istogramma di densità (un vettore monodimensionale) siamo avvantaggiati nella costruzione della mistura di gaussiane. In un modello unidimensionale il valore della probabilità è definito come

$$p(x) = \sum_{i=1}^{K} \phi_i \mathrm{N}(x|\mu_i, \sigma_i)$$

dove

$$\mathrm{N}(x|\mu_i, \sigma_i]) = \frac{1}{\sigma_i\sqrt{2\pi}} exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)$$

$\phi$ rappresenta il set di pesi assegnati a ciascun cluster. Il peso è un parametro indicativo dell'importanza del cluster, in particolare l'insieme dei pesi deve avere somma uguale a 1.

$$\sum_{i=1}^{K} \phi_i = 1$$

### 2.5.2 Stima dei parametri

Il modello di mistura gaussiana utilizza un algoritmo, chiamato expectation maximization (EM), per stimare i parametri esposti precedentemente, rispettivamente $\mu$, $\sigma$ e $\phi$. EM è una tecnica numerica per la stima del valore massimale di likelihood (maximum likelihood estimation) che lavora per iterazioni. Ad ogni iterazione, in particolare, il valore della stima massimale di likelihood dei dati cresce fino a giungere ad un massimo locale. L'algoritmo consiste di due step. Il primo step, conosciuto come l'expectation step (E step), calcola l'aspettativa dell'assegnamento di ciascuna componente $C_k$, ovvero per l'assegnamento a ciascun cluster di ogni punto $x_i \in X$, dati i parametri $\mu_k$, $\sigma_k$ e $\phi_k$. Il secondo step, conosciuto come maximization step (M

step), consiste nel massimizzare le aspettative calcolate nell'E step rispetto ai punti parametrici. L'obiettivo di questo step è quindi aggiornare i valori $\mu_k$, $\sigma_k$ e $\phi_k$.

L'intero processo iterativo continua fino a quando l'algoritmo non converge, emettendo una stima massima di likelihood.

Step di inizializzazione:

- Assegna randomicamente alcuni campioni del dataset iniziale $X = \{x_1, ..., x_n\}$ ai valori delle medie $\mu_1$,...,$\mu_k$. E.g. per K = 3 e N = 100, $\mu_1 = x_{45}$, $\mu_2 = x_{32}$, $\mu_3 = x_{10}$.
- Assegna a tutte le variabili della varianza stimate, $\overline{\sigma_1^2}$,...,$\overline{\sigma_k^2}$ il valore

$$\overline{\sigma_1^2}, ..., \overline{\sigma_k^2} = \frac{1}{N} \sum_{i=1} N(x_i - \overline{x})^2$$

  dove $\overline{x}$ è la media campionaria

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

- Assegna a tutti i valori dei pesi un valore $\phi_1, ..., \phi_k = \frac{1}{K}$

Expectation (E) step: $\forall i, k$

$$\gamma_{ik} = \frac{\phi_k N(x_i|\mu_k, \sigma_k)}{\sum_{j=1}^{K} \phi_j N(x_i|\mu_j, \sigma_j)}$$

dove $\gamma_{ik}$ è la probabilità che $x_i$ sia generata dalla componente $C_k$. Cosi $\gamma_{ik} = p(C_k|x_i, \phi, \mu, \sigma)$

Maximization (M) step: Usando il valore di $\gamma_{ik}$ calcolato precedentemente nell'E step, calcoliamo $\forall k$:

$$\phi_k = \sum_{i=1}^{k} \frac{\gamma_{ik}}{N}$$

$$\mu_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{\sum_{i=1}^{N} \gamma_{ik}}$$

$$\sigma_k^2 = \frac{\sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k)^2}{\sum_{i=1}^{N} \gamma_{ik}}$$

Una volta che l'algoritmo EM ha terminato l'esecuzione, il modello può essere usato per fare inferenza quale appunto il clustering. Utilizzando il teorema di Bayes e i parametri precedentemente stimati, è possibile assegnare la corretta componente (la label del cluster) a ciascun campione.

**Lemma 1** *Dati i parametri di un modello univariato, la probabilità che un dato punto x appartenga alla componente $C_i$ è calcolata usando il teorema di Bayes:*

$$p(C_i|x) = \frac{p(x, C_i)}{p(x)} = \frac{p(C_i)p(x|C_i)}{\sum_{j=1}^{K} p(C_j)p(x|C_j)} = \frac{\phi_i N(x|\mu_i\sigma_i)}{\sum_{j=1}^{K} \phi_j N(x|\mu_j, \sigma_j)}$$

A livello di codice, l'algoritmo di calcolo EM può essere implementato come segue:

**Algorithm 9** EM

```
1: Input: components -> numero k di cluster, theta -> vettore dei valori theta dalla fase 1
2: k ← components
3: weights ← new Array of 1's of size k
4: weights ← weights / k
5: means ← random.choice(theta, k)
6: variances ← random samples(size = k)
7: eps ← 1e-8
8: steps ← 100
9: for step:=0... 100 do
10:     likelihood ← []
11:     for j:=0... k do
12:         likelihood.append(pdf(theta, means[j], np.sqrt(variances[j])))
13:     end for
14:     likelihood ← array(likelihood)
15:     b = []
16:     for j:=0... k do
17:         for i:=0... k do
18:             tot ← tot + sum([likelihood[i] * weights[i] + eps
19:         end for
20:         b.append((likelihood[j] * weights[j]) / tot
21:         means[j] ← sum(b[j] * theta) / (sum(b[j]+eps))
22:         variances[j] ← sum(b[j] * square(theta - means[j])) / (sum(b[j]+eps))
23:         weights[j] ← mean(b[j])
24:     end for
25: end for
26: return means, variances, weights
```

**Algorithm 10** pdf

```
1: Input: theta, mean, variance
2: s1 ← 1/(sqrt(2* π *variance))
3: s2 ← exp(-(square(theta - mean)/(2*variance)))
4: return s1*s2
```

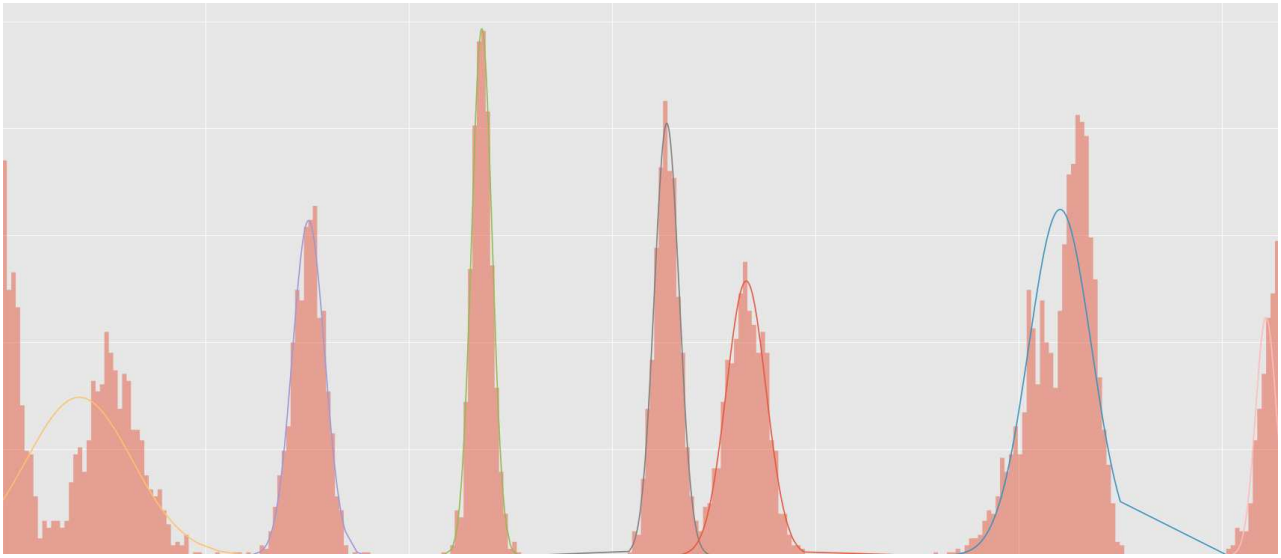


Figura 2.26: Risultato dell'algoritmo EM

In figura possiamo osservare il risultato finale del modello di misture. Le distribuzioni gaussiane permettono, tramite il calcolo delle probabilità di Bayes, di determinare la label corretta per ciascun punto dell'insieme iniziale.

Partendo dall'insieme iniziale dei dati sottoforma di matrice, siamo giunti alla conclusione. Abbiamo ottenuto il numero k di cluster presenti nell'insieme e la label associata a ciascun punto indicante il corretto cluster di appartenenza.

## 2.6 Il problema della matrice di similarità

Uno dei maggiori problemi che influisce sulla capacità e sulla velocità di computazione dell'algoritmo è la matrice di similarità. Il calcolo di tale matrice è infatti molto dispendioso e a volte nemmeno possibile. Si pensi ad esempio alla costruzione di una matrice di similarità a partire da un dataset di 50000 campioni. Tale matrice avrà 50000 righe e 50000 colonne per un totale di più di 2 miliardi di numeri. Se consideriamo ogni numero come un valore *float* di 32 bit, ovvero 4 byte otteniamo:

$$4\ byte * 2500000000 = 10000000000\ byte = 9,31\ GigaByte$$

Tale valore eccede la capienza RAM della maggior parte dei portatili in distribuzione; se poi si considera che il valore è destinato a crescere esponenzialmente, allora la matrice di similarità non può essere essenzialmente usata, in questo stato, per dataset di grandi dimensioni.

Tuttavia non è sempre necessario avere a disposizione tutti i valori presenti all'interno della matrice nello stesso momento, come in questo caso. A volte è necessario

un singolo valore oppure una singola riga. Possiamo quindi calcolare partizioni della la matrice di similarità ogni qualvolta ne necessitiamo l'utilizzo.

Dato un dataset A sotto forma di matrice, di grandezza $n$ in $p$ dimensioni, il calcolo della matrice di similarità $D$ tramite *pairwise euclidean distances*, la relazione di similarità da noi utilizzata, avviene nel seguente modo:

$$A_{n\times p} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{np} \end{bmatrix}$$

$$D_{n\times n} = \begin{bmatrix} dist(a_{1,1\ldots p}, a_{1,1\ldots p}) & dist(a_{1,1\ldots p}, a_{2,1\ldots p}) & \cdots & dist(a_{1,1\ldots p}, a_{n,1\ldots p}) \\ dist(a_{2,1\ldots p}, a_{1,1\ldots p}) & dist(a_{2,1\ldots p}, a_{2,1\ldots p}) & \cdots & dist(a_{2,1\ldots p}, a_{n,1\ldots p}) \\ \vdots & \vdots & \ddots & \vdots \\ dist(a_{n,1\ldots p}, a_{1,1\ldots p}) & dist(a_{n,1\ldots p}, a_{2,1\ldots p}) & \cdots & dist(a_{n,1\ldots p}, a_{n,1\ldots p}) \end{bmatrix}$$

dove $dist(a_{x,x..p}, a_{h,h\ldots p}) = \sqrt{res}$:

$$\begin{aligned} res = dot(a_x, a_{x+1}, ..., a_{x+p}; a_x, a_{x+1}, ..., a_{x+p}) \\ - 2 * dot(a_x, a_{x+1}, ..., a_{x+p}; a_h, a_{h+1}, ..., a_{h+p}) \\ + dot(a_h, a_{h+1}, ..., a_{h+p}; a_h, a_{h+1}, ..., a_{h+p}) \end{aligned} \quad (2.7)$$

L'opeatore *dot* rappresenta il *dot product*, calcolato come:

$$dot(\mathbf{a}; \mathbf{b}) = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

In caso di dataset, la cui matrice di similarità supera la grandezza della RAM disponibile, andremo a calcolare singoli valori della matrice oppure singole righe, attraverso la definizione $dist(a_{x,x..p}, a_{h,h\ldots p})$.

### 2.6.1 Utilizzo dell'accelerazione hardware

Per quanto il calcolo parziale della matrice di similarità sia logicamente corretto, non lo è per quanto riguarda le prestazioni. Il ricalcolo continuo di porzioni di numeri, più e più volte, comporta una perdita computazionale immane, fino a raggiungere stadi di non terminazione in tempi ragionevoli. Anche nel migliore dei casi, la perdita di prestazioni non può essere compensata, tuttavia è possibile alleggerirla, tramite

l'utilizzo dell'accelerazione hardware. E' possibile disegnare un kernel CUDA, tale da parallelizzare il calcolo delle righe.

Supponiamo di voler determinare una riga della matrice di similarità $D_{n,n}$, è allora possibile definire una griglia che calcoli ogni punto della riga parallelamente. Lo stesso calcolo può poi essere scomposto sui *dot product*, attraverso il parallelismo dinamico, per ottenere una parallelizzazione ancora più efficiente.

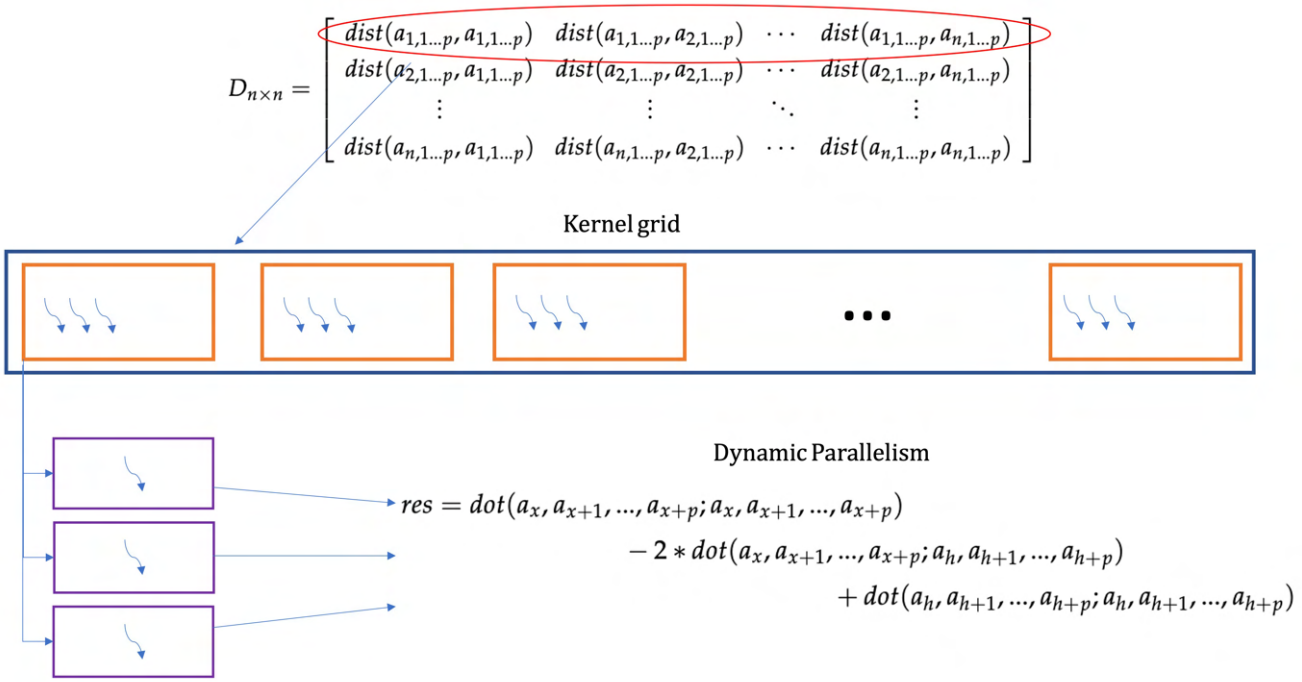



Figura 2.27: kernel schema

# Capitolo 3

# Simulazioni numeriche

Quando si parla di analisi di algoritmi di clustering, è necessario ribadire che non esiste, e probabilmente non esisterà mai, un unico algoritmo di clustering migliore di tutti gli altri. Ogni algoritmo, a seconda di come è stato costruito, presenta dei vantaggi su certi particolari insiemi di dati e svantaggi in altre occasioni. Un approccio comune per valutare un algoritmo di clustering ed ottenere una stima del suo funzionamento rispetto altre procedure, è quello di confrontare i risultati ottenuti con i risultati originali del database, ovvero i risultati creati manualmente dagli esperti del settore dell'insieme dei dati.

## 3.1 Benchmark Suite

La suite di test che utilizzeremo è composta da un totale di più di 200 dataset, disposti in otto batterie. Ogni batteria prende in esame casi differenti, alcuni artificiali e altri reali, generati per testare gli algoritmo di clustering. I parametri di ciascun dataset sono:

- nome
- n = numero di campioni presenti nel dataset
- d = dimensioni, numero di features per ciascun campione
- k = numero di cluster, sottogruppi, presenti nel database
- noise points = punti rumorosi, dati che non appartengono a nessun cluster
- g = distribuzione delle dimensioni dei cluster tramite indice di Gini, g = 0 significa che tutti i cluster sono costituiti dallo stesso numero di punti

Lista dei dataset:

- WUT
  22 dataset in $\mathcal{R}^2$ e $\mathcal{R}^3$ dall'Università di Tecnologia di Varsavia, facoltà di matematica e Scienza dell'informazione.

Tabella 3.1: wut

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 1 | circles | 4000 | 2 | 4 | 0 | 0 |
| 2 | cross | 2000 | 2 | 4 | 0 | 0 |
| 3 | UM | 2500 | 2 | 10 | 0 | 0 |
| 4 | isolation | 9000 | 2 | 3 | 0 | 0 |
| 5 | labirynth | 3546 | 2 | 6 | 0 | 0.5 |
| 6 | mk1 | 300 | 2 | 3 | 0 | 0 |
| 7 | mk2 | 1000 | 2 | 2 | 0 | 0 |
| 8 | mk3 | 600 | 3 | 3 | 0 | 0 |
| 9 | mk4 | 1500 | 3 | 3 | 0 | 0 |
| 10 | olympic | 5000 | 2 | 5 | 0 | 0 |
| 11 | smile | 1000 | 2 | 6 | 0 | 0.4 |
| 12 | stripes | 5000 | 2 | 2 | 0 | 0 |
| 13 | trajectories | 10000 | 2 | 4 | 0 | 0 |
| 14 | trapped_lovers | 5000 | 3 | 3 | 0 | 0.4 |
| 15 | torosplashes | 400 | 2 | 2 | 0 | 0 |
| 16 | wInclows | 2977 | 2 | 5 | 0 | 0.58 |
| 17 | xi | 120 | 2 | 3 | 0 | 0.17 |
| 18 | rt2 | 120 | 2 | 3 | 0 | 0.17 |
| 19 | ra | 185 | 2 | 4 | 0 | 0.21 |
| 20 | zi | 192 | 2 | 3 | 0 | 0 |
| 21 | .2 | 900 | 2 | 5 | 0 | 0.58 |
| 22 | z.3 | 1000 | 2 | 4 | 0 | 0.33 |

- SIPU
  Batteria di 20 dataset diversi creati, compilati e mantenuti da P.Franti, i colleghi e gli studenti dell'Università della Finlandia

Tabella 3.2: sipu

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 1 | a1 | 3000 | 2 | 20 | 0 | 0 |
| 2 | a2 | 5250 | 2 | 35 | 0 | 0 |

*Continua nella prossima pagina*

Tabella 3.2: *(Continued)* Caption for the multi-page table

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 3 | a3 | 7500 | 2 | 50 | 0 | 0 |
| 4 | aggregation | 788 | 2 | 7 | 0 | 0.45 |
| 5 | compound | 399 | 2 | 6 | 0 | 0.44 |
| 6 | d31 | 3100 | 2 | 31 | 0 | 0 |
| 7 | flame | 240 | 2 | 2 | 0 | 0.28 |
| 8 | join | 373 | 2 | 2 | 0 | 0.48 |
| 9 | pathbased | 300 | 2 | 3 | 0 | 0.06 |
| 10 | r15 | 600 | 2 | 15 | 0 | 0 |
| 11 | s1 | 5000 | 2 | 15 | 0 | 0.03 |
| 12 | s2 | 5000 | 2 | 15 | 0 | 0.03 |
| 13 | s3 | 5000 | 2 | 15 | 0 | 0.03 |
| 14 | s4 | 5000 | 2 | 15 | 0 | 0.03 |
| 15 | spiral | 312 | 2 | 3 | 0 | 0.02 |
| 16 | unbalance | 6500 | 2 | 8 | 0 | 0.63 |

- FCPS

  9 set di dati dalla Fundamental Clustering Problem Suite proposta da A. Ultsch dell'Università di Marburg, Germania.

  Ogni set di dati è composto da 212–4096 osservazioni in 2–3 dimensioni.

Tabella 3.3: fcps

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 1 | atom | 800 | 3 | 2 | 0 | 0 |
| 2 | chainlink | 1000 | 3 | 2 | 0 | 0 |
| 3 | engytime | 4096 | 2 | 2 | 0 | 0 |
| 4 | hepta | 212 | 3 | 7 | 0 | 0.01 |
| 5 | sun | 400 | 2 | 3 | 0 | 0.25 |
| 6 | target | 770 | 2 | 6 | 0 | 0.79 |
| 7 | tam | 400 | 3 | 4 | 0 | 0 |
| 8 | twodiamonds | 800 | 2 | 2 | 0 | 0 |
| 9 | wingnut | 1016 | 2 | 2 | 0 | 0 |

- GRAVES

  10 set di dati sintetici discussi da D. Graves e W. Pedrycz

  Ogni set di dati è composto da 200-1050 osservazioni in 2 dimensioni. Originariamente, vengono forniti senza etichette di riferimento.

Tabella 3.4: graves

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 1 | dense | 200 | 2 | 2 | 0 | 0 |
| 2 | fuzzyx | 1000 | 2 | 5 | 0 | 0.06 |
| 3 | line | 250 | 2 | 2 | 0 | 0.6 |
| 4 | parabolic | 1000 | 2 | 2 | 0 | 0.02 |
| 5 | ring | 1000 | 2 | 2 | 0 | 0 |
| 6 | ring_noisy | 1050 | 2 | 2 | 43 | 0 |
| 7 | ring_outliers | 1030 | 2 | 5 | 0 | 0.71 |
| 8 | zigzag | 250 | 2 | 3 | 0 | 0.4 |
| 9 | zigzag_noisy | 300 | 2 | 3 | 38 | 0.41 |
| 10 | zigzag_outliers | 280 | 2 | 3 | 30 | 0.4 |

- OTHER
  Set di dati da più fonti:

  chameleon_t4_8k, chameleon_t5_8k, chameleon_t7_10k, chameleon_t8_8k sono correlati all'algoritmo CHAMELEON di G. Karypis et al.

  hdbscan – set di dati utilizzato per dimostrare gli output dell'algoritmo hdbscan

  iris, iris5 - famoso set di dati Iris e la sua versione sbilanciata

Tabella 3.5: other

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 1 | chameleon_t4_8k | 8000 | 2 | 6 | 761 | 0.25 |
| 2 | chameleon_t5_8k | 8000 | 2 | 6 | 1187 | 0.03 |
| 3 | chameleon_t7_10k | 10000 | 2 | 9 | 926 | 0.47 |
| 4 | chameleon_t8_8k | 8000 | 2 | 8 | 346 | 0.37 |
| 5 | hdbscan | 2309 | 2 | 6 | 510 | 0.18 |
| 6 | iris | 150 | 4 | 3 | 0 | 0 |
| 7 | iris5 | 105 | 4 | 3 | 0 | 0.43 |
| 8 | square | 1000 | 2 | 2 | 0 | 0 |

- UCI
  8 dataset ad alta dimensione disponibili presso l'UCI (Università della California, Irvine) Machine Learning Repository.

Tabella 3.6: uci

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 1 | ecoli | 336 | 7 | 8 | 0 | 0.65 |
| 2 | glass | 214 | 9 | 6 | 0 | 0.48 |
| 3 | ionosphere | 351 | 34 | 2 | 0 | 0.28 |
| 4 | sonar | 208 | 60 | 2 | 0 | 0.07 |
| 5 | statlog | 2310 | 19 | 7 | 0 | 0 |
| 6 | wdbc | 569 | 30 | 2 | 0 | 0.25 |
| 7 | wine | 178 | 13 | 3 | 0 | 0.13 |
| 8 | yeast | 1484 | 8 | 10 | 0 | 0.63 |

- G2MG
  Ogni set di dati è costituito da 2.048 osservazioni da due cluster gaussiani equidimensionati in dimensioni (i componenti sono campionati indipendentemente da una distribuzione normale). Sono modificati i valori di distribuzione tra i due diversi campionamenti.

Tabella 3.7: g2mg

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 1 | g2mg_2_10 | 2048 | 2 | 2 | 0 | 0 |
| 2 | g2mg_2_20 | 2048 | 2 | 2 | 0 | 0 |
| 3 | g2mg_2_30 | 2048 | 2 | 2 | 0 | 0 |
| 4 | g2mg_2_40 | 2048 | 2 | 2 | 0 | 0 |
| 5 | g2mg_2_50 | 2048 | 2 | 2 | 0 | 0 |
| 6 | g2mg_2_60 | 2048 | 2 | 2 | 0 | 0 |
| 7 | g2mg_2_70 | 2048 | 2 | 2 | 0 | 0 |
| 8 | g2mg_2_80 | 2048 | 2 | 2 | 0 | 0 |
| 9 | g2mg_2_90 | 2048 | 2 | 2 | 0 | 0 |
| 10 | g2mg_4_10 | 2048 | 4 | 2 | 0 | 0 |
| 11 | g2mg_4_20 | 2048 | 4 | 2 | 0 | 0 |
| 12 | g2mg_4_30 | 2048 | 4 | 2 | 0 | 0 |
| 13 | g2mg_4_40 | 2048 | 4 | 2 | 0 | 0 |
| 14 | g2mg_4_50 | 2048 | 4 | 2 | 0 | 0 |
| 15 | g2mg_4_60 | 2048 | 4 | 2 | 0 | 0 |
| 16 | g2mg_4_70 | 2048 | 4 | 2 | 0 | 0 |
| 17 | g2mg_4_80 | 2048 | 4 | 2 | 0 | 0 |
| 18 | g2mg_4_90 | 2048 | 4 | 2 | 0 | 0 |

*Continua nella prossima pagina*

Tabella 3.7: *(Continued)* Caption for the multi-page table

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... |
| 54 | g2mg_128_10 | 2048 | 128 | 2 | 0 | 0 |
| 55 | g2mg_128_20 | 2048 | 128 | 2 | 0 | 0 |
| 56 | g2mg_128_30 | 2048 | 128 | 2 | 0 | 0 |
| 57 | g2mg_128_40 | 2048 | 128 | 2 | 0 | 0 |
| 58 | g2mg_128_50 | 2048 | 128 | 2 | 0 | 0 |
| 59 | g2mg_128_60 | 2048 | 128 | 2 | 0 | 0 |
| 60 | g2mg_128_70 | 2048 | 128 | 2 | 0 | 0 |
| 61 | g2mg_128_80 | 2048 | 128 | 2 | 0 | 0 |
| 62 | g2mg_128_90 | 2048 | 128 | 2 | 0 | 0 |

- H2MG
  Due concentrati di tipo gaussiano di dimensioni uguali, con diffusione dipendente dalle dimensioni dei set di dati. Ogni dataset è composto da 2.048 osservazioni in 1, 2, . . . , 128 dimensioni. Ogni punto viene campionato dal centro del proprio nucleo, di raggio che segue la distribuzione gaussiana con un parametro di scala predefinito.

Tabella 3.8: h2mg

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 1 | h2mg_2_10 | 2048 | 2 | 2 | 0 | 0 |
| 2 | h2mg_2_20 | 2048 | 2 | 2 | 0 | 0 |
| 3 | h2mg_2_30 | 2048 | 2 | 2 | 0 | 0 |
| 4 | h2mg_2_40 | 2048 | 2 | 2 | 0 | 0 |
| 5 | h2mg_2_50 | 2048 | 2 | 2 | 0 | 0 |
| 6 | h2mg_2_60 | 2048 | 2 | 2 | 0 | 0 |
| 7 | h2mg_2_70 | 2048 | 2 | 2 | 0 | 0 |
| 8 | h2mg_2_80 | 2048 | 2 | 2 | 0 | 0 |
| 9 | h2mg_2_90 | 2048 | 2 | 2 | 0 | 0 |
| 10 | h2mg_4_10 | 2048 | 4 | 2 | 0 | 0 |
| 11 | h2mg_4_20 | 2048 | 4 | 2 | 0 | 0 |
| 12 | h2mg_4_30 | 2048 | 4 | 2 | 0 | 0 |
| 13 | h2mg_4_40 | 2048 | 4 | 2 | 0 | 0 |
| 14 | h2mg_4_50 | 2048 | 4 | 2 | 0 | 0 |
| 15 | h2mg_4_60 | 2048 | 4 | 2 | 0 | 0 |

*Continua nella prossima pagina*

Tabella 3.8: *(Continued)* Caption for the multi-page table

| | dataset | n | d | k | noise points | g |
|---|---|---|---|---|---|---|
| 16 | h2mg_4_70 | 2048 | 4 | 2 | 0 | 0 |
| 17 | h2mg_4_80 | 2048 | 4 | 2 | 0 | 0 |
| 18 | h2mg_4_90 | 2048 | 4 | 2 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... |
| 54 | h2mg_128_10 | 2048 | 128 | 2 | 0 | 0 |
| 55 | h2mg_128_20 | 2048 | 128 | 2 | 0 | 0 |
| 56 | h2mg_128_30 | 2048 | 128 | 2 | 0 | 0 |
| 57 | h2mg_128_40 | 2048 | 128 | 2 | 0 | 0 |
| 58 | h2mg_128_50 | 2048 | 128 | 2 | 0 | 0 |
| 59 | h2mg_128_60 | 2048 | 128 | 2 | 0 | 0 |
| 60 | h2mg_128_70 | 2048 | 128 | 2 | 0 | 0 |
| 61 | h2mg_128_80 | 2048 | 128 | 2 | 0 | 0 |
| 62 | h2mg_128_90 | 2048 | 128 | 2 | 0 | 0 |

## 3.2 Divisione in classi

Sono ora esposti i risultati sperimentali dell'algoritmo CircleClustering. La struttura è per batterie, dove ognuna di esse è composta da un sottoinsieme di dataset di quelli presenti nella sezione precedente. Poichè non è possibile inserire tutti i risultati sperimentali ottenuti, a causa del loro numero e della loro grandezza, sono inseriti solo i risultati più evidenti e interessanti da un punto di vista scientifico. Per ogni dataset è presentato il labeling originale, seguito dal labeling ottenuto tramite il CircleClustering, seguito a sua volta dalle label prodotte da altri algoritmi di clustering tra cui:

* KMeans
* Dbscan
* Optics
* Spectral Clustering
* Agglomerative Clustering (Ward)
* Meanshift
* Affinity Propagation
* Birch

Per confrontare le label prodotte da un determinato algoritmo di clustering, rispetto le label originali, esistono più modi. Tra gli indici più utilizzati troviamo sicuramente Adjusted Rand index (ARI) e Adjusted Mutual Information Score (AMI).

### 3.2.1 Adjusted Rand index (ARI)

Per poter capire correttamente l'Adjusted Rand index, è prima necessario dare una definizione del Rand Index (RI).

Dato un insieme di $n$ elementi $\mathcal{S} = \{o_1, ..., o_n\}$ e due partizioni di $\mathcal{S}$ da confrontare, $\mathcal{X} = \{X_1, ..., X_r\}$, una partizione di $\mathcal{S}$ in $r$ sottoinsiemi, e $\mathcal{Y} = \{Y_1, ..., Y_s\}$, una partizione di S in $s$ sottoinsiemi, definiamo:

- a, il numero di coppie di elementi in $\mathcal{S}$ che sono nello stesso sottoinsieme $\mathcal{X}$ e nello stesso sottoinsieme $\mathcal{Y}$
- b, il numero di coppie di elementi in $\mathcal{S}$ che sono in diversi sottoinsiemi in $\mathcal{X}$ e in diversi sottoinsiemi in $\mathcal{Y}$
- c, il numero di coppie di elementi in $\mathcal{S}$ che sono nello stesso sottoinsieme in $\mathcal{X}$ e in diversi sottoinsiemi in $\mathcal{Y}$
- d, il numero di coppie di elementi in $\mathcal{S}$ che sono in diversi sottoinsiemi in $\mathcal{X}$ e che sono nello stesso sottoinsieme in $\mathcal{Y}$

Il Rand Index (RI) vale:

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}}$$

Similmente, si può vedere il Rand Index come una misura della percentuale di decisioni corrette fatte dall'algoritmo. Può essere quindi calcolato come segue:

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

dove TP è il numero di veri positivi, TN è il numero di veri negativi, FP è il numero di falsi positivi e FN è il numero di falsi negativi.

Adjusted Rand index (ARI) è una misura simile a Rand Index, ma che tiene conto del numero di permutazioni che due insiemi di punti possono generare. Dato un insieme $\mathcal{S}$ di $n$ elementi, e due partizioni di clustering di tali elementi, rispettivamente $X = \{X_1, X_2, \ldots, X_r\}$ e $Y = \{Y_1, Y_2, \ldots, Y_s\}$, la sovrapposizione tra $X$ e $Y$ può essere riassunta in una tabella di contingenza $\left[n_{ij}\right]$ dove ogni entrata $n_{ij}$ denota il numero di oggetti in comune tra $X_i$ e $Y_j$ : $n_{ij} = |X_i \cap Y_j|$

| $X \diagdown Y$ | $Y_1$ | $Y_2$ | $\cdots$ | $Y_s$ | sums |
|---|---|---|---|---|---|
| $X_1$ | $n_{11}$ | $n_{12}$ | $\cdots$ | $n_{1s}$ | $a_1$ |
| $X_2$ | $n_{21}$ | $n_{22}$ | $\cdots$ | $n_{2s}$ | $a_2$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| $X_r$ | $n_{r1}$ | $n_{r2}$ | $\cdots$ | $n_{rs}$ | $a_r$ |
| sums | $b_1$ | $b_2$ | $\cdots$ | $b_s$ | |

Figura 3.1: tabella di contingenza

ARI è quindi definito come:

$$Adjusted\_Rand\_Index = \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex}$$

Ovvero

$$Adjusted\_Rand\_Index = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}$$

### 3.2.2 Adjusted Mutual Information Score (AMI)

In probabilità e statistica, la mutual information (MI) di due variabili casuali è una misura della dipendenza reciproca tra le due variabili. Più precisamente, quantifica la "quantità di informazione" ottenuta su una variabile casuale osservando l'altra variabile presa in considerazione. Il concetto di mutua informazione è strettamente legato a quello di entropia di una variabile casuale. L'adjusted mutual information (AMI) rappresenta una variazione della mutua informazione che può essere usata per misurare prodotti del clustering.

Dato un insieme S di N elementi S = $s_1, s_2, ..., s_n$, si considerino due partizioni di $S$, rispettivamente $U = \{U_1, U_2, ..., U_R\}$ con R cluster, e $V = \{V_1, V_2, ..., V_c\}$ con C clusters. Si presume che le partizioni siano hard cluster; ovvero che un punto sia assegnato ad una e una sola partizione:

$$U_i \cap U_j = V_i \cap V_j$$

per ogni $i \neq j$

$$\cup_{i=1}^{R} U_i = \cup_{j=1}^{C} V_j = S$$

La mutua informazione della sovrapposizione tra U e V può essere riassunta nella forma di una tabella di contingenza $M = [n\_ij]\_j = 1 \dots C^{i=1 \dots R}$, dove $n_{ij}$ denota il

numero di oggetti che sono comuni ai cluster $U_i e V_j$.

$$n_{ij} = \left|U_i \cap V_j\right|$$

Supponiamo che un oggetto sia scelto a caso da $S$; la probabilità che l'oggetto appartenga al cluster $U_i$ è:

$$P_U(i) = \frac{|U_i|}{N}$$

L'entropia associata con il partizionamento $U$ è

$$H(U) = -\sum_{i=1}^{R} P_U(i) \log P_U(i)$$

$H(U)$ è non negativa e assume il valore 0 solo quando non c'è incertezza riguardo l'appartenenza al cluster di un oggetto. Allo stesso modo, l'entropia del clustering può essere calcolata come:

$$H(V) = -\sum_{j=1}^{C} P_V(j) \log P_V(j)$$

dove $P_V(j) = |V_j|/N$. La mutua informazione (MI) tra due partizioni è allora

$$MI(U,V) = \sum_{i=1}^{R}\sum_{j=1}^{C} P_{UV}(i,j) \log \frac{P_{UV}(i,j)}{P_U(i)P_V(j)}$$

dove $P_{UV}(i,j)$ denota la probabilità che un punto appartenga a entrambi i cluster $U_i$ in $U$ e $V_j$ in $V$:

$$P_{UV}(i,j) = \frac{|U_i \cap V_j|}{N}$$

Come il Rand Index, il valore di base della mutua informazione tra due raggruppamenti casuali non assume un valore costante e tende ad essere maggiore quando le due partizioni hanno un numero maggiore di cluster (con un numero fisso di elementi dell'insieme N). Si può dimostrare che la Expected mutual information tra due clustering casuali è:

$$E\{MI(U,V)\} = \sum_{i=1}^{R}\sum_{j=1}^{C} \sum_{n_{ij}=(a_i+b_j-N)^+}^{\min(a_i,b_j)} \frac{n_{ij}}{N} \log\left(\frac{N \cdot n_{ij}}{a_i b_j}\right)$$

$$\times$$

$$\frac{a_i! b_j! (N-a_i)! (N-b_j)!}{N! n_{ij}! (a_i - n_{ij})! (b_j - n_{ij})! (N - a_i - b_j + n_{ij})!}$$

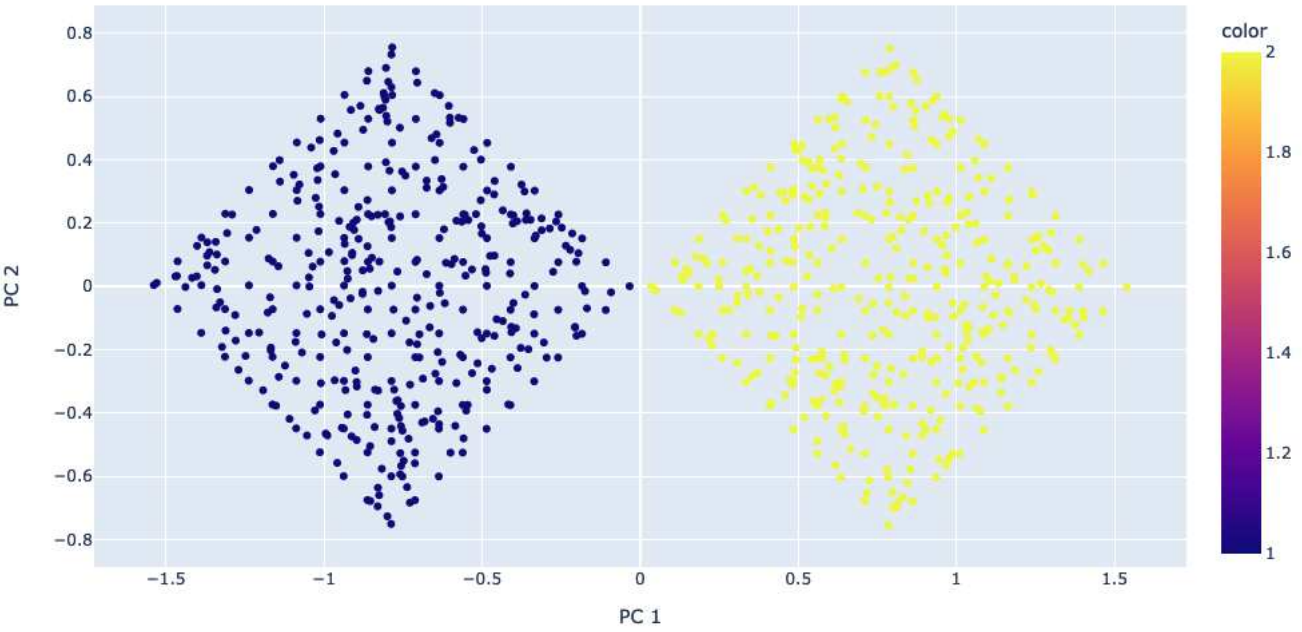
La misura adeguata per l'informazione reciproca può quindi essere definita come:

$$AMI(U,V) = \frac{MI(U,V) - E\{MI(U,V)\}}{\max\{H(U),H(V)\} - E\{MI(U,V)\}}$$

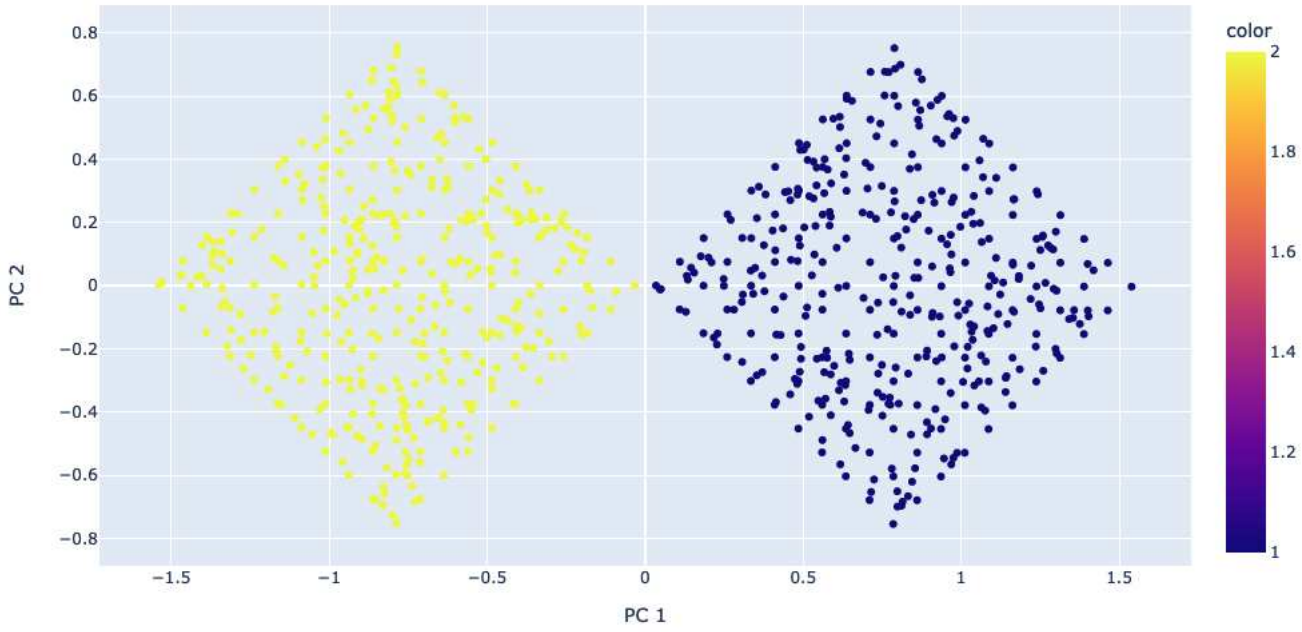
### 3.2.3 Risultati sperimentali

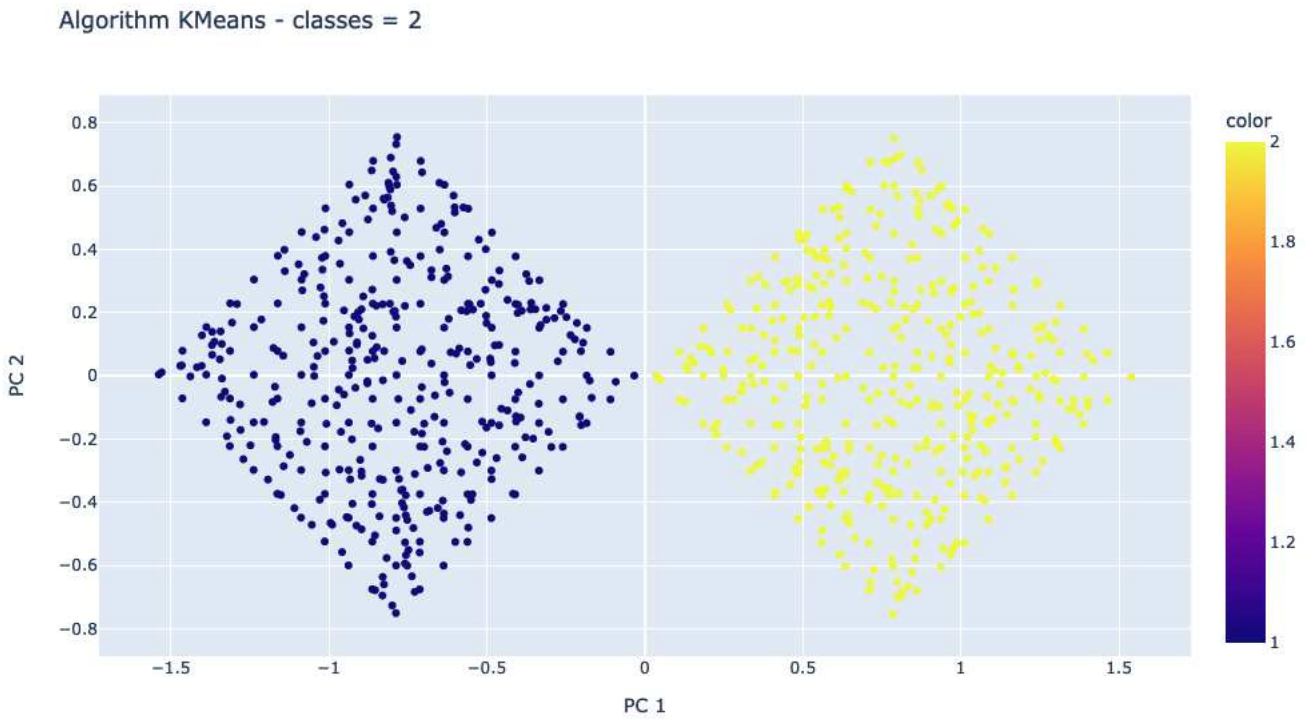
**Batteria: fcps**

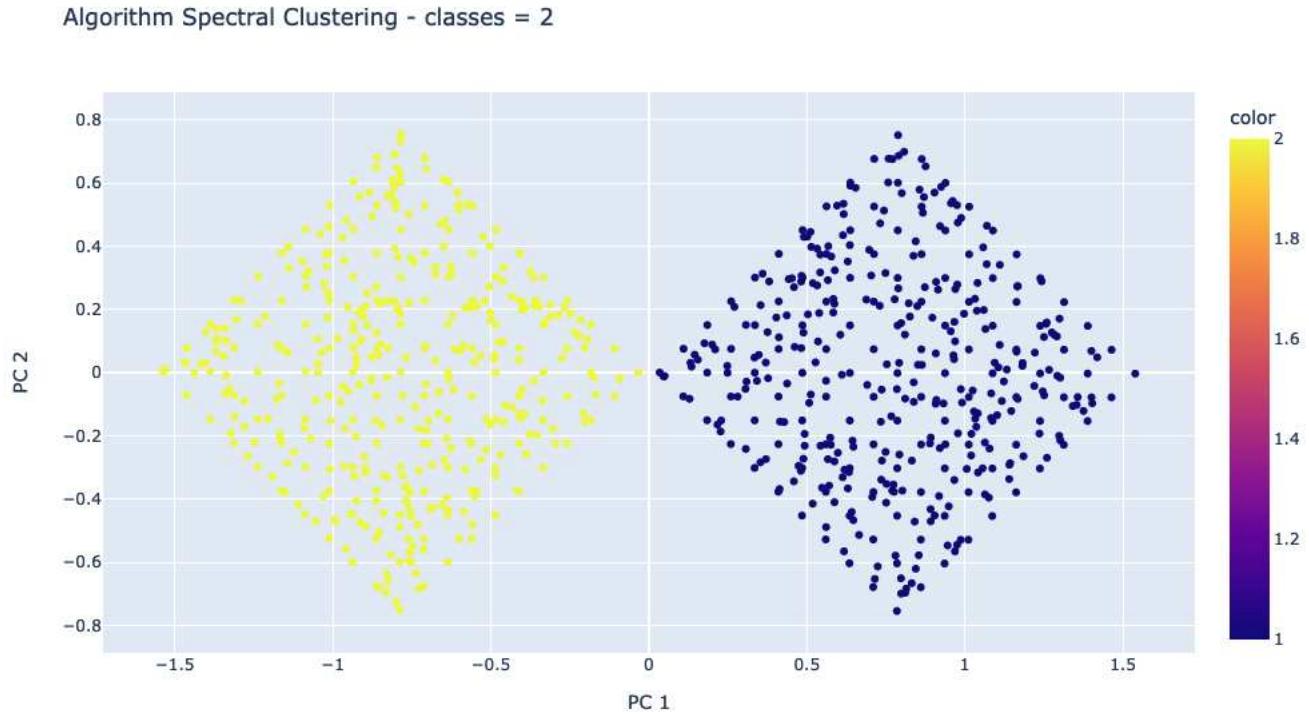
- Dataset: target
  Samples = 800, Features = 2, Classes = 2

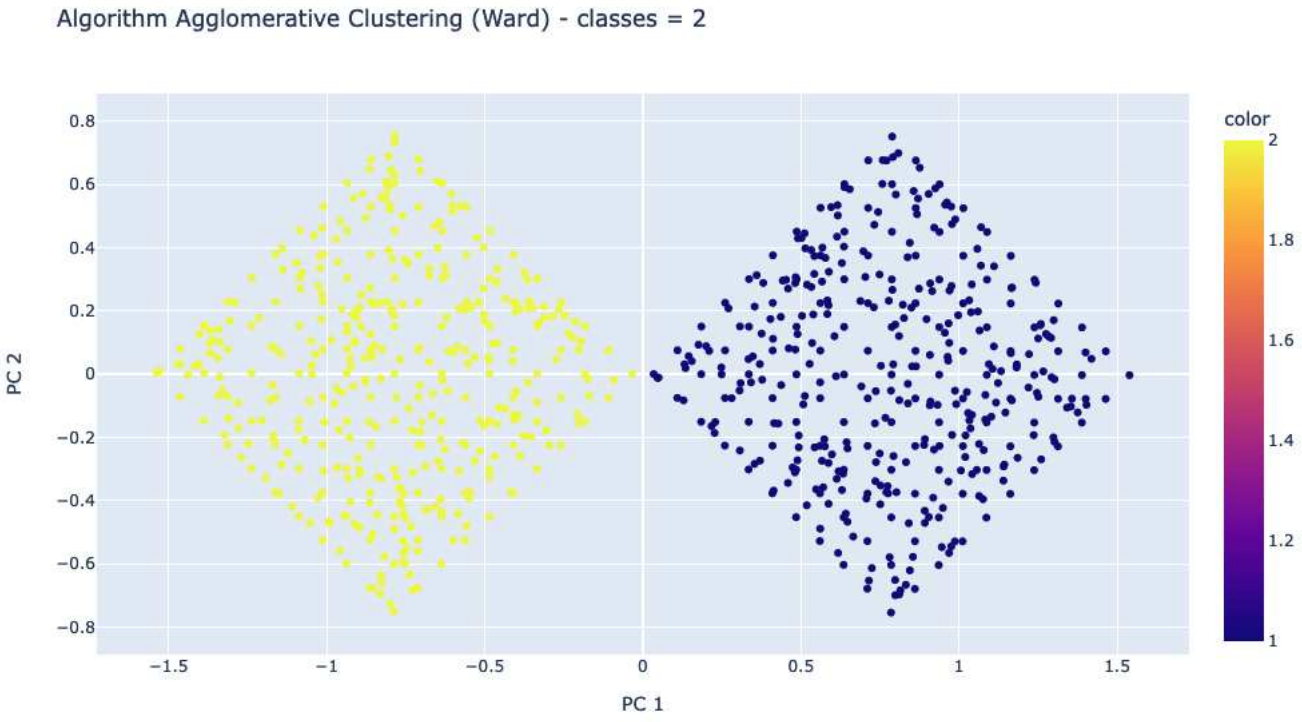


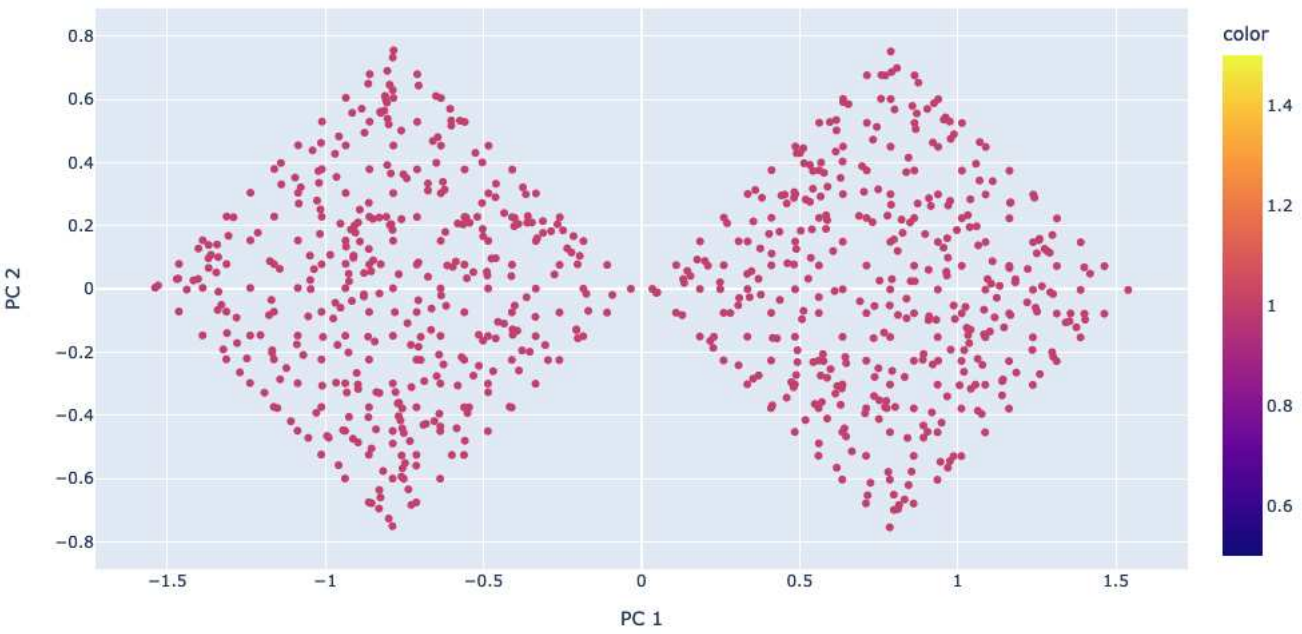

Algorithm CircleClustering - classes = 2

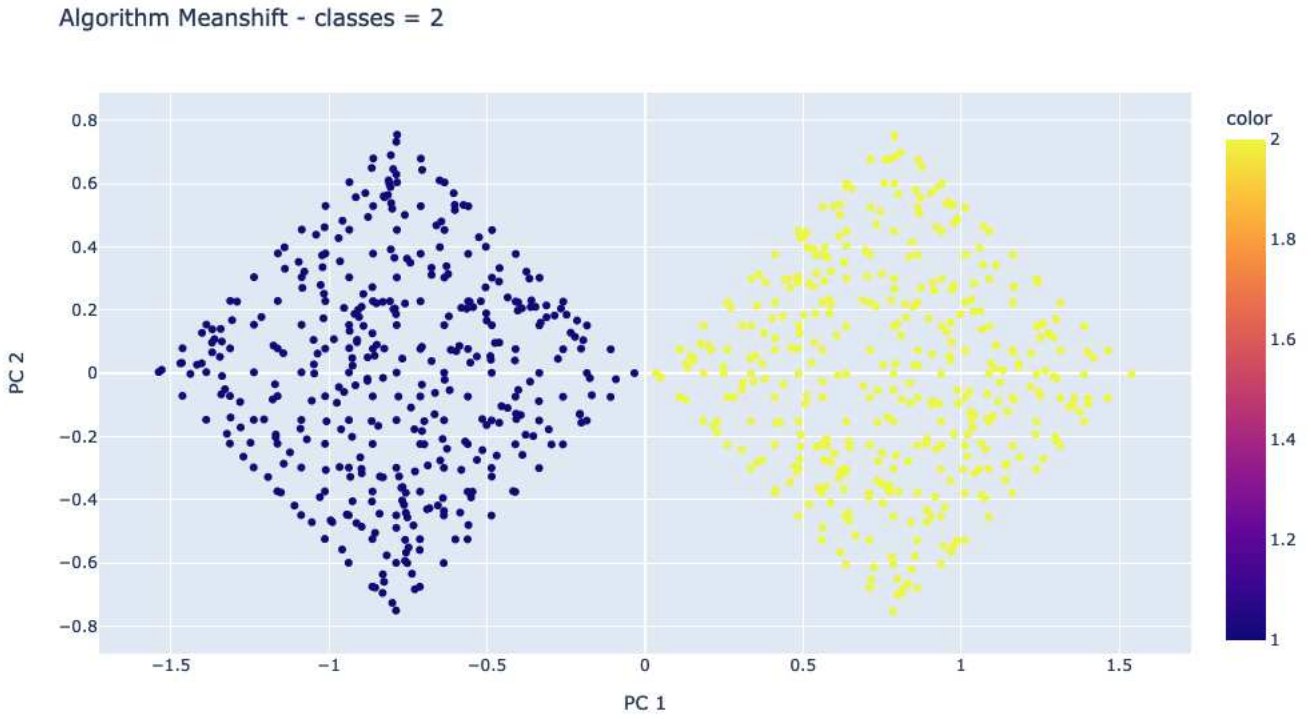


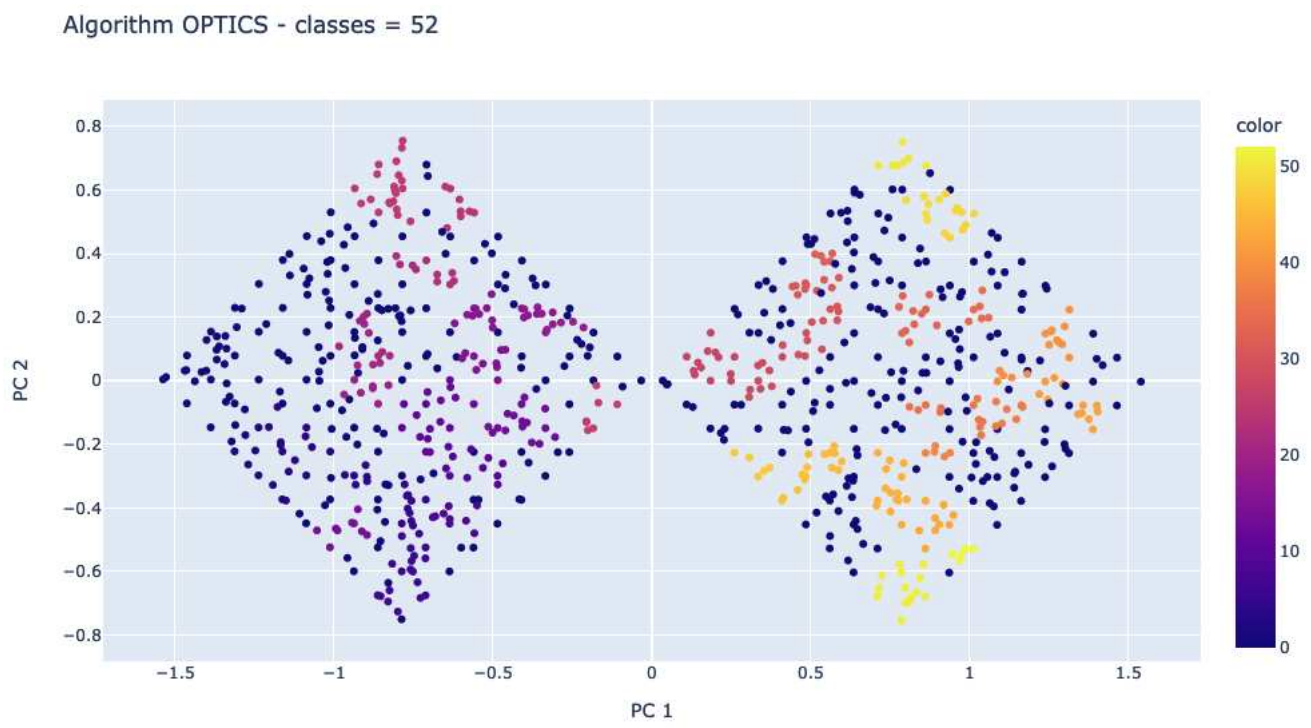

Algorithm KMeans - classes = 2
PC 2
PC 1
color
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1.5
−1
−0.5
0
0.5
1
1.5
2
1.8
1.6
1.4
1.2
1


Algorithm Spectral Clustering - classes = 2
PC 2
PC 1
color
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1.5
−1
−0.5
0
0.5
1
1.5
2
1.8
1.6
1.4
1.2
1

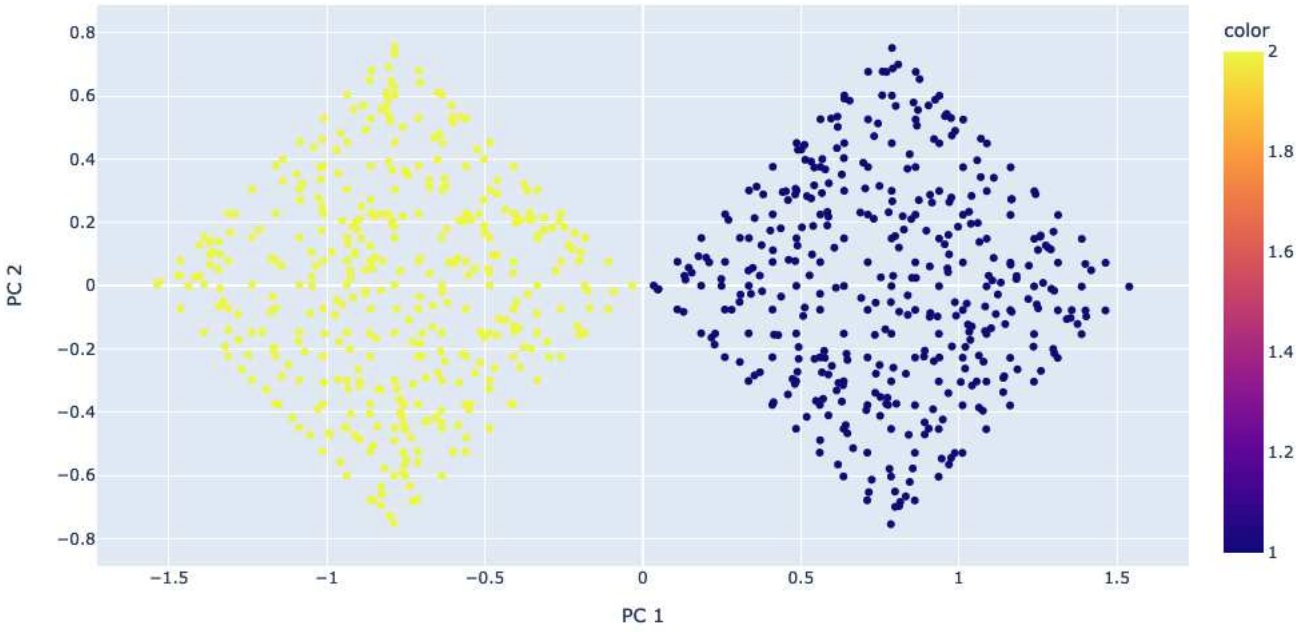

Algorithm Agglomerative Clustering (Ward) - classes = 2
color
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
PC 2
−1.5
−1
−0.5
0
0.5
1
1.5
PC 1


Algorithm DBSCAN - classes = 1
color
1.4
1.2
1
0.8
0.6
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
PC 2
−1.5
−1
−0.5
0
0.5
1
1.5
PC 1


Algorithm Meanshift - classes = 2
PC 2
PC 1
color
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1.5
−1
−0.5
0
0.5
1
1.5
2
1.8
1.6
1.4
1.2
1


Algorithm OPTICS - classes = 52
PC 2
PC 1
color
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1.5
−1
−0.5
0
0.5
1
1.5
50
40
30
20
10
0

Algorithm Birch - classes = 2




Algorithm Affinity Propagation - classes = 20

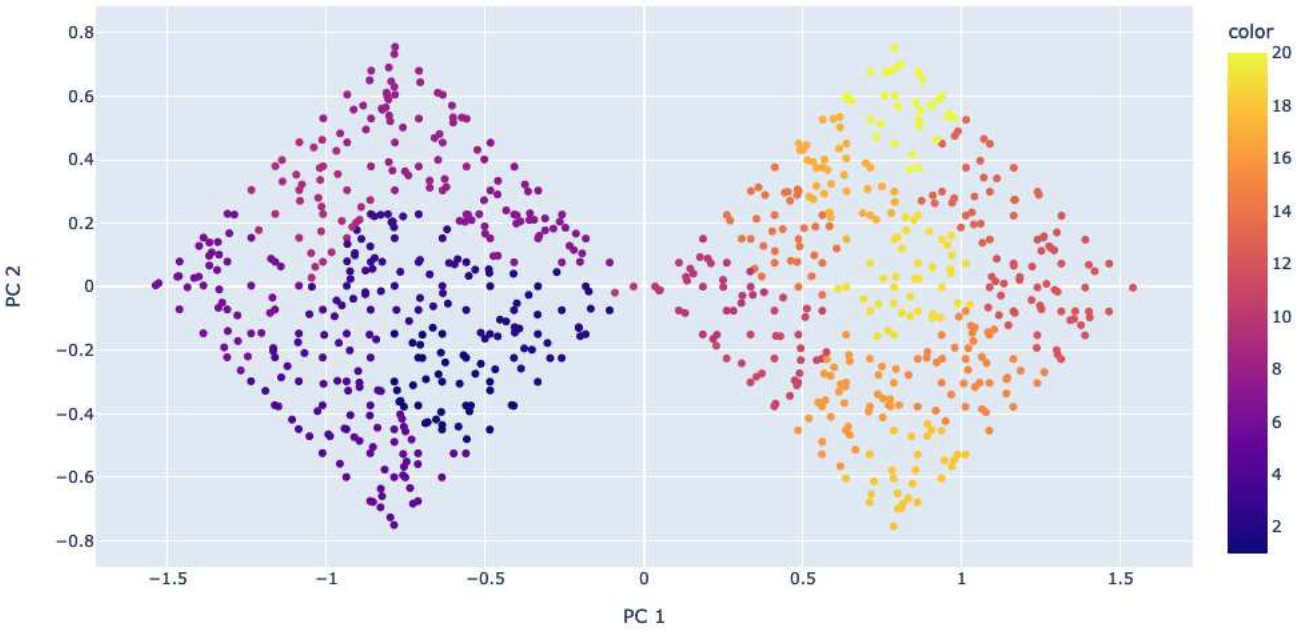
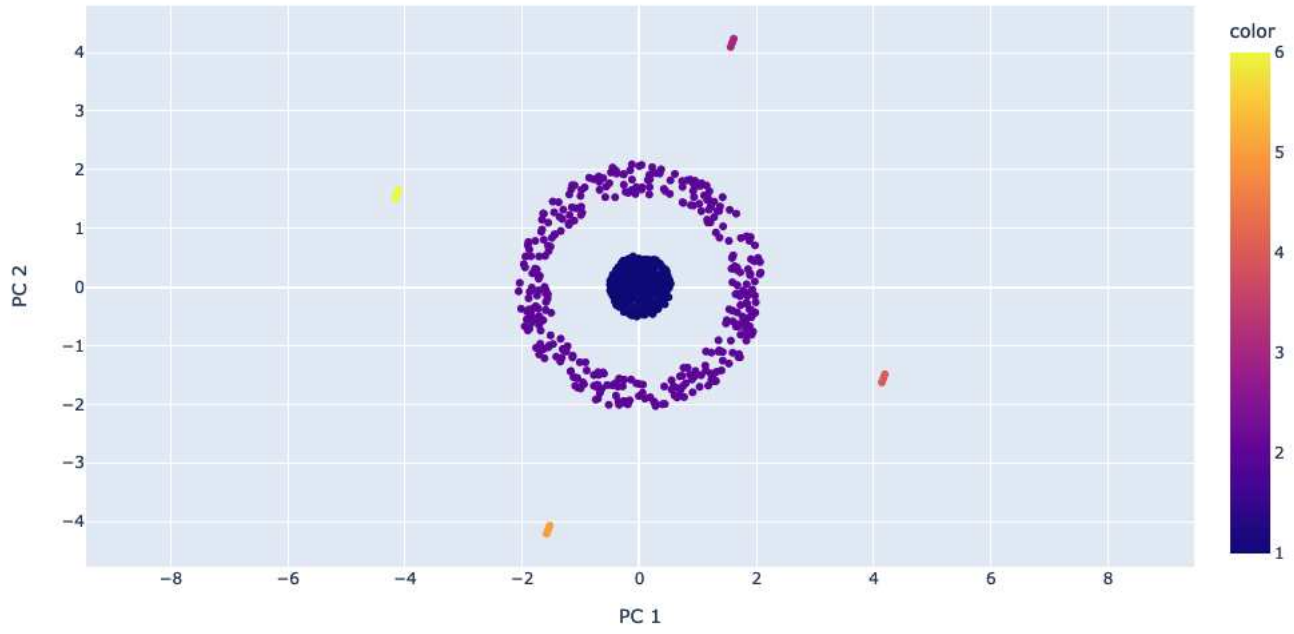


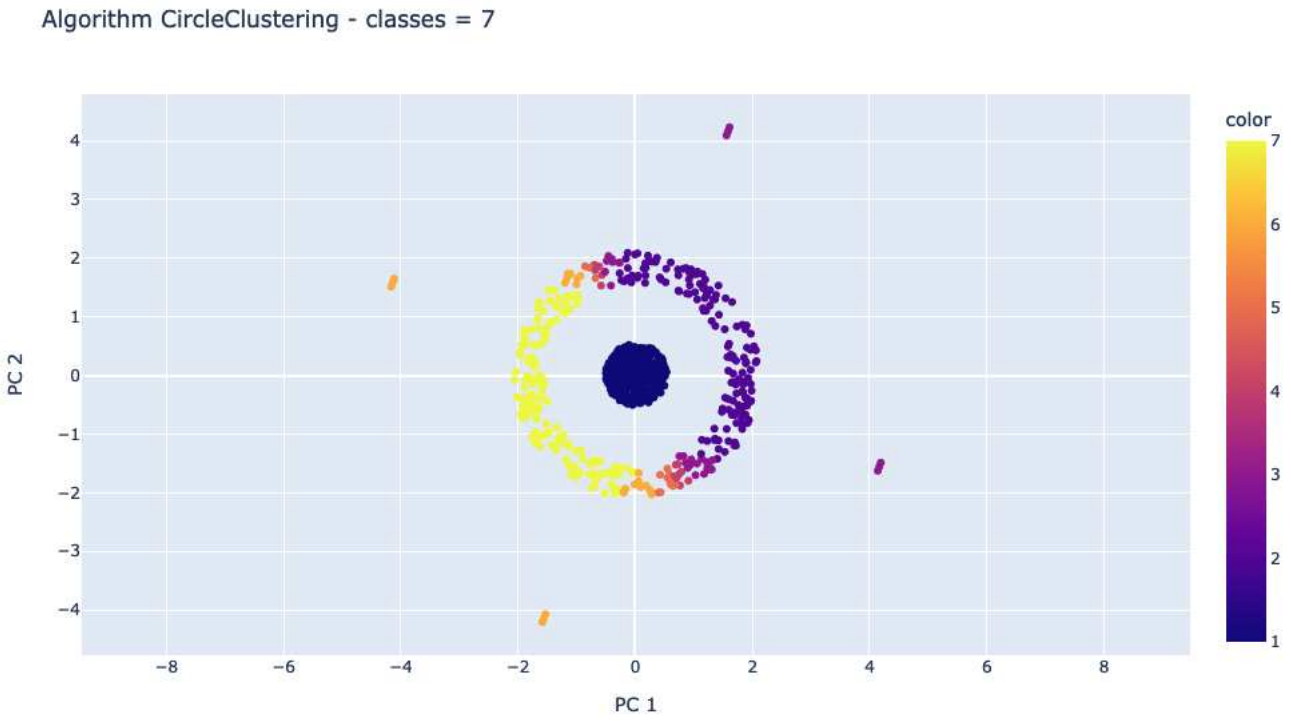
Tabella 3.9: twodiamonds

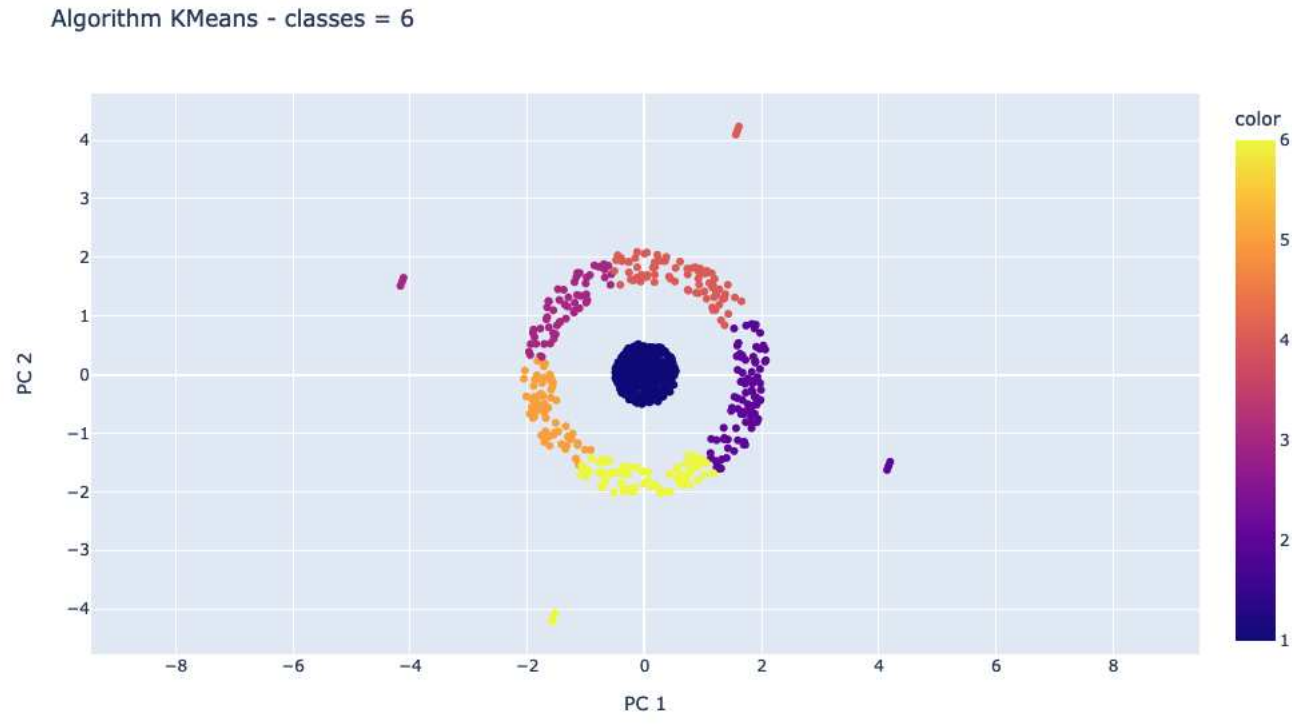
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 1.0 | 1.0 | 2.950 |
| KMeans | 1.0 | 1.0 | 0.160 |
| Spectral C. | 1.0 | 1.0 | 0.262 |
| Ward | 1.0 | 1.0 | 0.013 |
| DBscan | 0.0 | 0.0 | 0.006 |
| MeanShift | 1.0 | 1.0 | 2.506 |
| Optics | 0.0142 | 0.2 | 0.493 |
| Birch | 1.0 | 1.0 | 0.013 |
| Affinity Prop. | 0.1007 | 0.3684 | 1.275 |

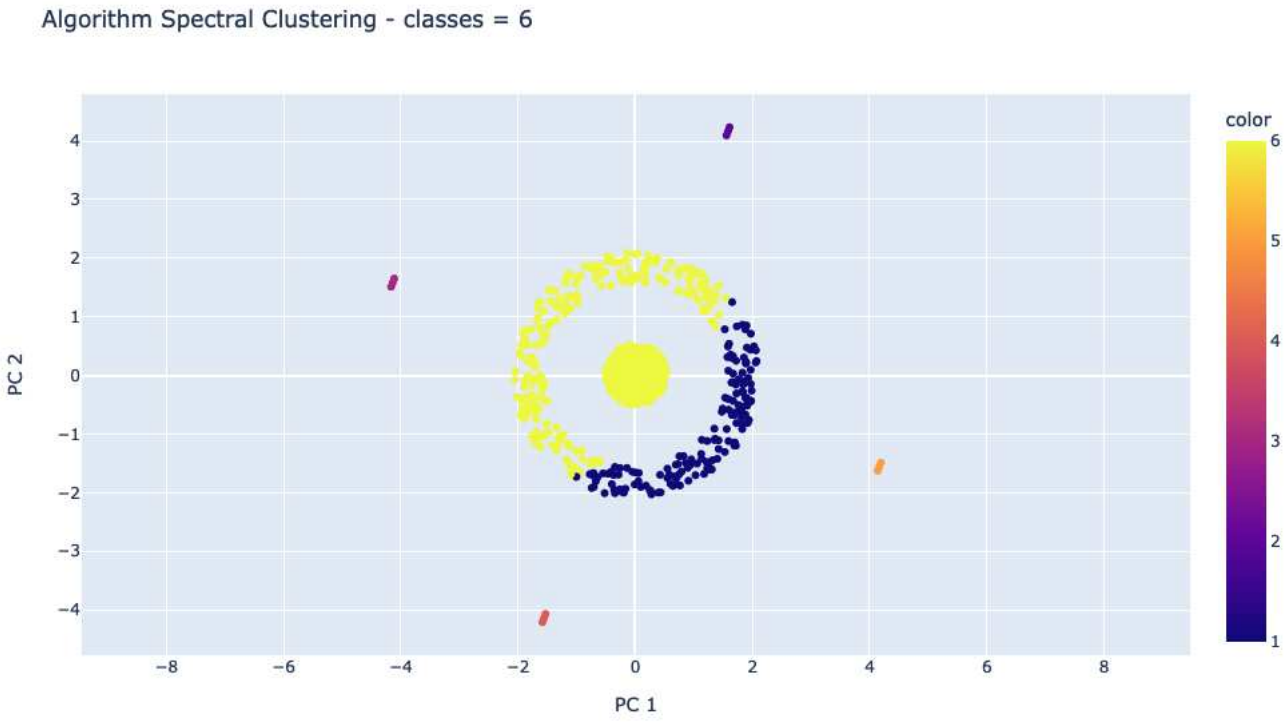
CircleClustering > KMeans, Spectral Clustering, Agglomerative Clustering (Ward), DBSCAN, Meanshift, OPTICS, Birch, Affinity Propagation

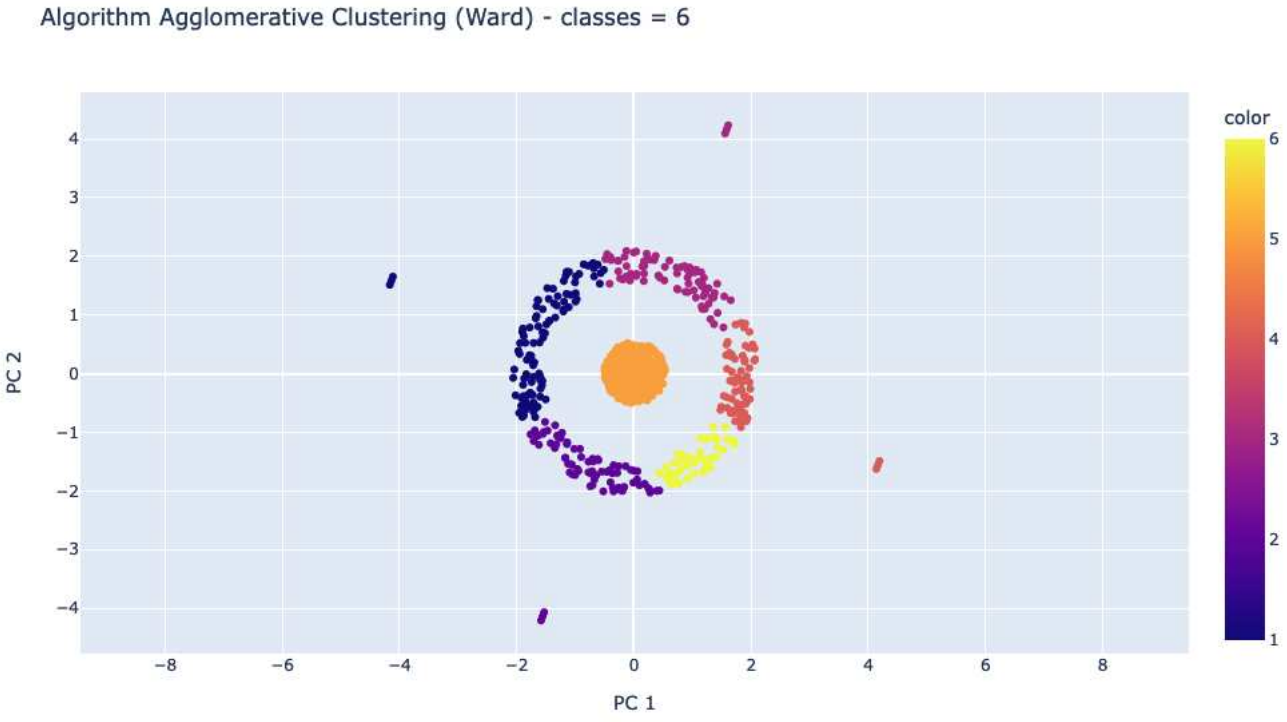
- Dataset: target
  Samples = 770, Features = 2, Classes = 6

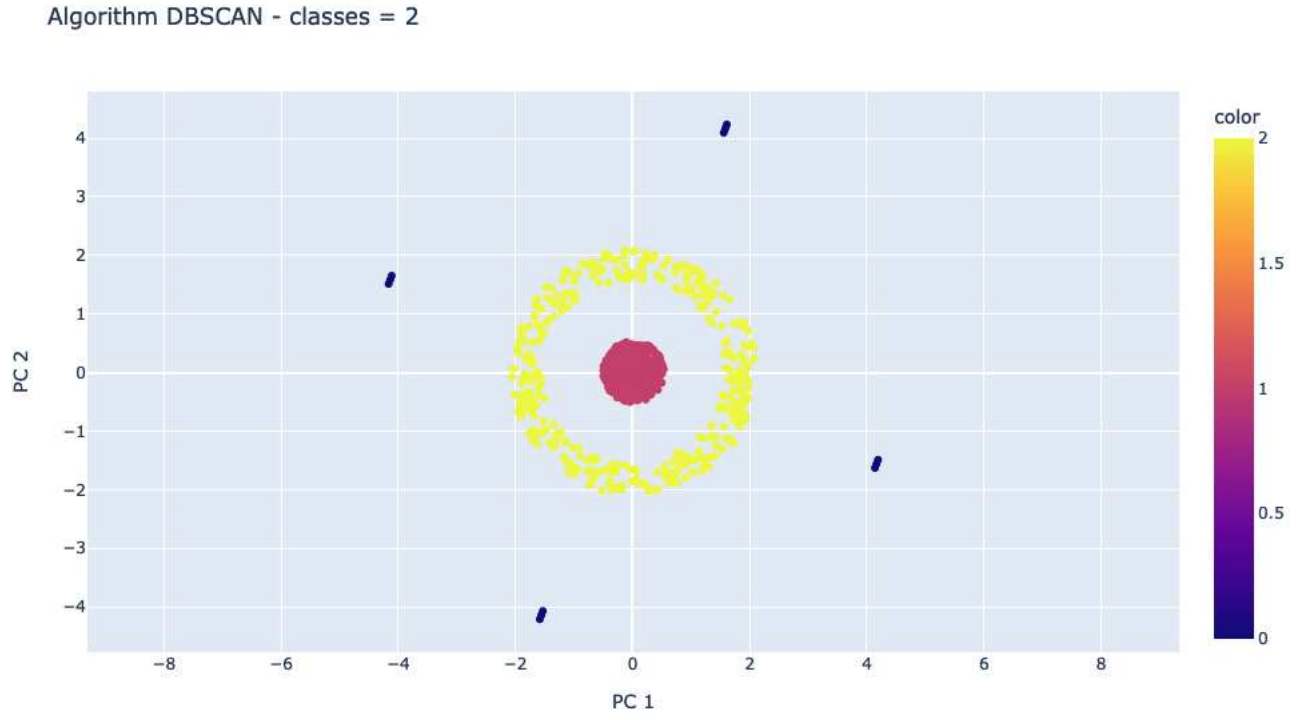


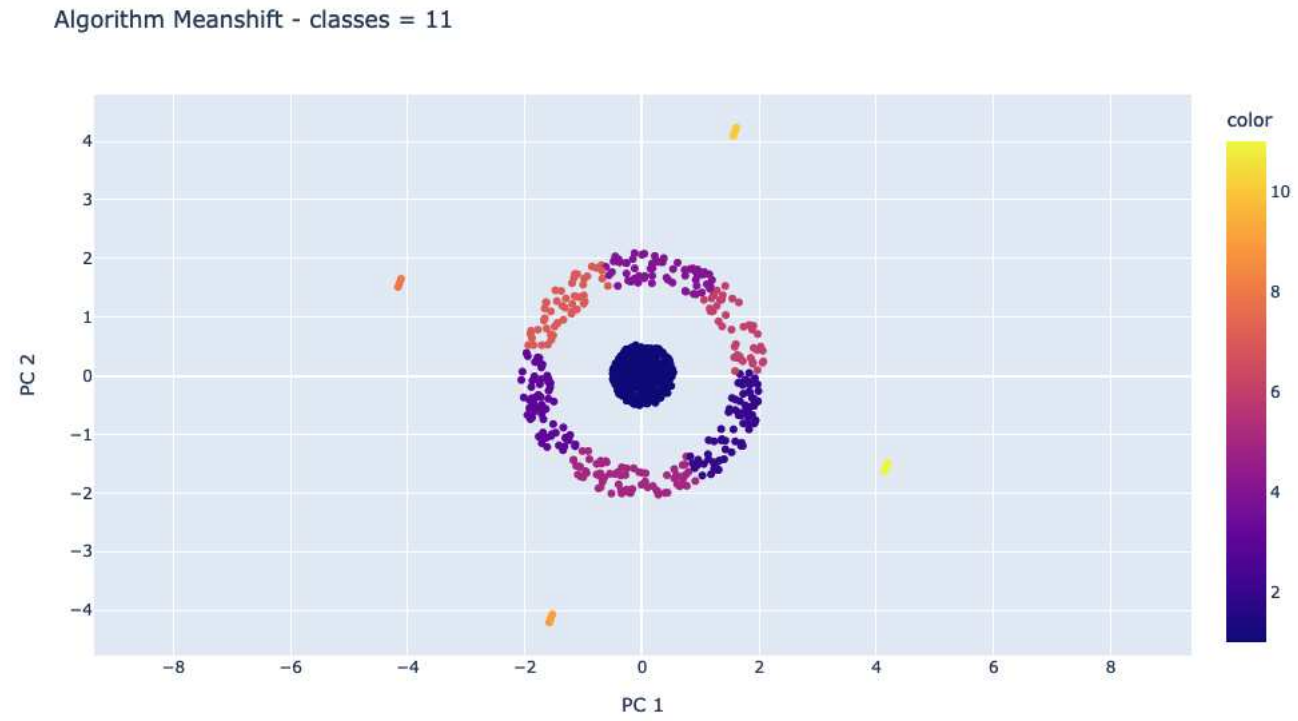

Algorithm CircleClustering - classes = 7
color
7
6
5
4
3
2
1
PC 2
PC 1
4
3
2
1
0
−1
−2
−3
−4
−8
−6
−4
−2
0
2
4
6
8

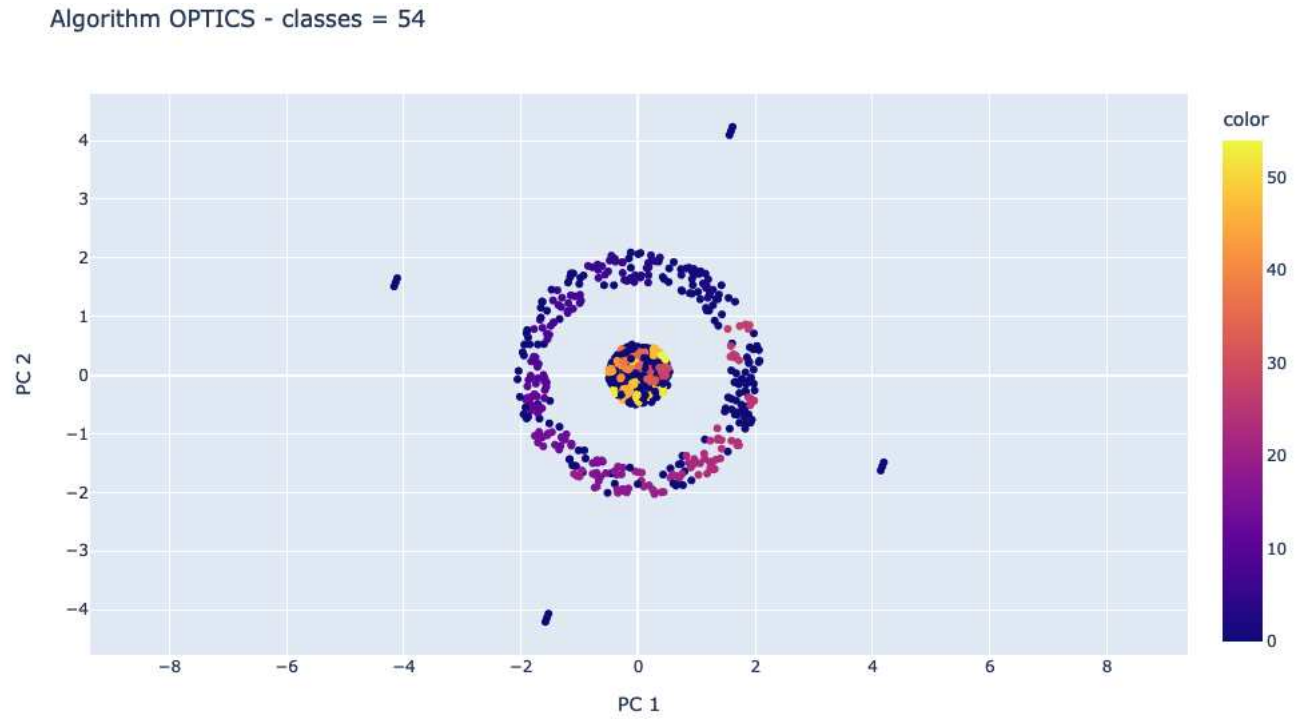

Algorithm KMeans - classes = 6
color
6
5
4
3
2
1
PC 2
PC 1
4
3
2
1
0
−1
−2
−3
−4
−8
−6
−4
−2
0
2
4
6
8


Algorithm Spectral Clustering - classes = 6
color
6
5
4
3
2
1
PC 2
PC 1
4
3
2
1
0
−1
−2
−3
−4
−8
−6
−4
−2
0
2
4
6
8


Algorithm Agglomerative Clustering (Ward) - classes = 6
color
6
5
4
3
2
1
PC 2
PC 1
4
3
2
1
0
−1
−2
−3
−4
−8
−6
−4
−2
0
2
4
6
8


Algorithm DBSCAN - classes = 2
color
2
1.5
1
0.5
0
PC 2
PC 1
4
3
2
1
0
−1
−2
−3
−4
−8
−6
−4
−2
0
2
4
6
8


Algorithm Meanshift - classes = 11
color
10
8
6
4
2
PC 2
PC 1
4
3
2
1
0
−1
−2
−3
−4
−8
−6
−4
−2
0
2
4
6
8


Algorithm OPTICS - classes = 54
color
50
40
30
20
10
0
PC 2
PC 1
4
3
2
1
0
−1
−2
−3
−4
−8
−6
−4
−2
0
2
4
6
8

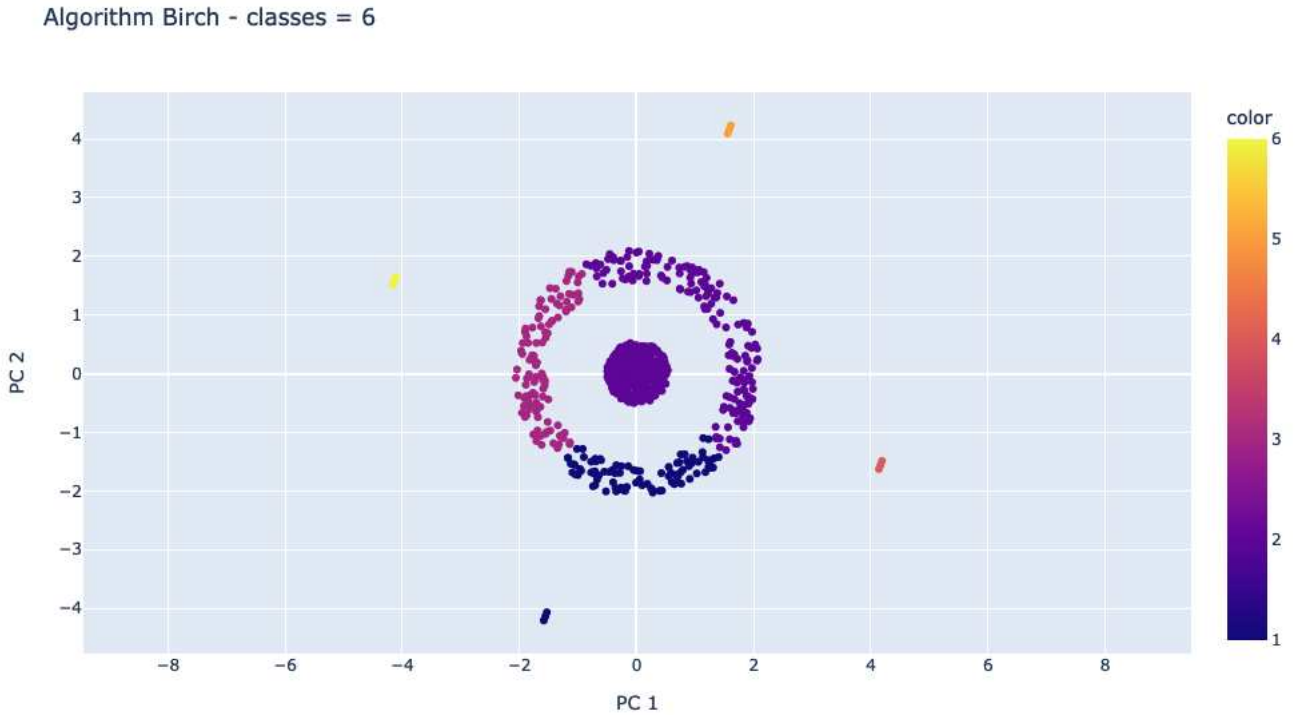
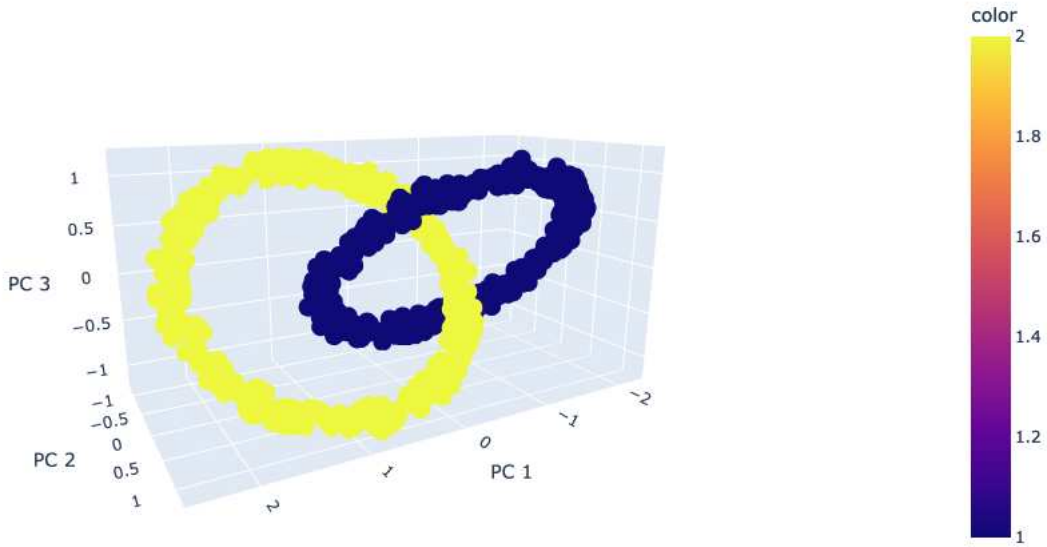

Algorithm Birch - classes = 6
color
6
5
4
3
2
1
PC 2
PC 1
4
3
2
1
0
−1
−2
−3
−4
−8
−6
−4
−2
0
2
4
6
8

Tabella 3.10: target

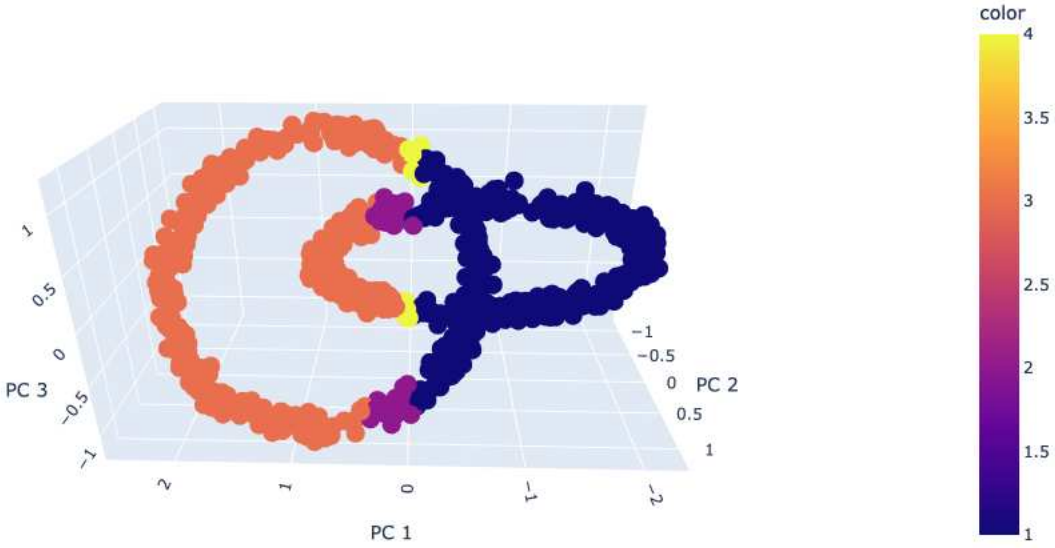
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.701 | 0.686 | 0.465 |
| KMeans | 0.6356 | 0.6316 | 0.029 |
| Spectral C. | 0.221 | 0.3868 | 0.287 |
| Ward | 0.638 | 0.6337 | 0.009 |
| DBscan | 0.999 | 0.985 | 0.005 |
| MeanShift | 0.627 | 0.645 | 1.776 |
| Optics | -0.001 | 0.1912 | 0.463 |
| Birch | 0.283 | 0.402 | 0.013 |
| Affinity Prop. | 0.013 | 0.241 | 2.046 |

CircleClustering > KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Meanshift, OPTICS, Birch, Affinity Propagation
CircleClustering < DBSCAN

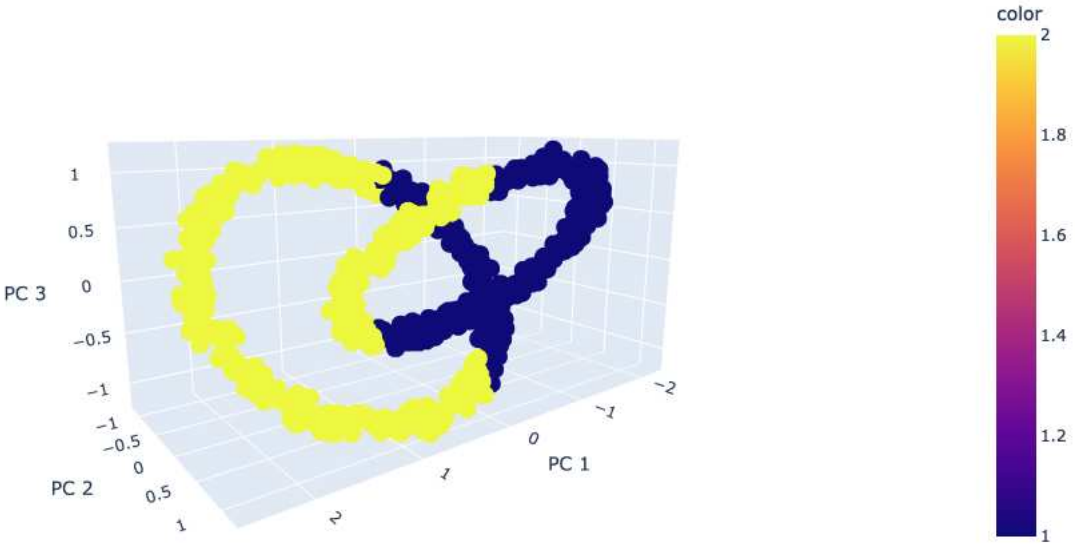
- Dataset: chainlink
  Samples = 1000, Features = 3, Classes = 2



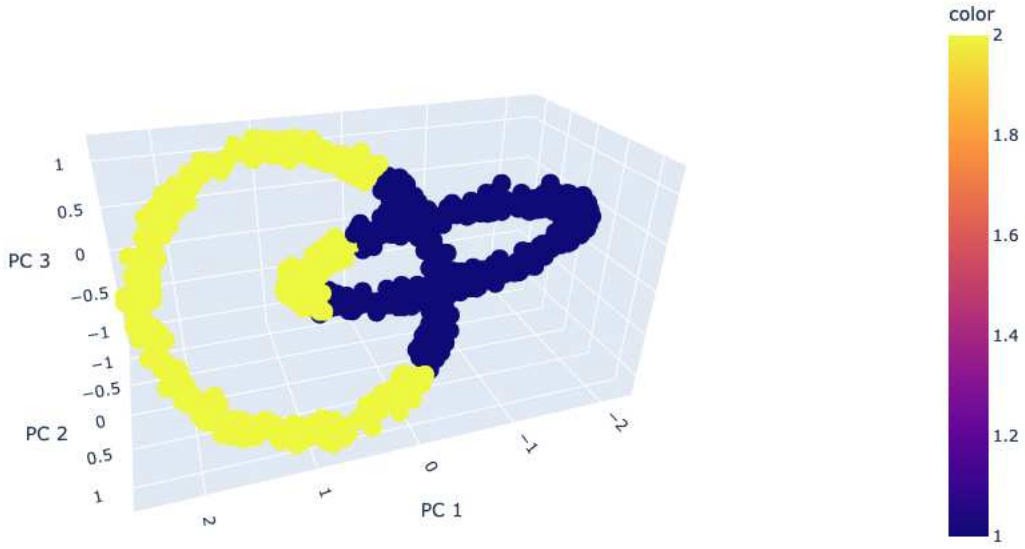
Algorithm CircleClustering - classes = 4



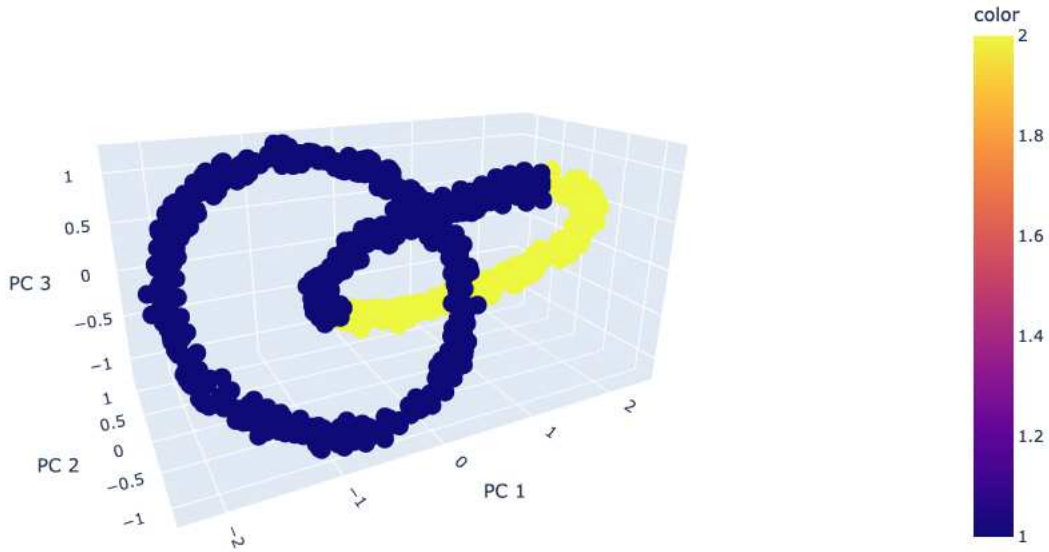

Algorithm KMeans - classes = 2



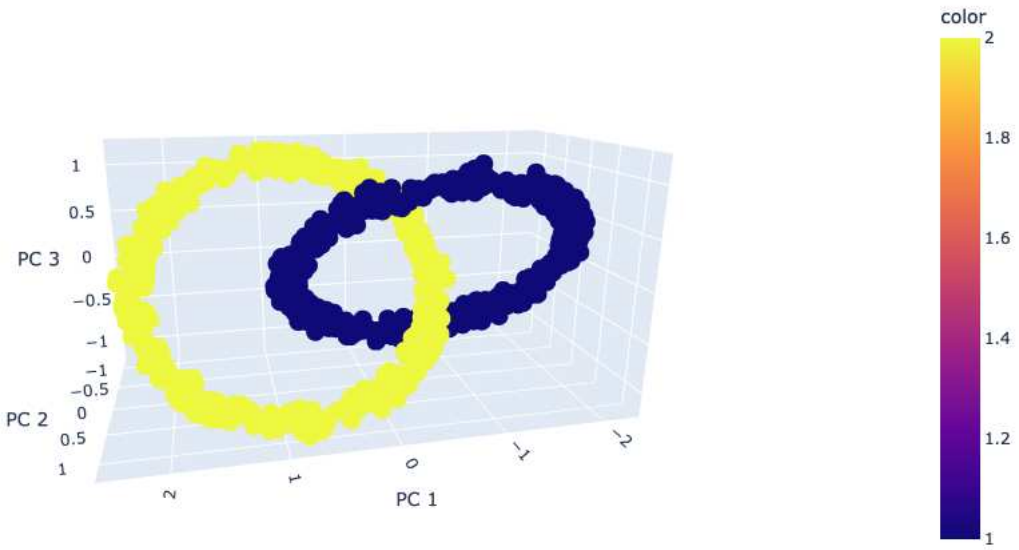
Algorithm Spectral Clustering - classes = 2



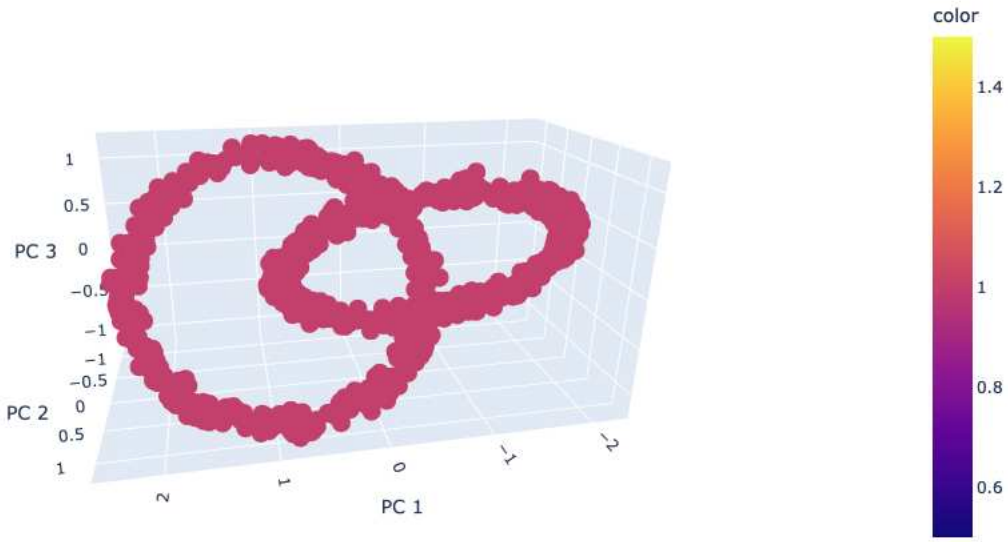

Algorithm Agglomerative Clustering (Ward) - classes = 2



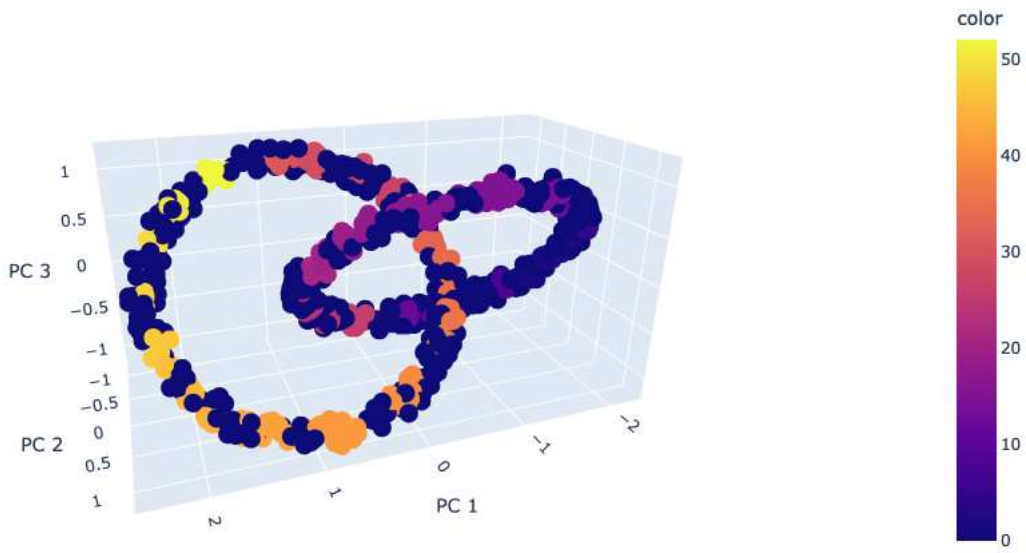
Algorithm DBSCAN - classes = 2



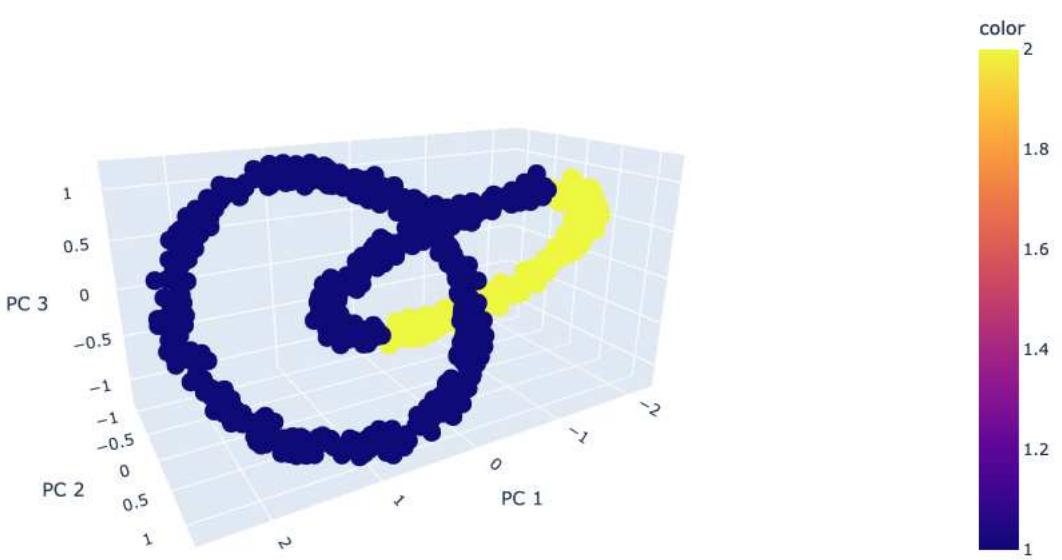

Algorithm Meanshift - classes = 1



Algorithm OPTICS - classes = 52




Algorithm Birch - classes = 2



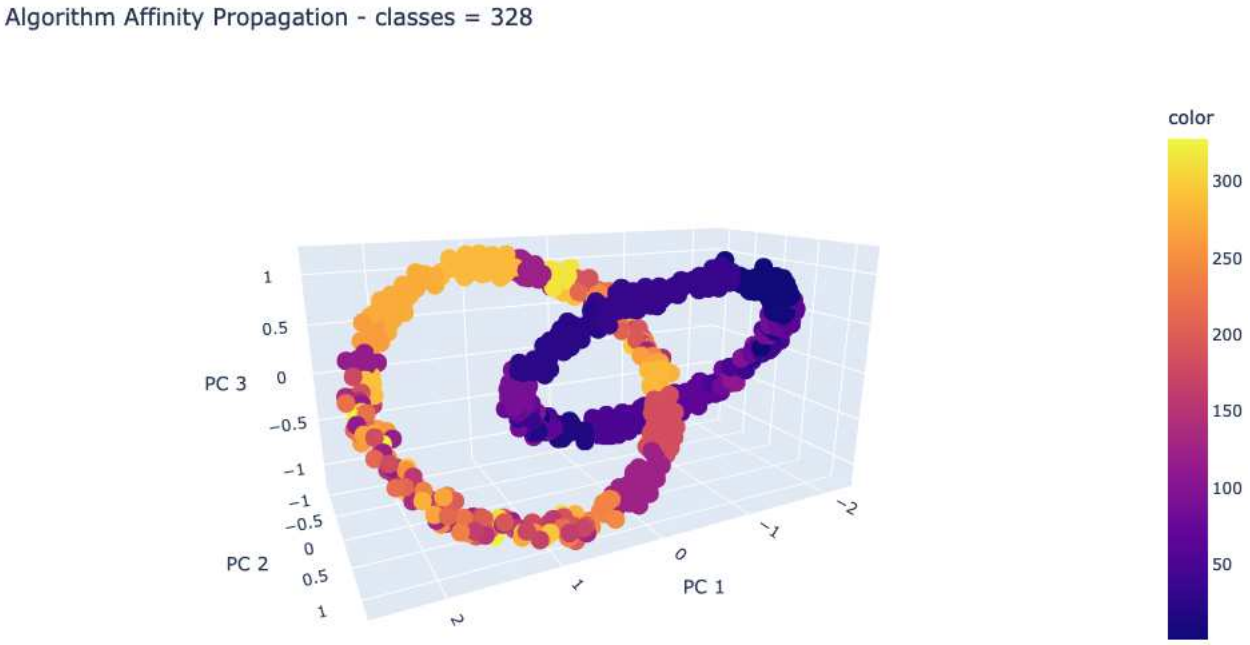
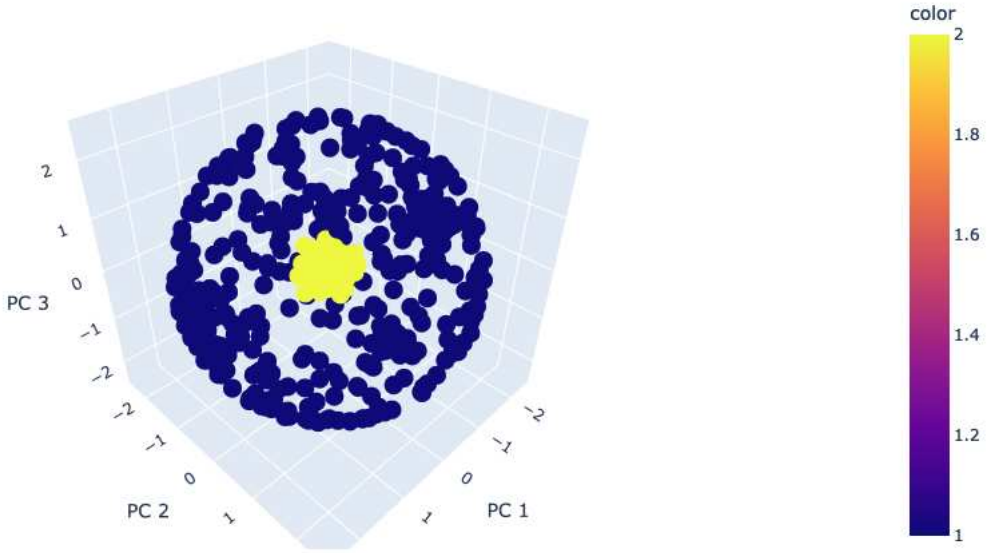



Tabella 3.11: chainlink

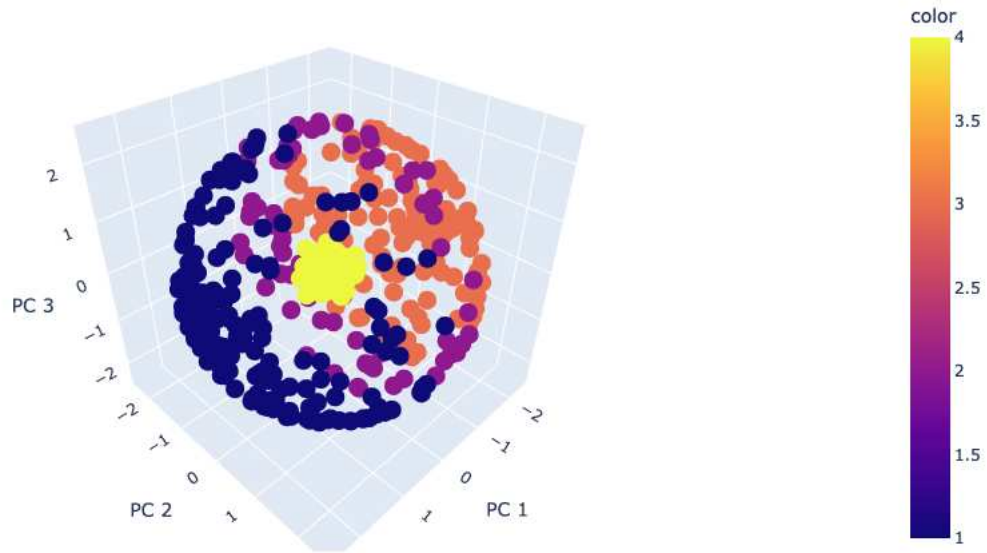
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.097 | 0.0686 | 0.375 |
| KMeans | 0.094 | 0.0689 | 0.019 |
| Spectral C. | 0.251 | 0.191 | 0.317 |
| Ward | 0.280 | 0.365 | 0.018 |
| DBscan | 1.0 | 1.0 | 0.007 |
| MeanShift | 0.0 | 0.0 | 6.167 |
| Optics | 0.0099 | 0.195 | 0.614 |
| Birch | 0.317 | 0.3914 | 0.017 |
| Affinity Prop. | 0.053 | 0.1989 | 3.524 |

CircleClustering > KMeans, Meanshift, OPTICS, Affinity Propagation
CircleClustering < Spectral Clustering, Agglomerative Clustering (Ward), DBSCAN, Birch

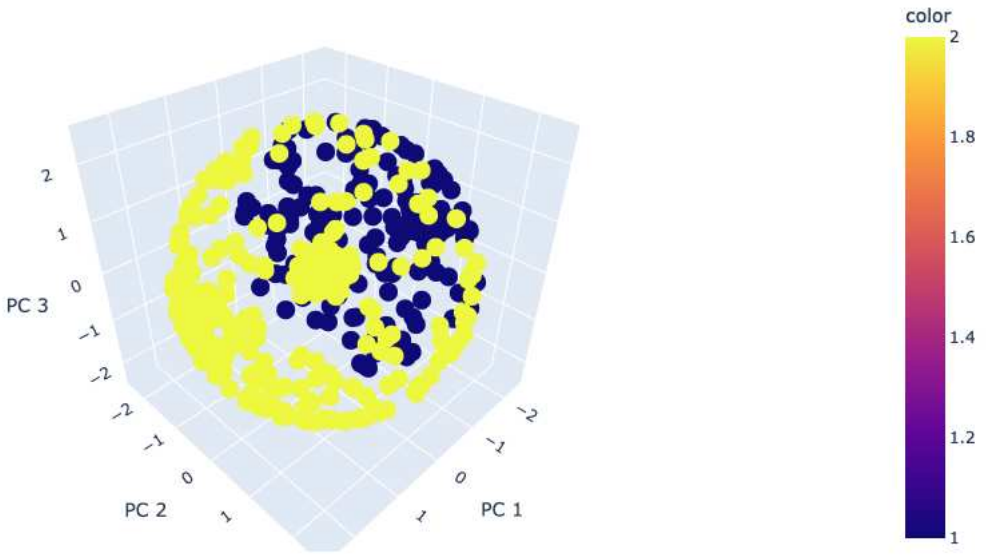
- Dataset: atom
  Samples = 800, Features = 3, Classes = 2



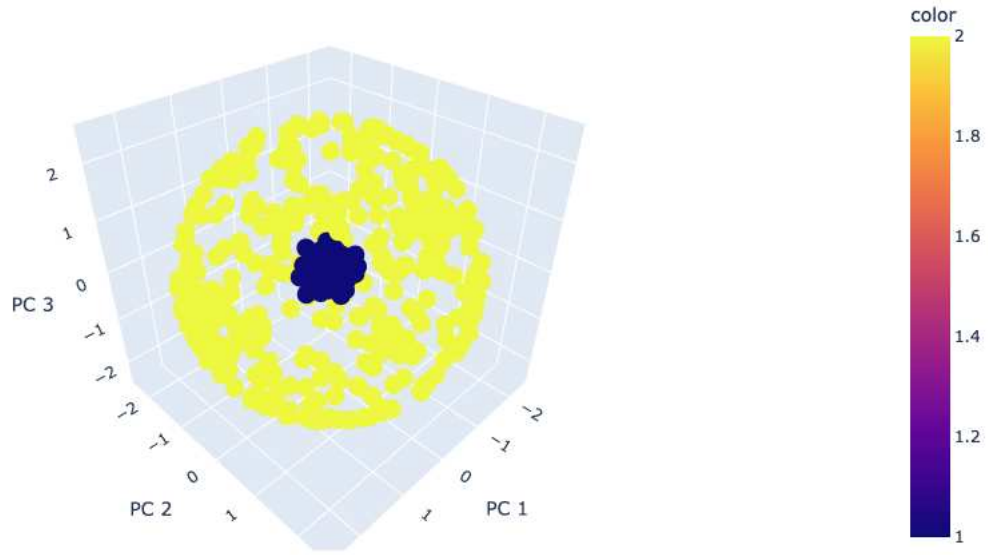

Algorithm CircleClustering - classes = 4



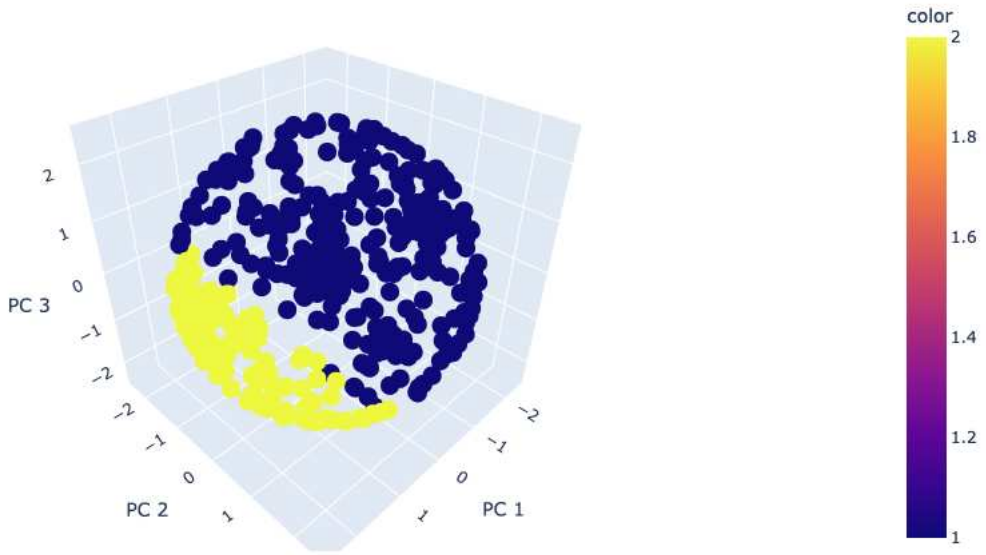
Algorithm KMeans - classes = 2



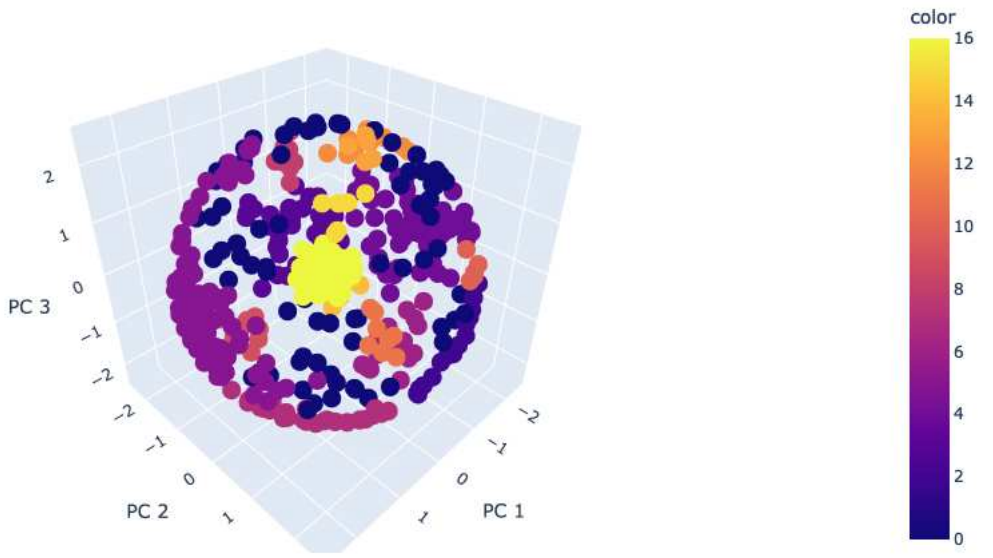

Algorithm Spectral Clustering - classes = 2



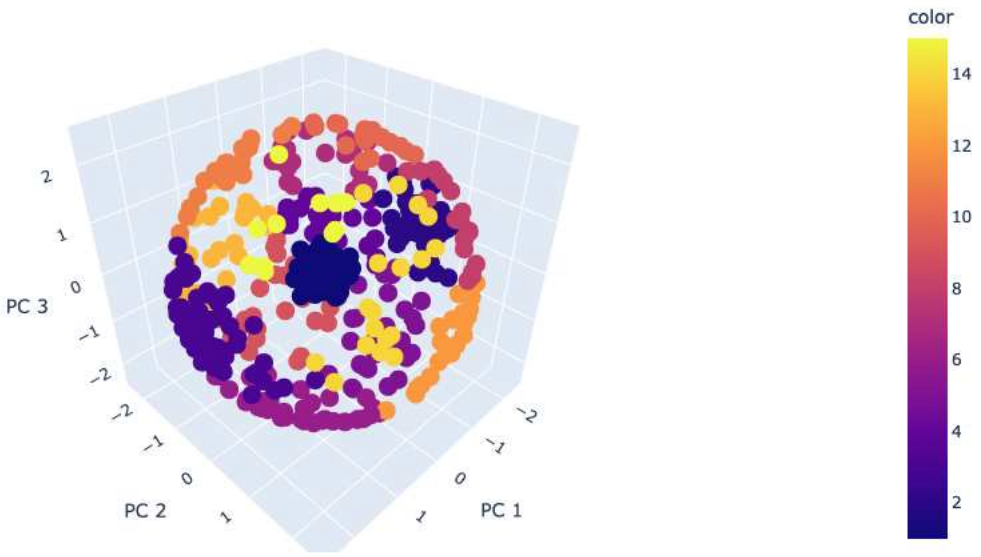
Algorithm Agglomerative Clustering (Ward) - classes = 2



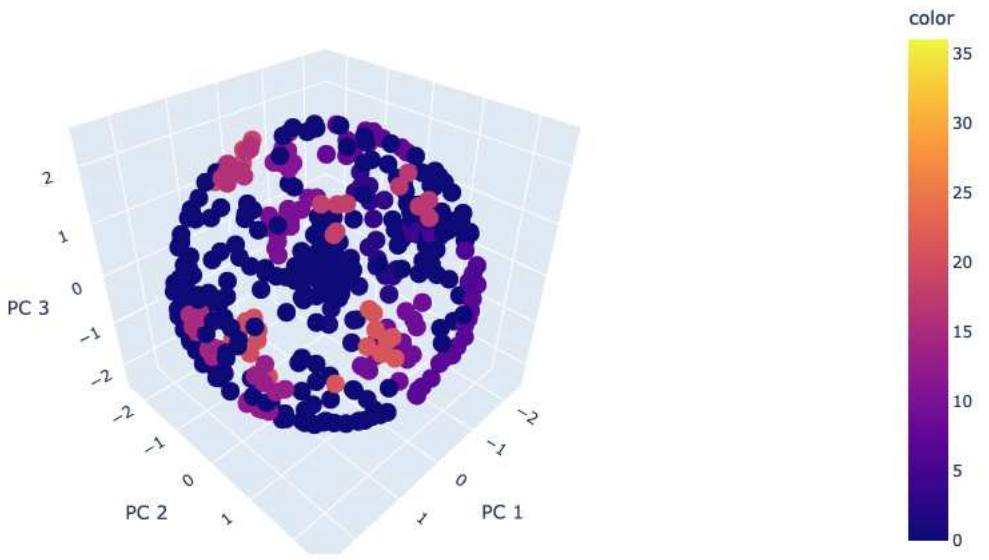

Algorithm DBSCAN - classes = 16



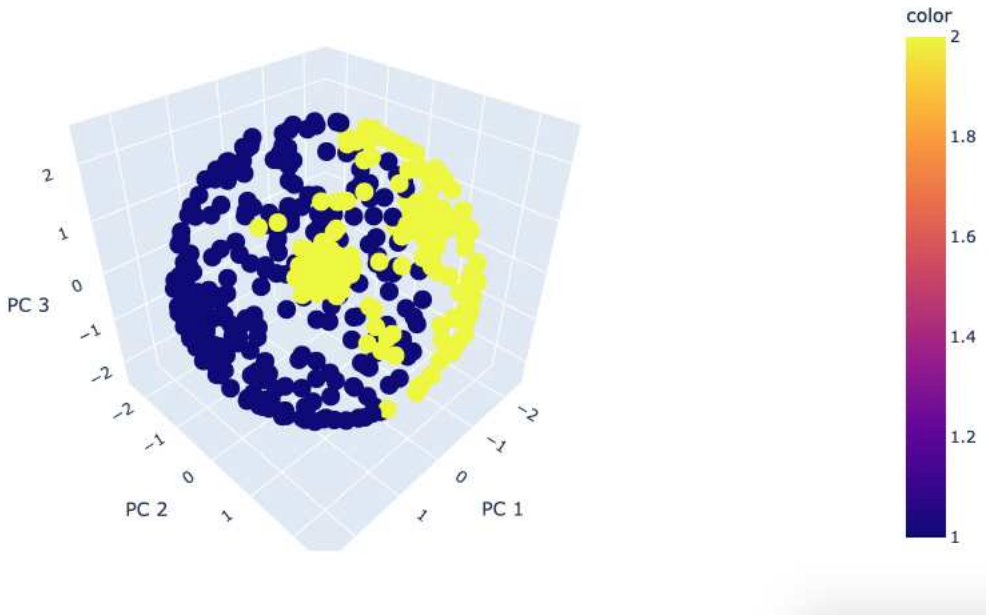
Algorithm Meanshift - classes = 15




Algorithm OPTICS - classes = 36



Algorithm Birch - classes = 2




Algorithm Affinity Propagation - classes = 30

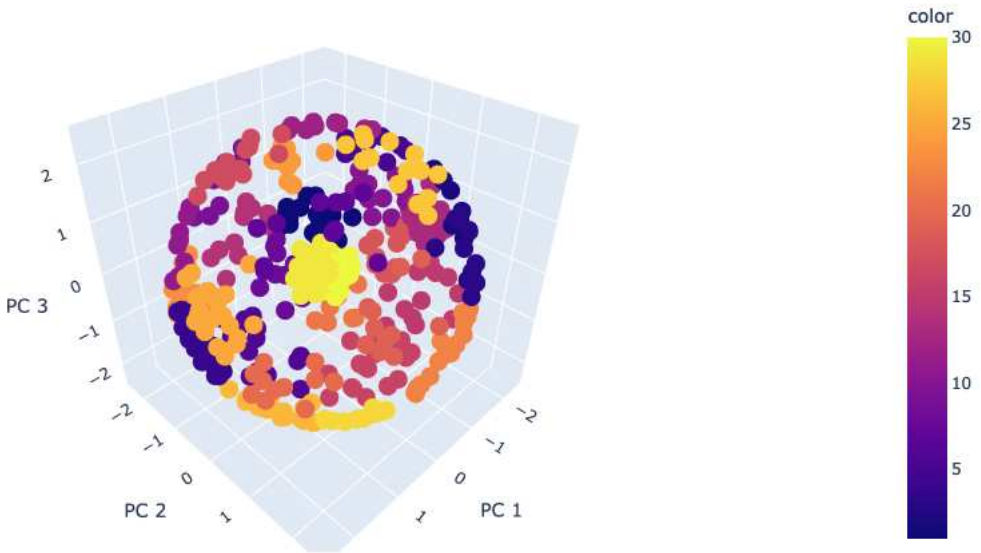
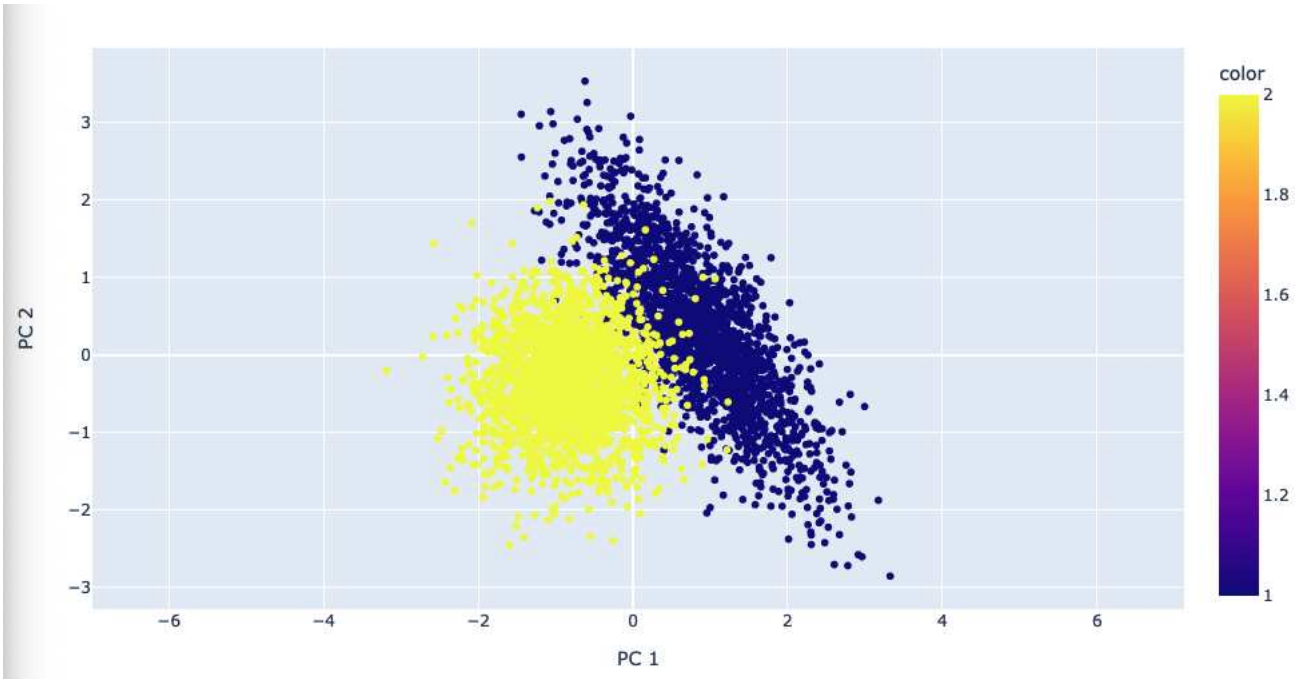


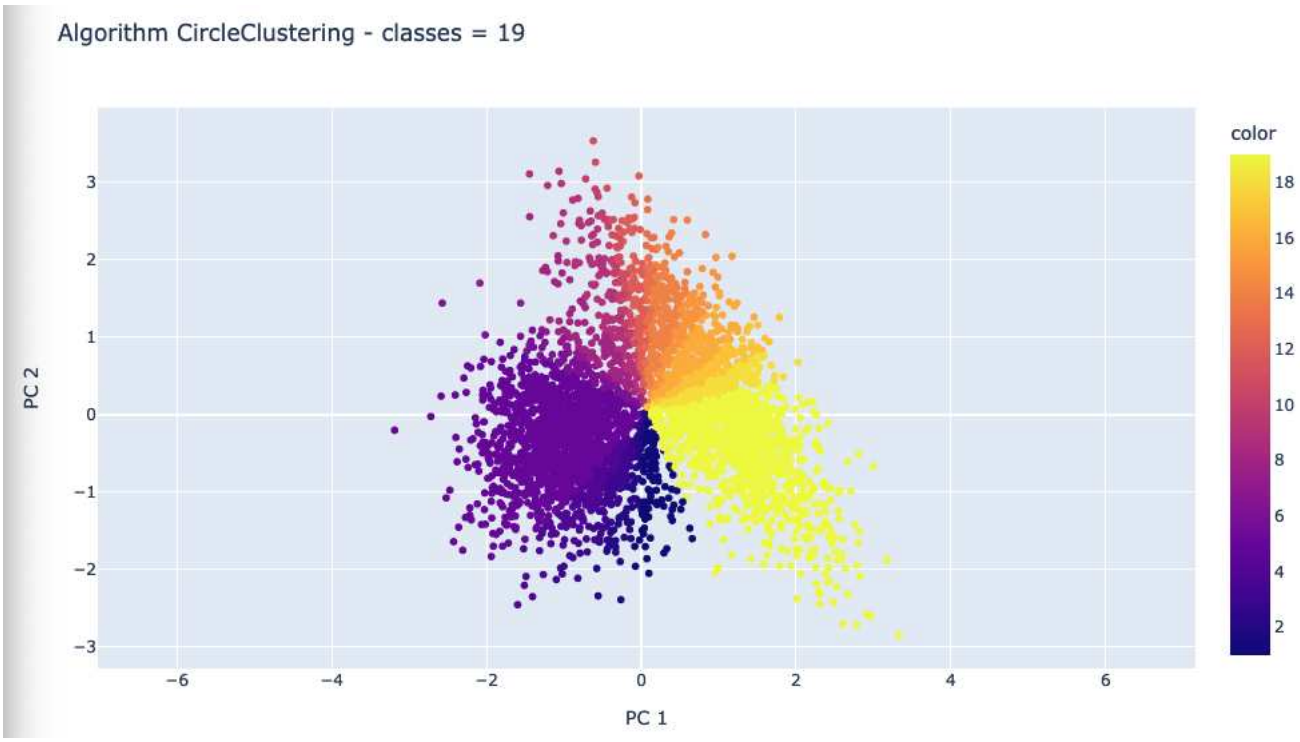
Tabella 3.12: atom

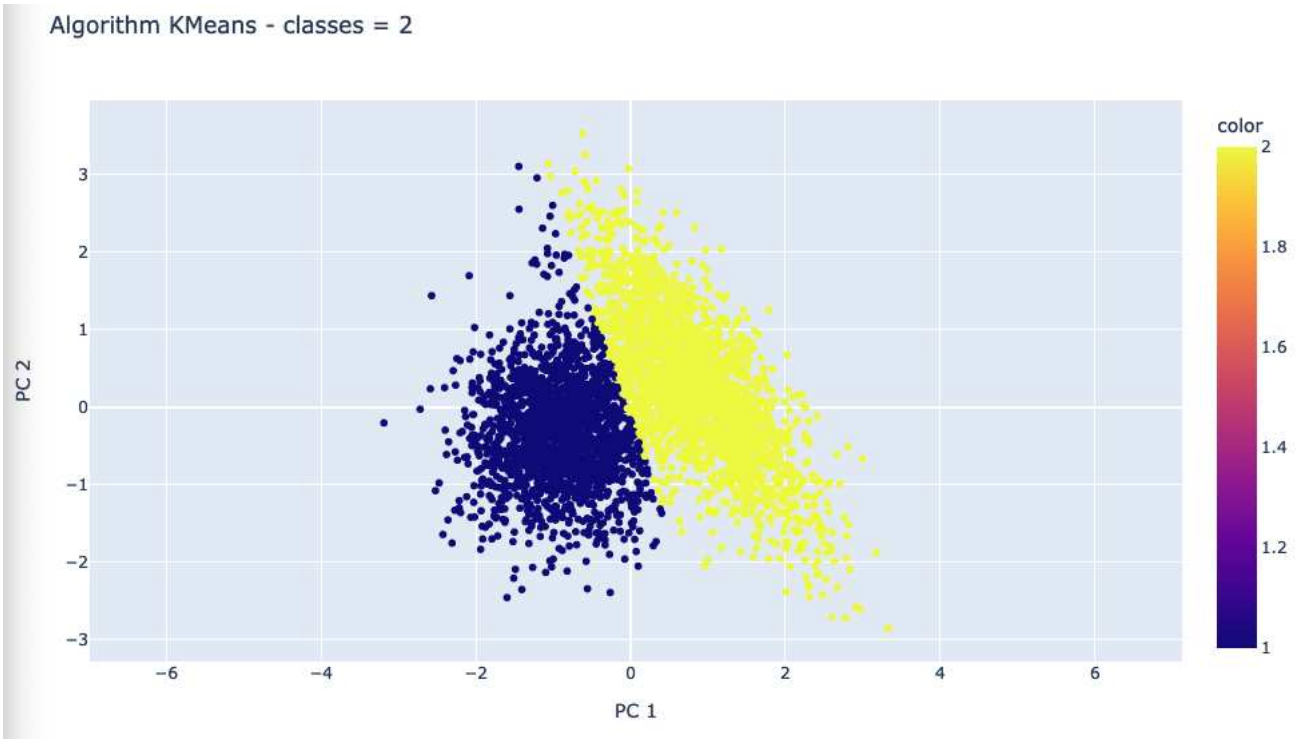
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.6892 | 0.7301 | 0.281 |
| KMeans | 0.182 | 0.292 | 0.018 |
| Spectral C. | 1.0 | 1.0 | 0.280 |
| Ward | 0.098 | 0.2188 | 0.011 |
| DBscan | 0.570 | 0.547 | 0.007 |
| MeanShift | 0.543 | 0.519 | 1.594 |
| Optics | 0.027 | 0.182 | 0.481 |
| Birch | 0.461 | 0.490 | 0.0231 |
| Affinity Prop. | 0.278 | 0.4066 | 1.737 |

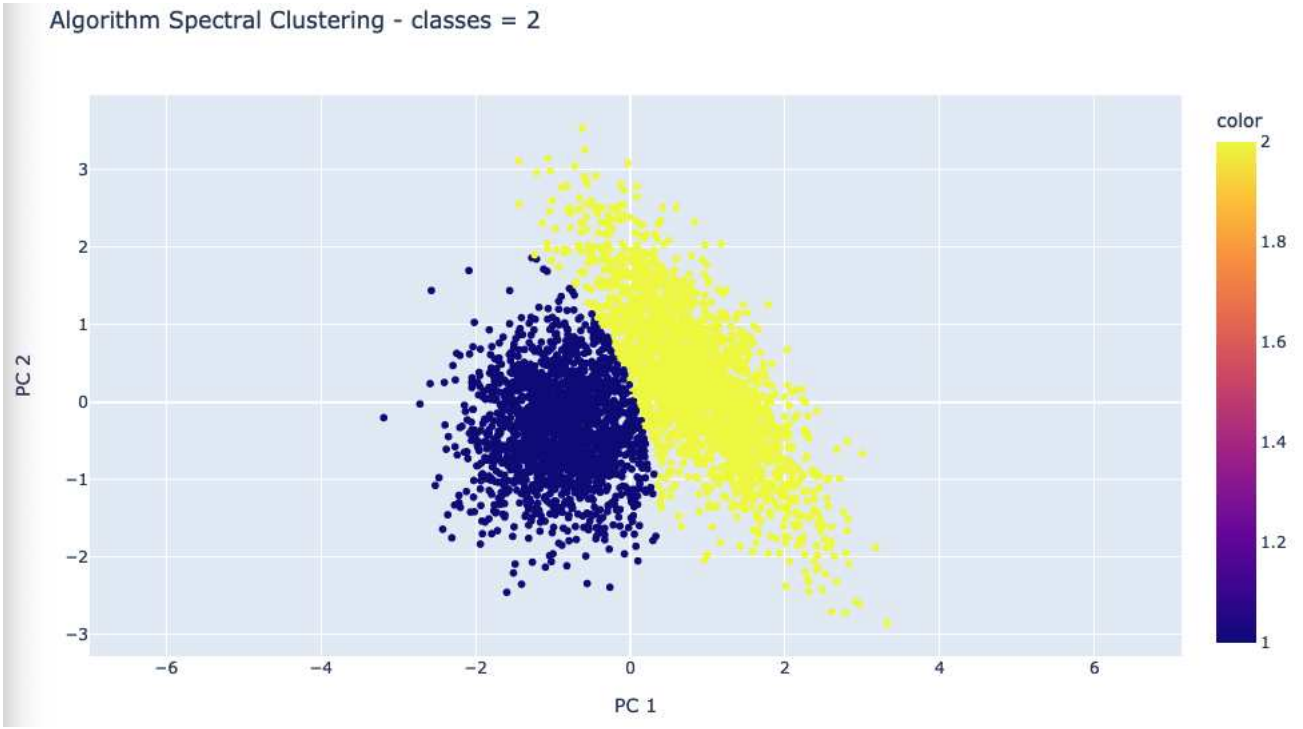
CircleClustering > KMeans, Agglomerative Clustering (Ward), DBSCAN, Meanshift, OPTICS, Birch, Affinity Propagation
CircleClustering < Spectral Clustering

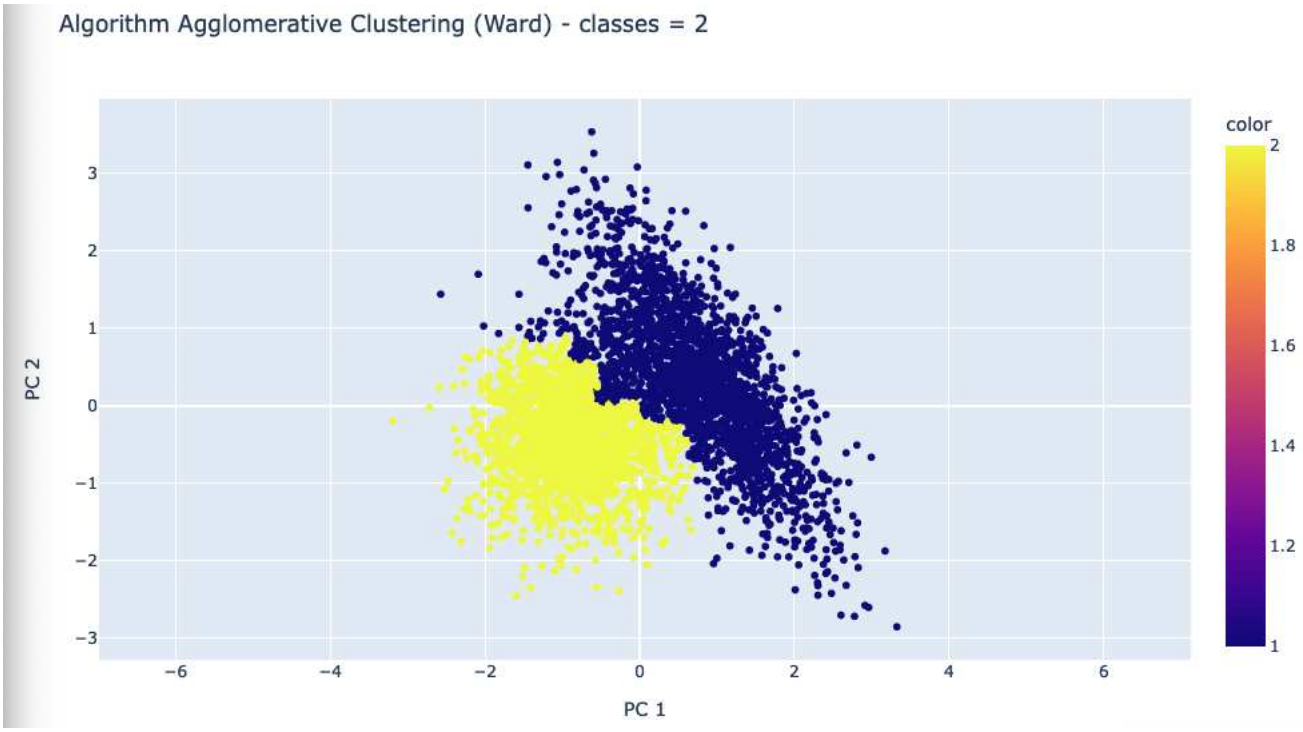
- Dataset: engytime
  Samples = 4096, Features = 2, Classes = 2

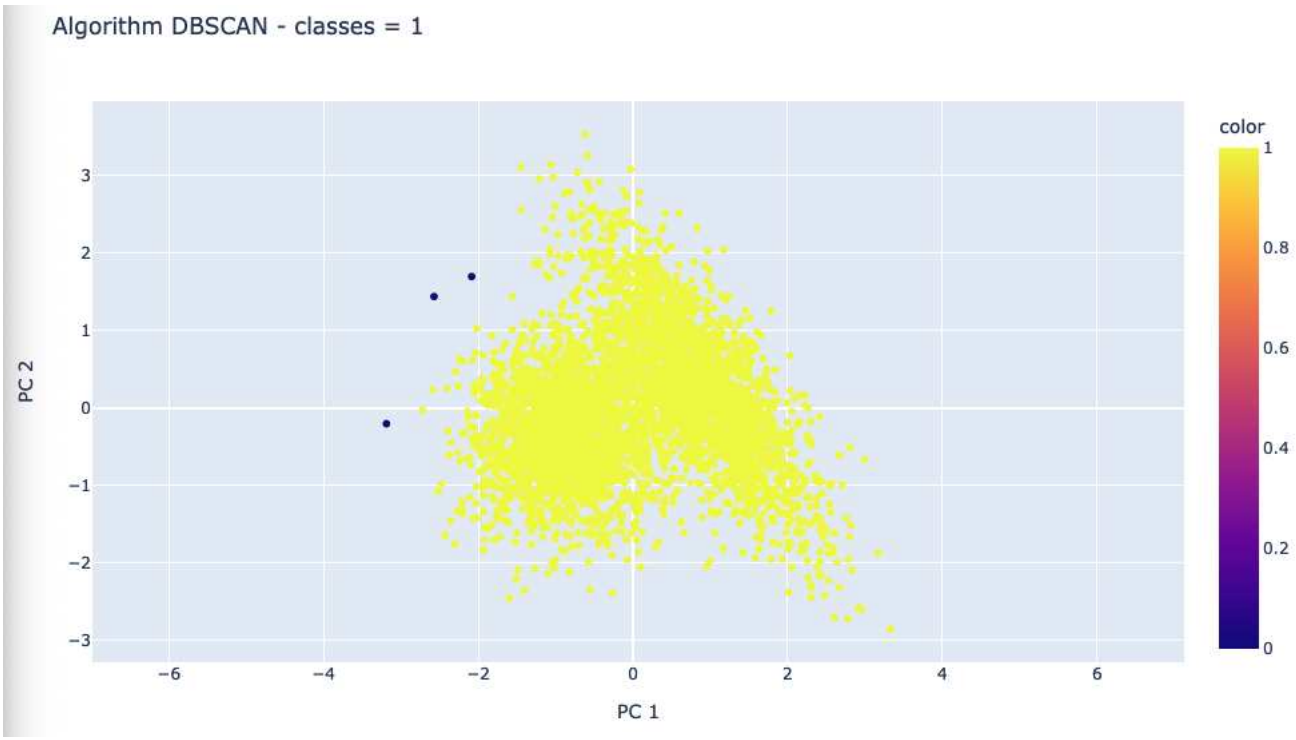


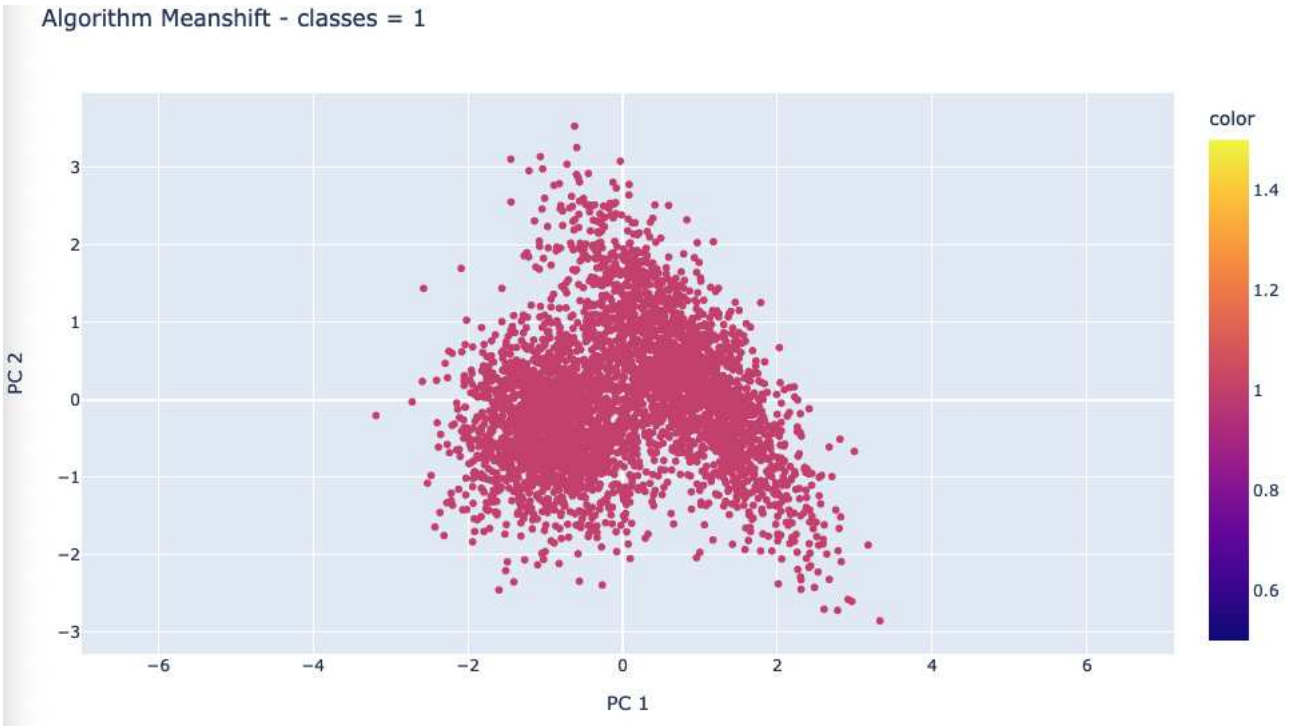

Algorithm CircleClustering - classes = 19
color
18
16
14
12
10
8
6
4
2
PC 2
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6
PC 1

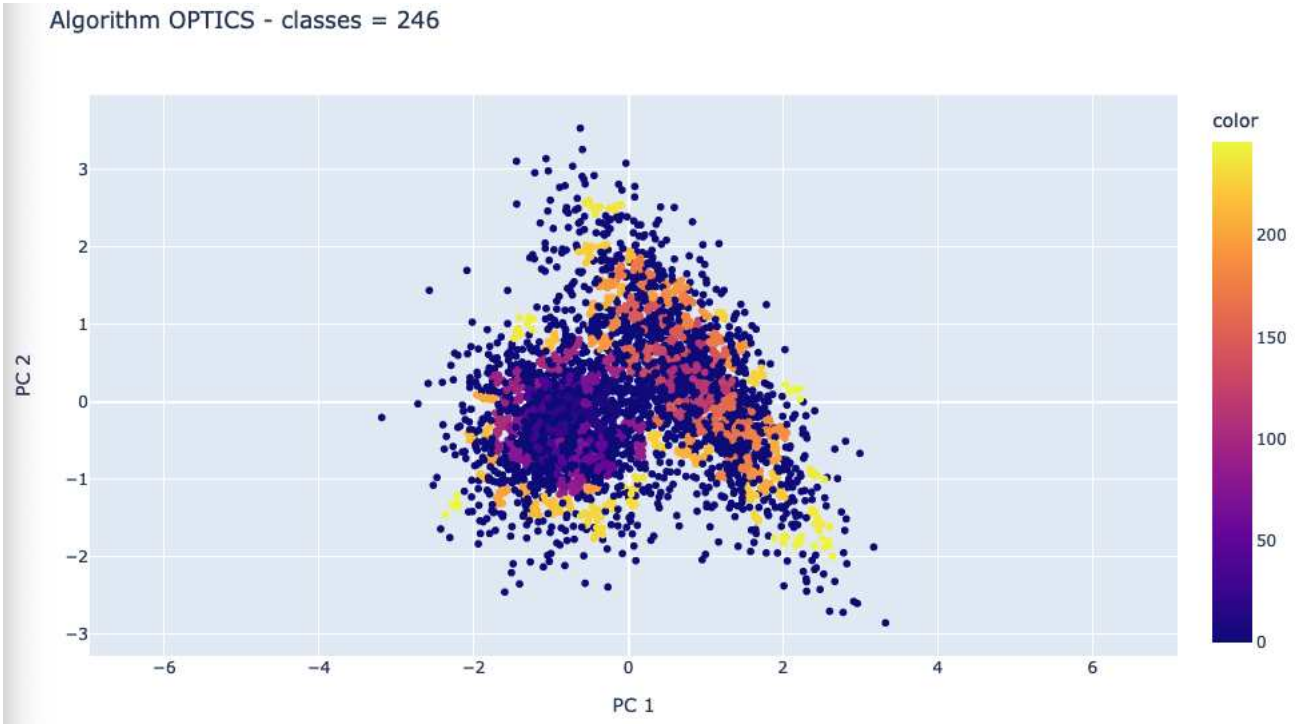

Algorithm KMeans - classes = 2
color
2
1.8
1.6
1.4
1.2
1
PC 2
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6
PC 1

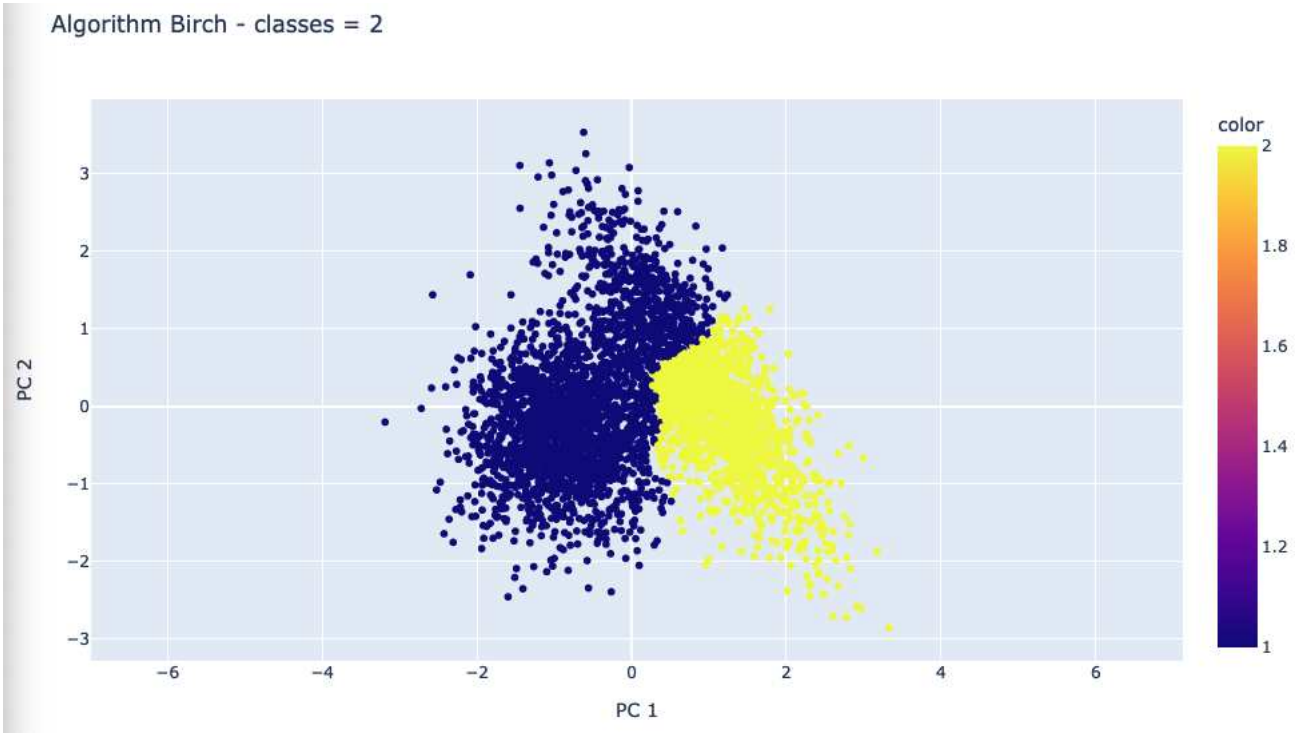

Algorithm Spectral Clustering - classes = 2
color
2
1.8
1.6
1.4
1.2
1
PC 2
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6
PC 1


Algorithm Agglomerative Clustering (Ward) - classes = 2
color
2
1.8
1.6
1.4
1.2
1
PC 2
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6
PC 1


Algorithm DBSCAN - classes = 1
color
1
0.8
0.6
0.4
0.2
0
PC 2
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6
PC 1


Algorithm Meanshift - classes = 1
color
1.4
1.2
1
0.8
0.6
PC 2
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6
PC 1


Algorithm OPTICS - classes = 246
color
200
150
100
50
0
PC 2
PC 1
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6


Algorithm Birch - classes = 2
color
2
1.8
1.6
1.4
1.2
1
PC 2
PC 1
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6

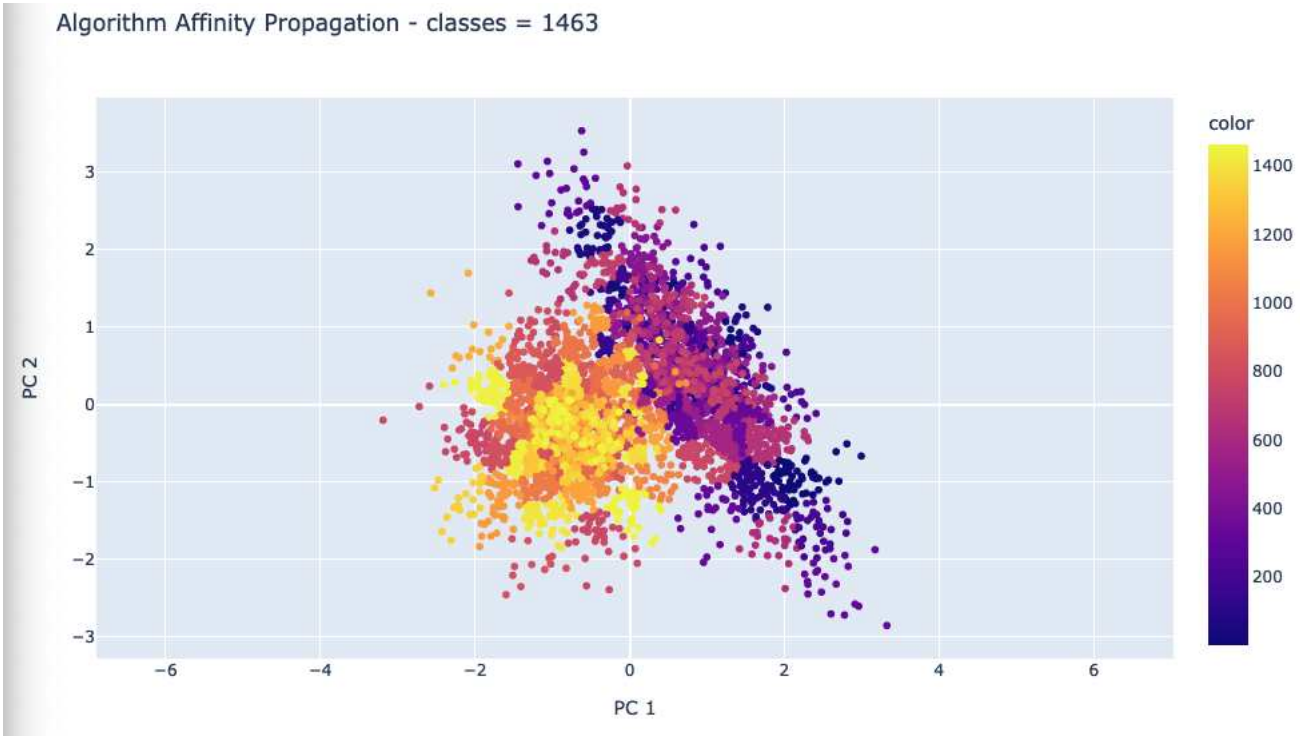
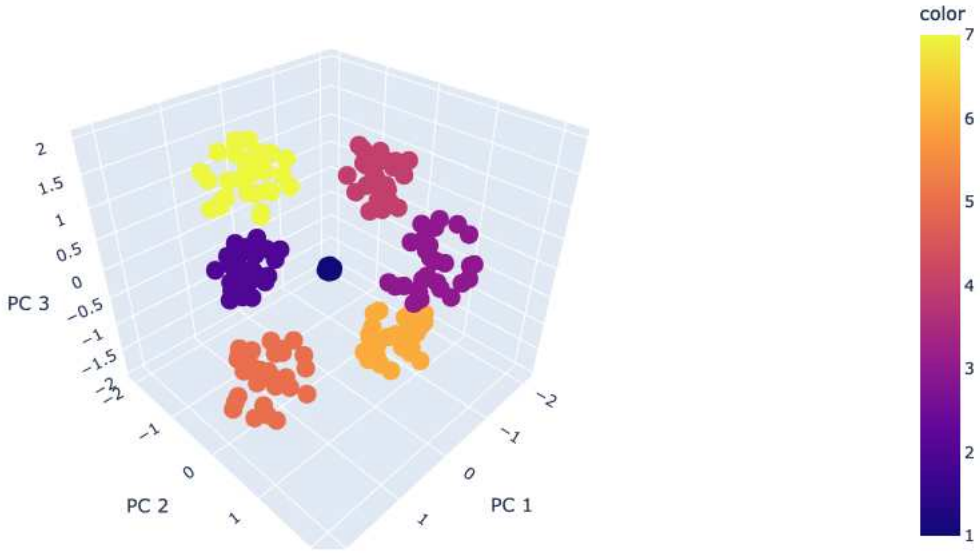



Tabella 3.13: engytime

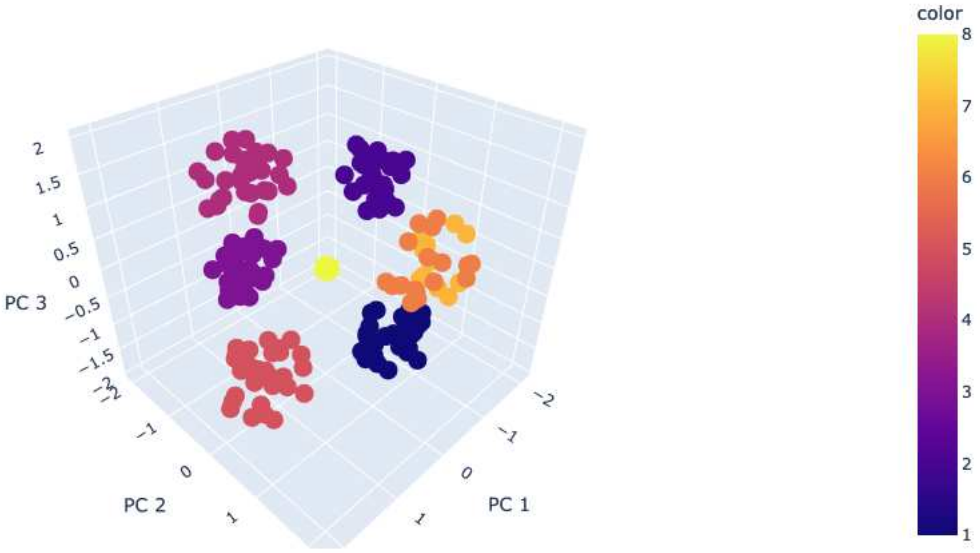
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.324 | 0.377 | 1.404 |
| KMeans | 0.815 | 0.729 | 0.020 |
| Spectral C. | 0.846 | 0.846 | 2.802 |
| Ward | 0.718 | 0.646 | 0.347 |
| DBscan | 0.0 | 0.001 | 0.058 |
| MeanShift | 0.0 | 0.0 | 132.97 |
| Optics | 0.0017 | 0.1353 | 2.960 |
| Birch | 0.376 | 0.380 | 0.068 |
| Affinity Prop. | 0.0091 | 0.1187 | 58.43 |

CircleClustering > DBSCAN, Meanshift, OPTICS, Affinity Propagation
CircleClustering < KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Birch

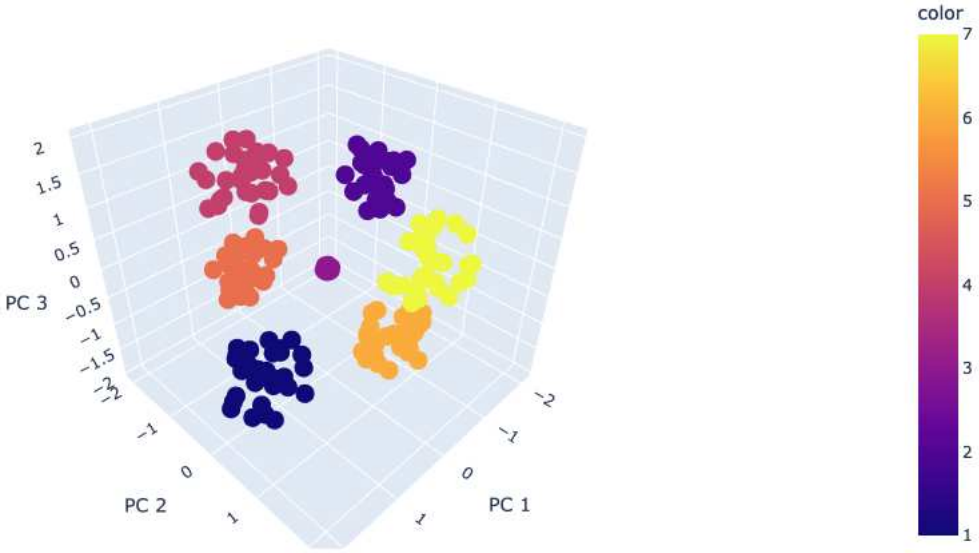
- Dataset: hepta
  Samples = 212, Features = 3, Classes = 7



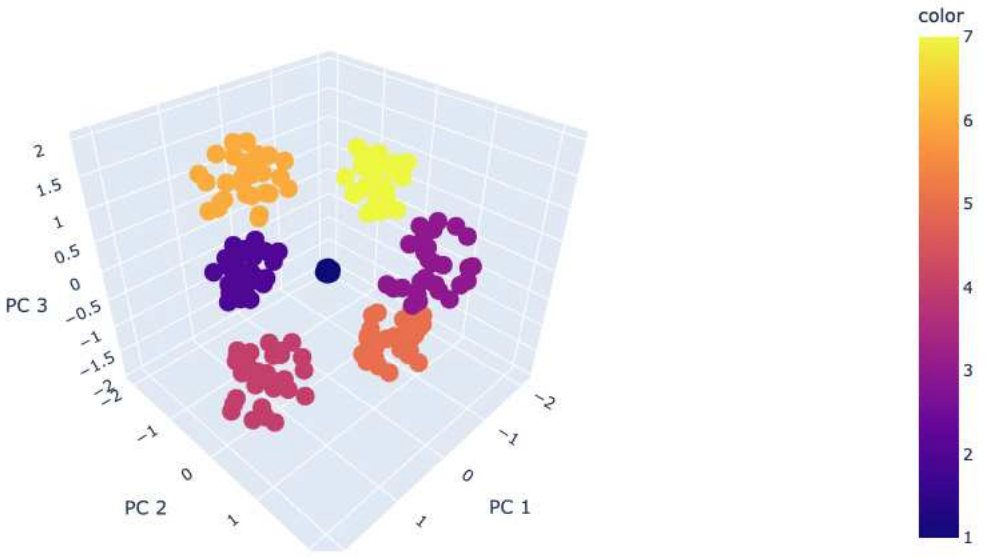

Algorithm CircleClustering - classes = 8



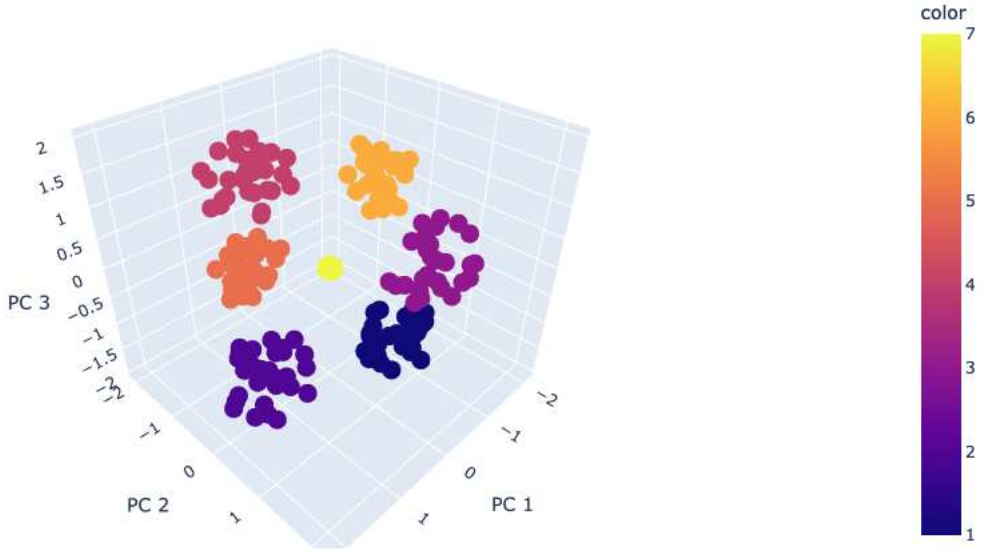
Algorithm KMeans - classes = 7



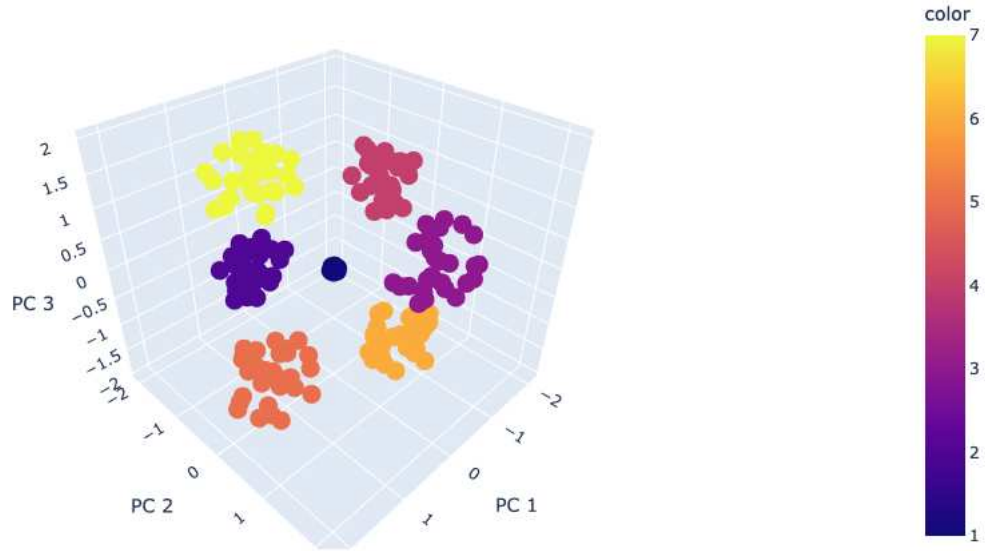

Algorithm Spectral Clustering - classes = 7



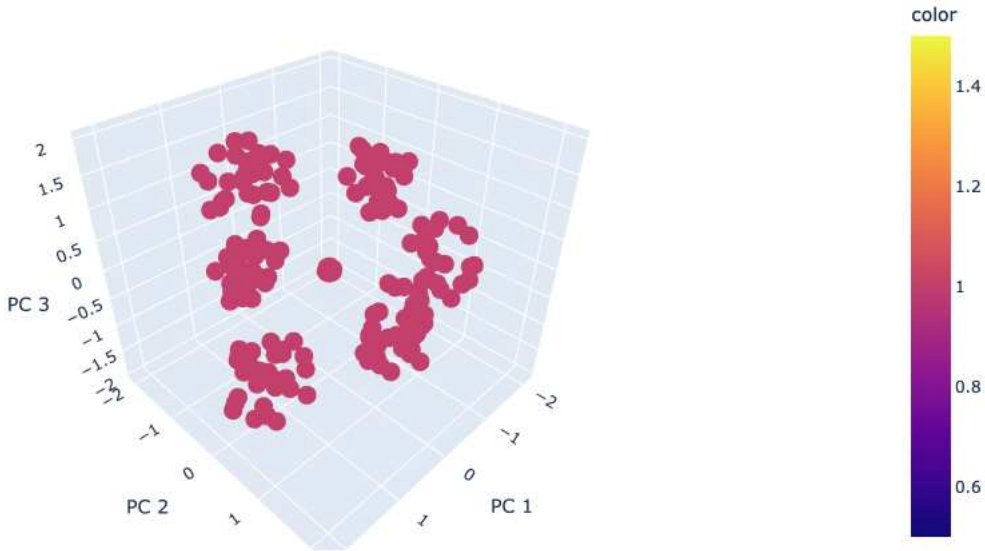
Algorithm Agglomerative Clustering (Ward) - classes = 7



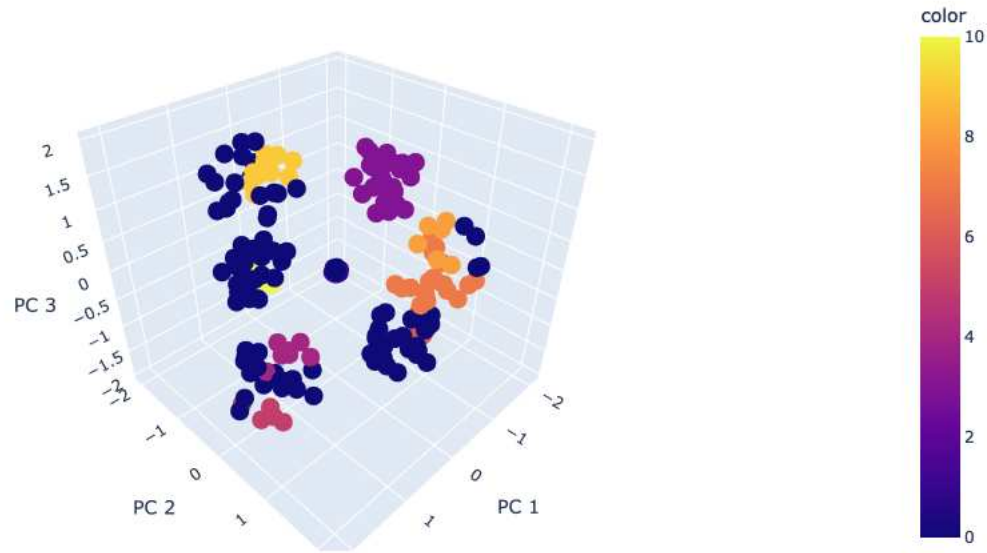

Algorithm DBSCAN - classes = 7

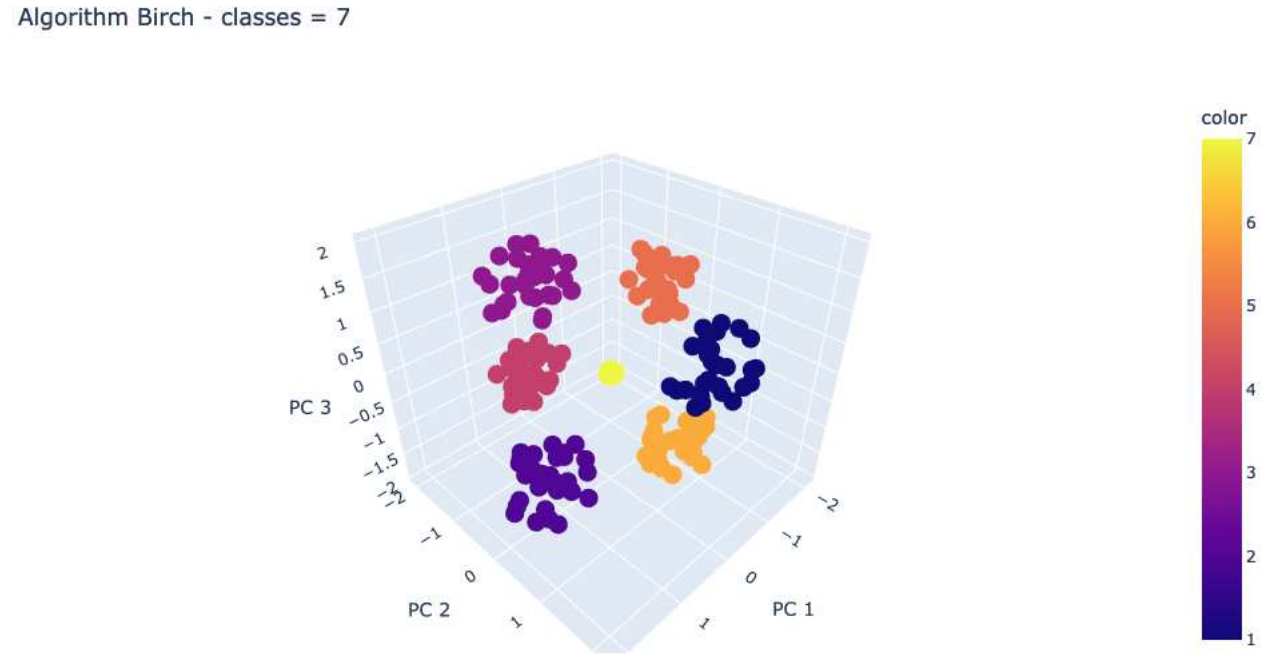


Algorithm Meanshift - classes = 1




Algorithm OPTICS - classes = 10





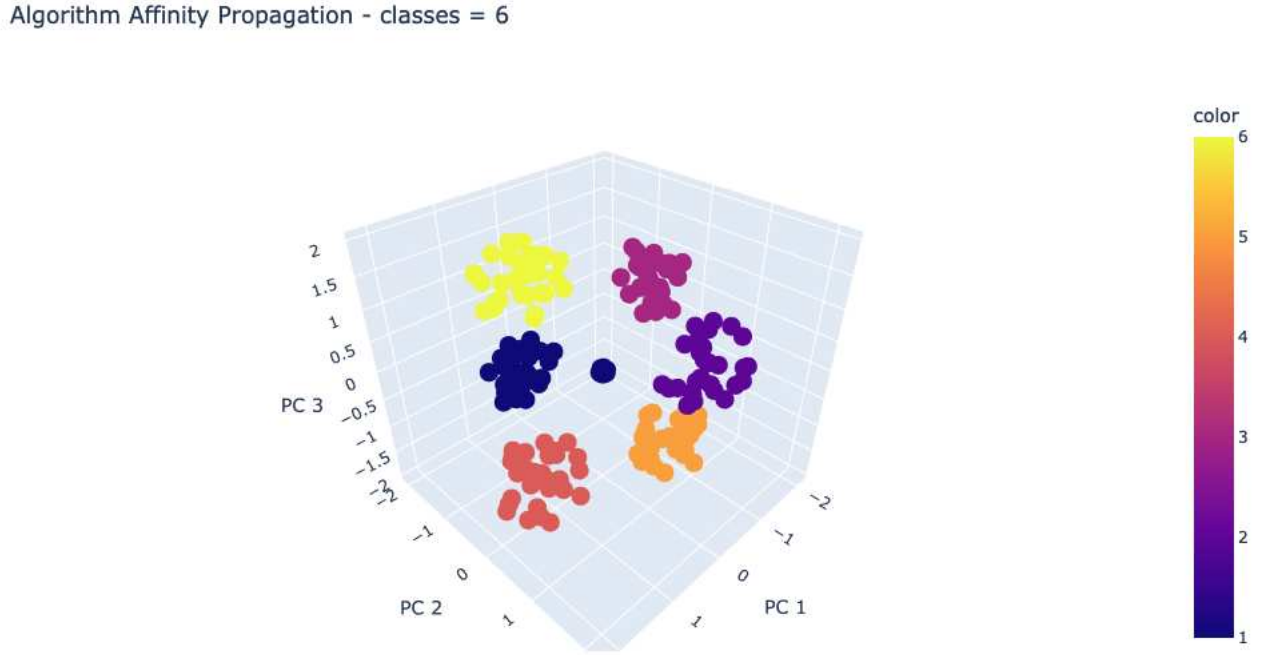




Tabella 3.14: hepta

| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.957 | 0.974 | 0.261 |
| KMeans | 1.0 | 1.0 | 0.019 |

*Continua nella prossima pagina*

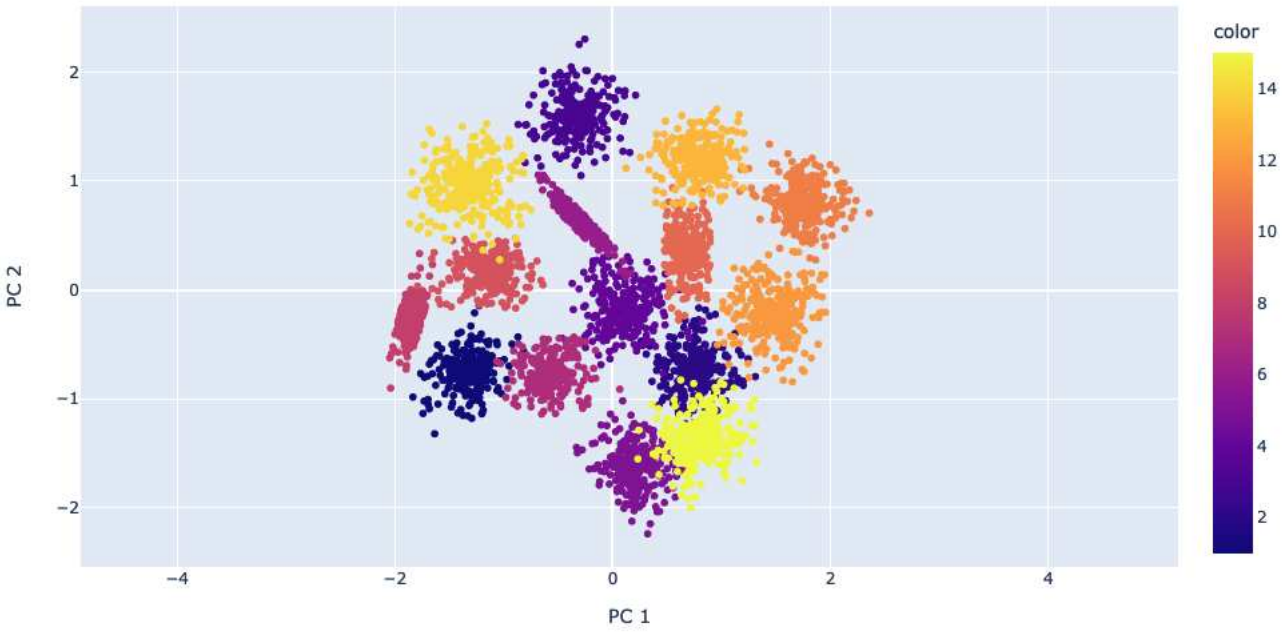
Tabella 3.14: *(Continued)*

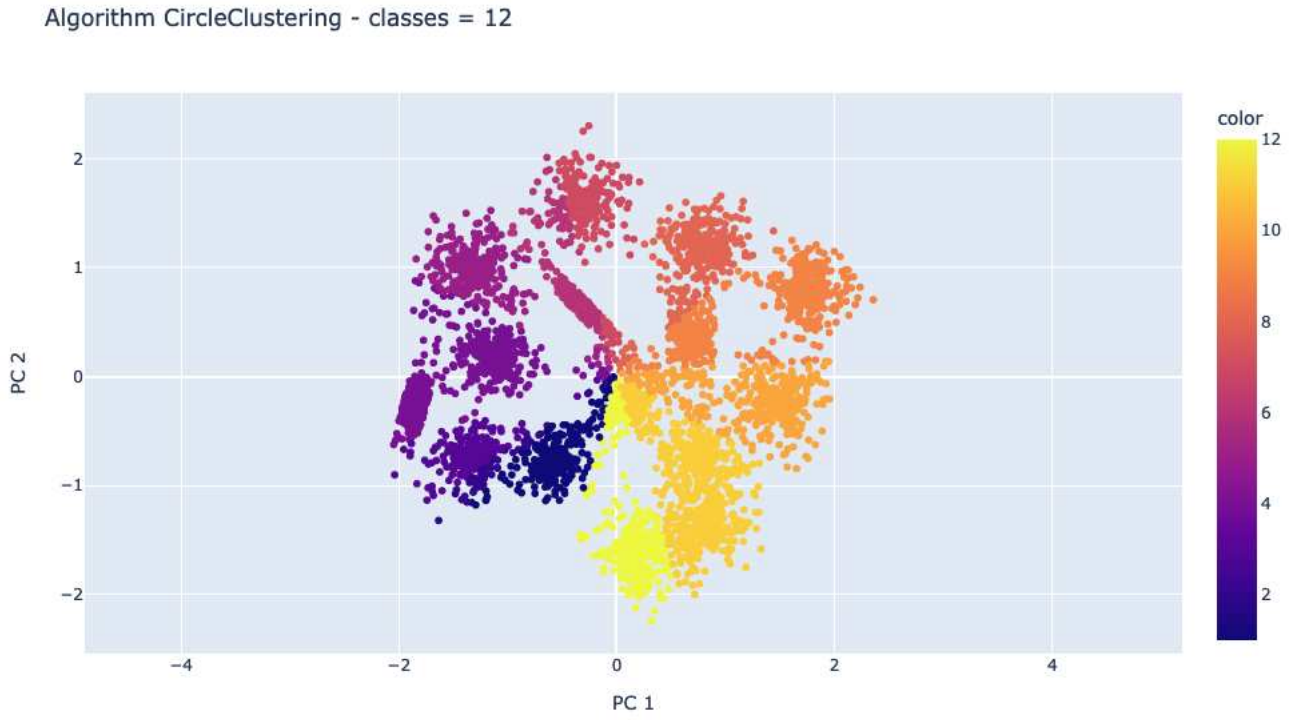
| | adjusted rand score | adjusted mutual info score | time cpu |
|---|---|---|---|
| Spectral C. | 1.0 | 1.0 | 0.157 |
| Ward | 1.0 | 1.0 | 0.001 |
| DBscan | 1.0 | 1.0 | 0.001 |
| MeanShift | 0.0 | 0.0 | 0.926 |
| Optics | 0.289 | 0.5897 | 0.129 |
| Birch | 1.0 | 1.0 | 0.004 |
| Affinity Prop. | 0.841 | 0.942 | 0.169 |

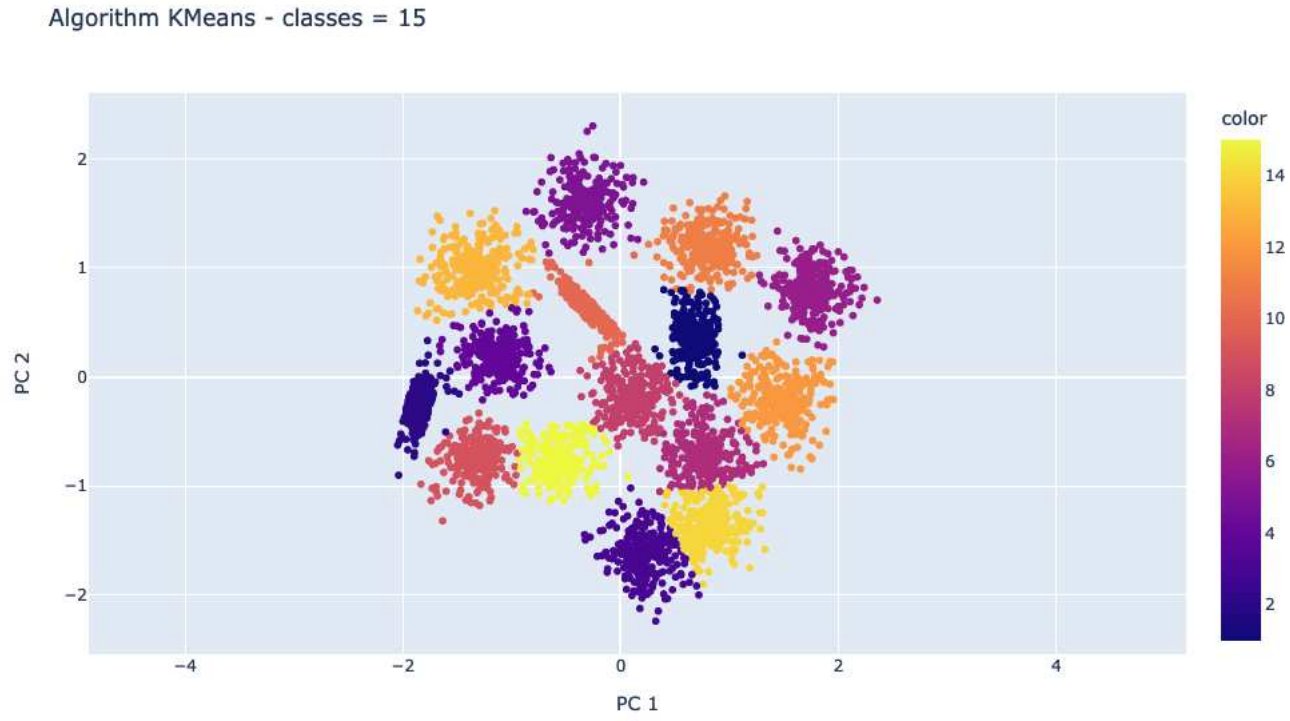
CircleClustering > Meanshift, OPTICS, Affinity Propagation
CircleClustering < KMeans, Spectral Clustering, Agglomerative Clustering (Ward), DBSCAN, Birch

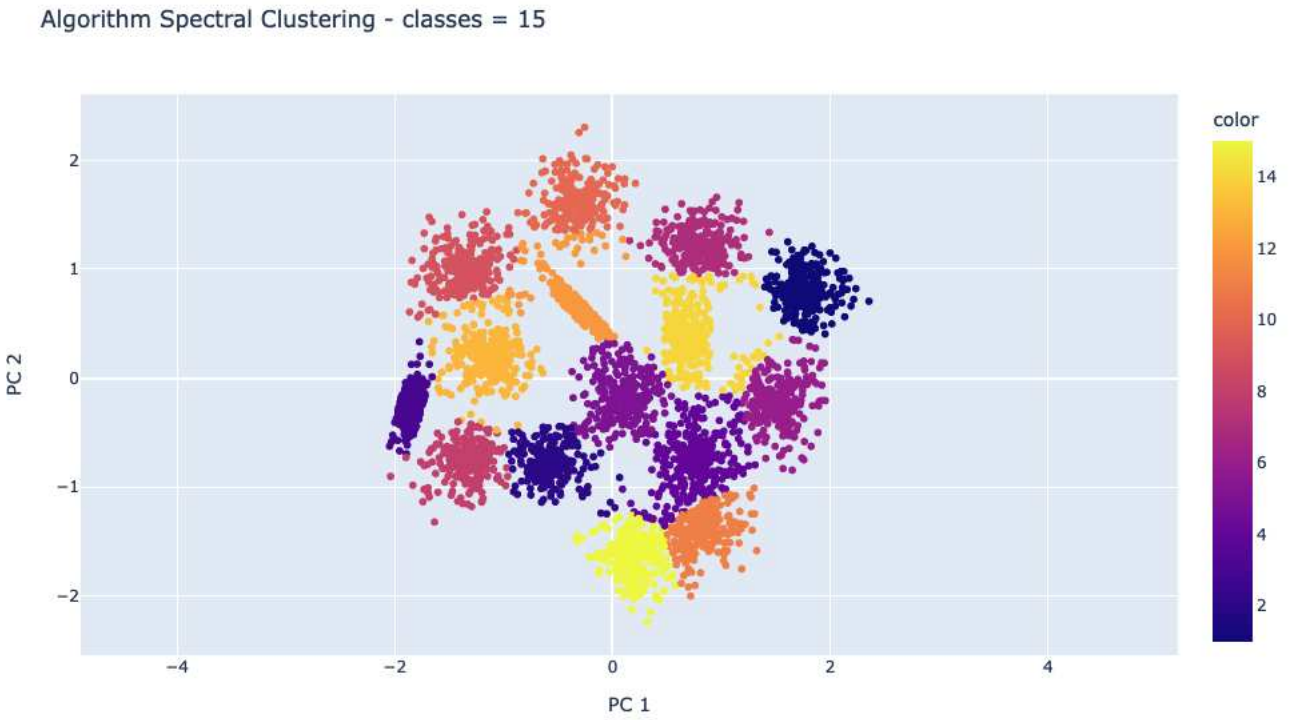
**Batteria: sipu**

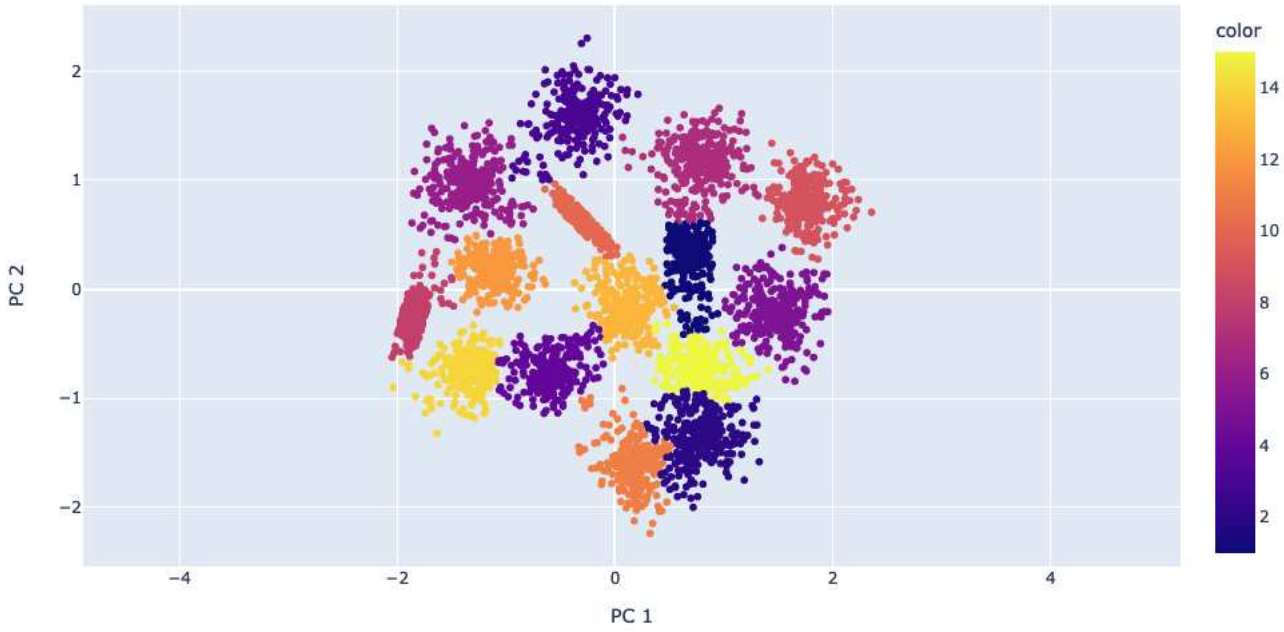
- Dataset: s2
  Samples = 5000, Features = 2, Classes = 15

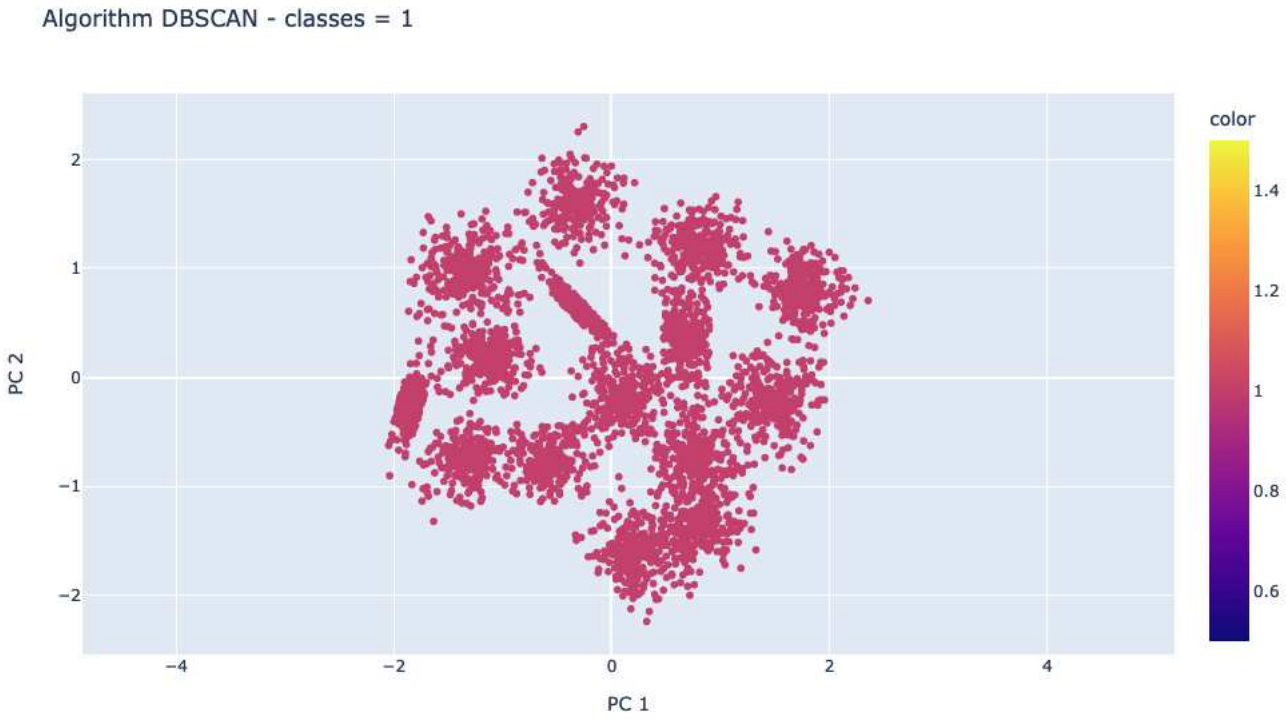


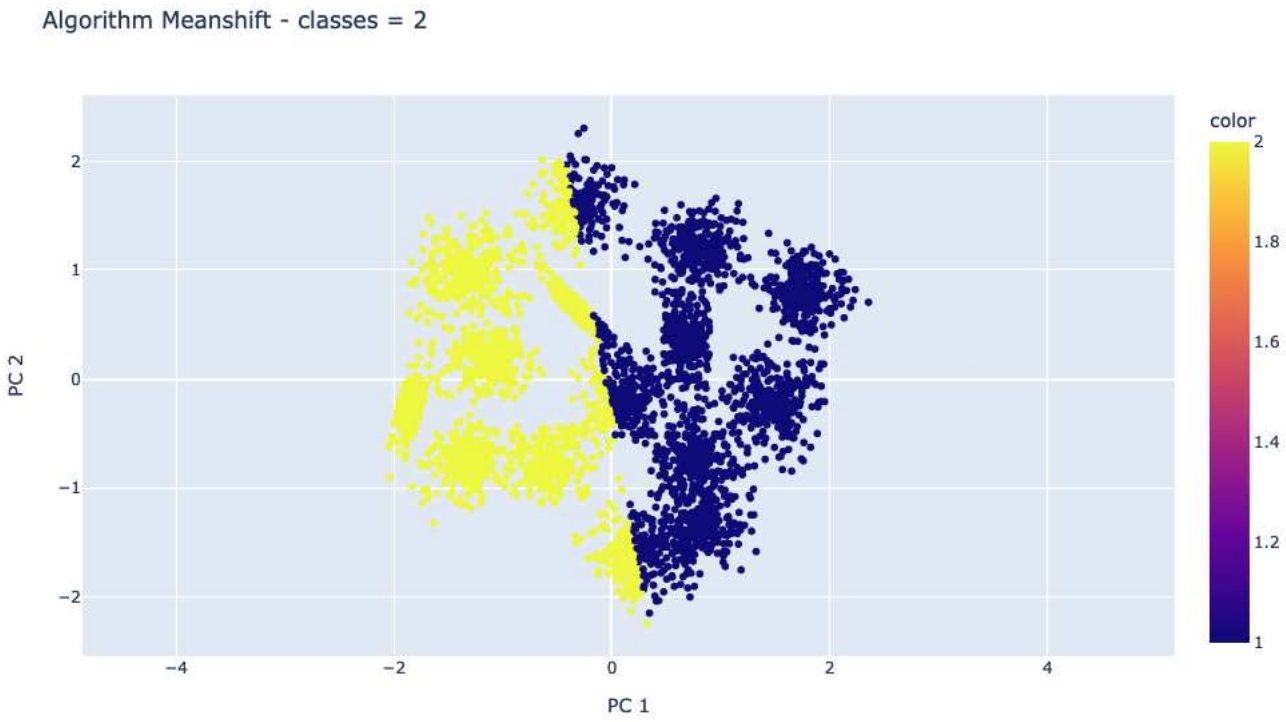

Algorithm CircleClustering - classes = 12
color
12
10
8
6
4
2
PC 2
PC 1
2
1
0
−1
−2
−4
−2
0
2
4

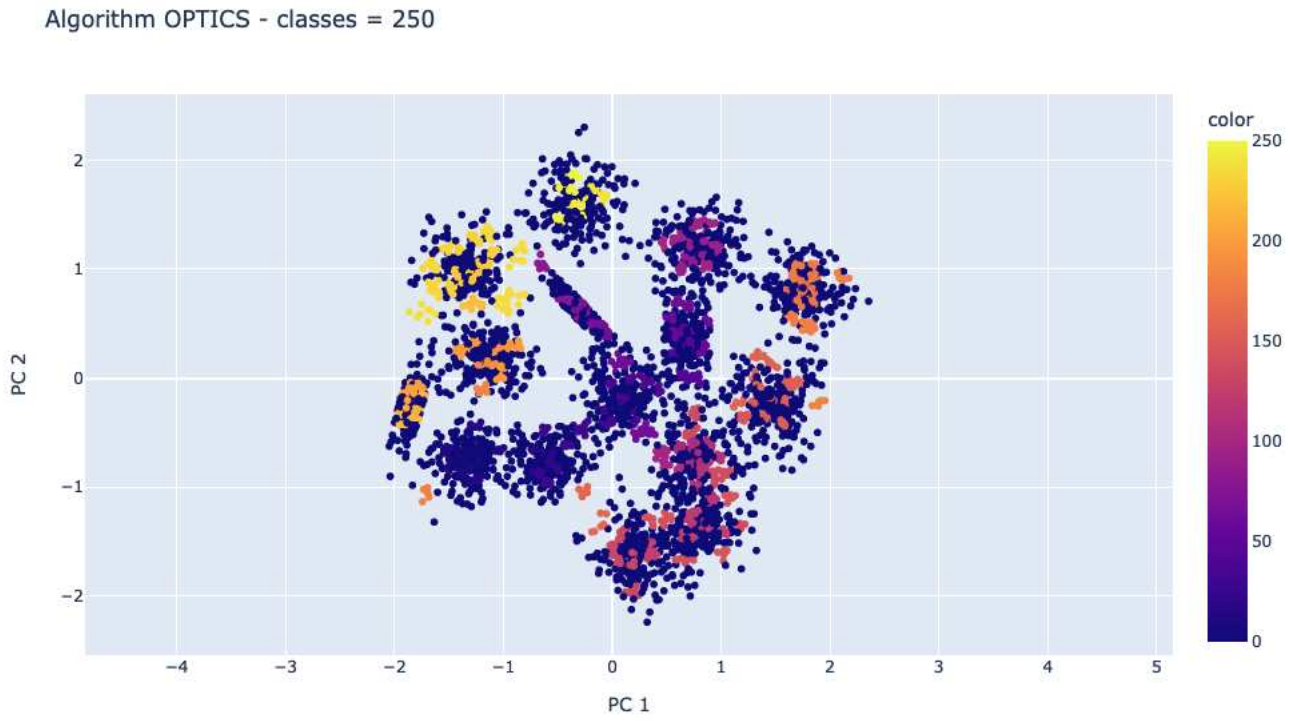

Algorithm KMeans - classes = 15
color
14
12
10
8
6
4
2
PC 2
PC 1
2
1
0
−1
−2
−4
−2
0
2
4

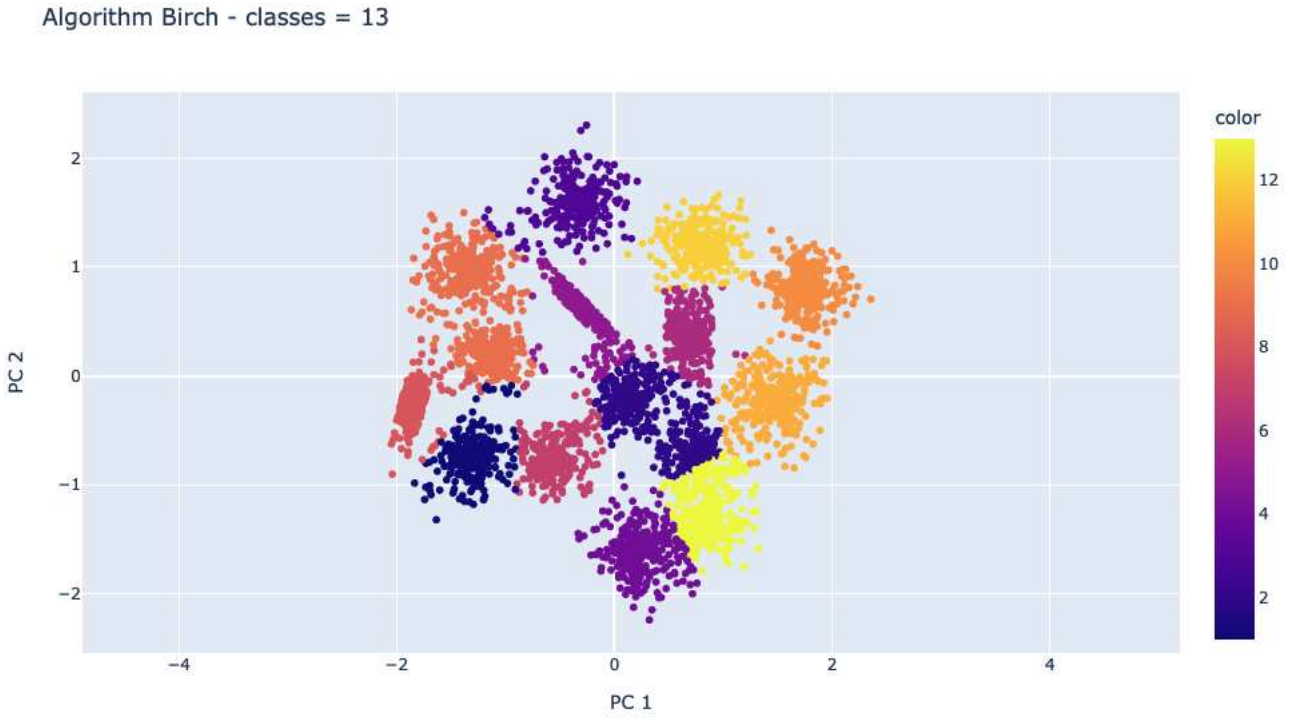

Algorithm Spectral Clustering - classes = 15
color
14
12
10
8
6
4
2
PC 2
2
1
0
−1
−2
−4
−2
0
2
4
PC 1


color
14
12
10
8
6
4
2
PC 2
2
1
0
−1
−2
−4
−2
0
2
4
PC 1


Algorithm DBSCAN - classes = 1
color
1.4
1.2
1
0.8
0.6
PC 2
PC 1
2
1
0
−1
−2
−4
−2
0
2
4


Algorithm Meanshift - classes = 2
color
2
1.8
1.6
1.4
1.2
1
PC 2
PC 1
2
1
0
−1
−2
−4
−2
0
2
4


Algorithm OPTICS - classes = 250
color
250
200
150
100
50
0
PC 2
PC 1
2
1
0
−1
−2
−4
−3
−2
−1
0
1
2
3
4
5


Algorithm Birch - classes = 13
color
12
10
8
6
4
2
PC 2
PC 1
2
1
0
−1
−2
−4
−2
0
2
4

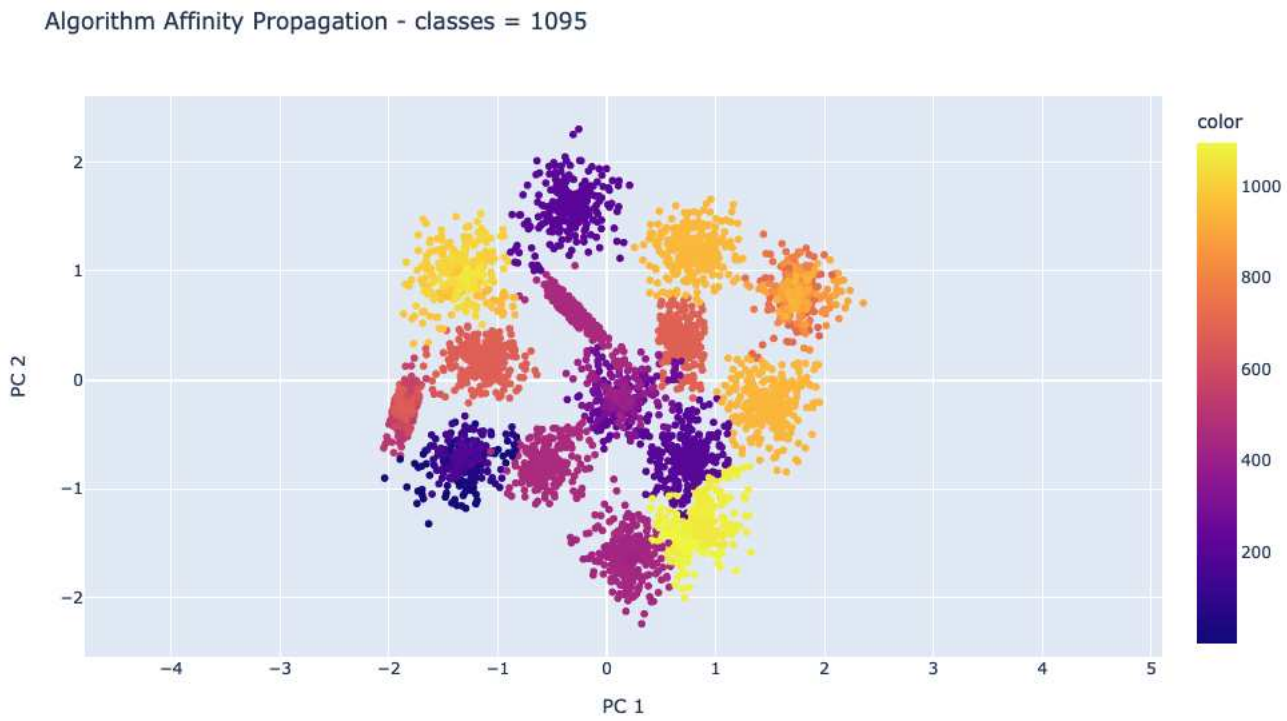
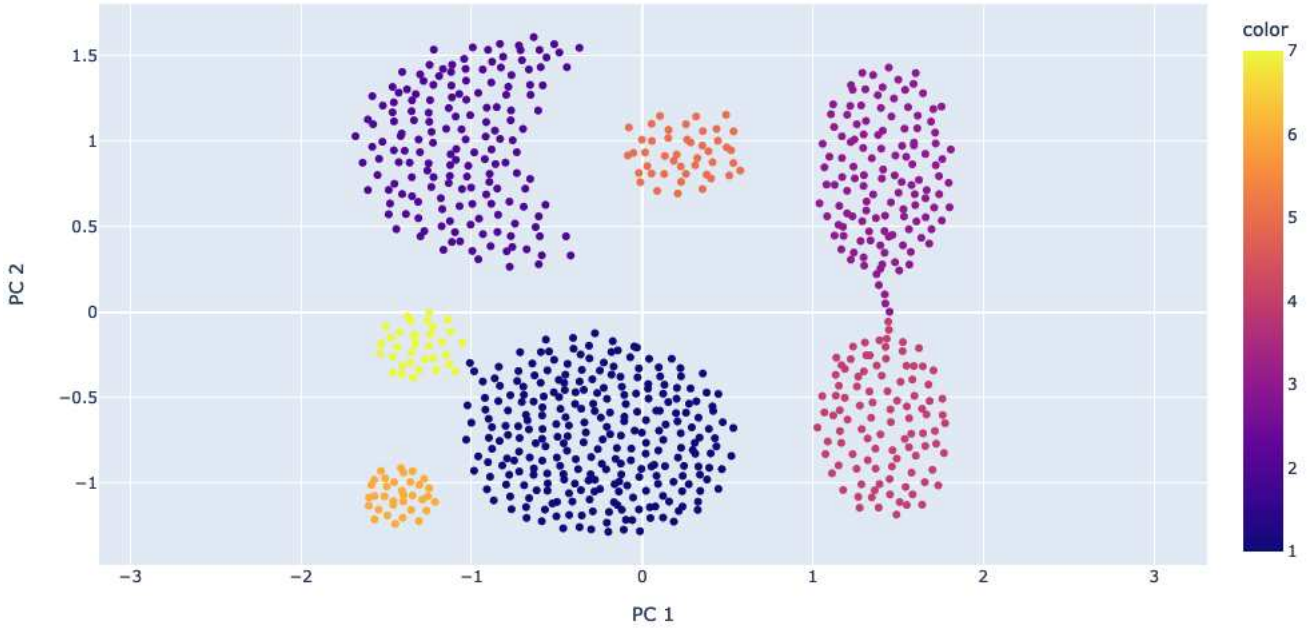



Tabella 3.15: s2

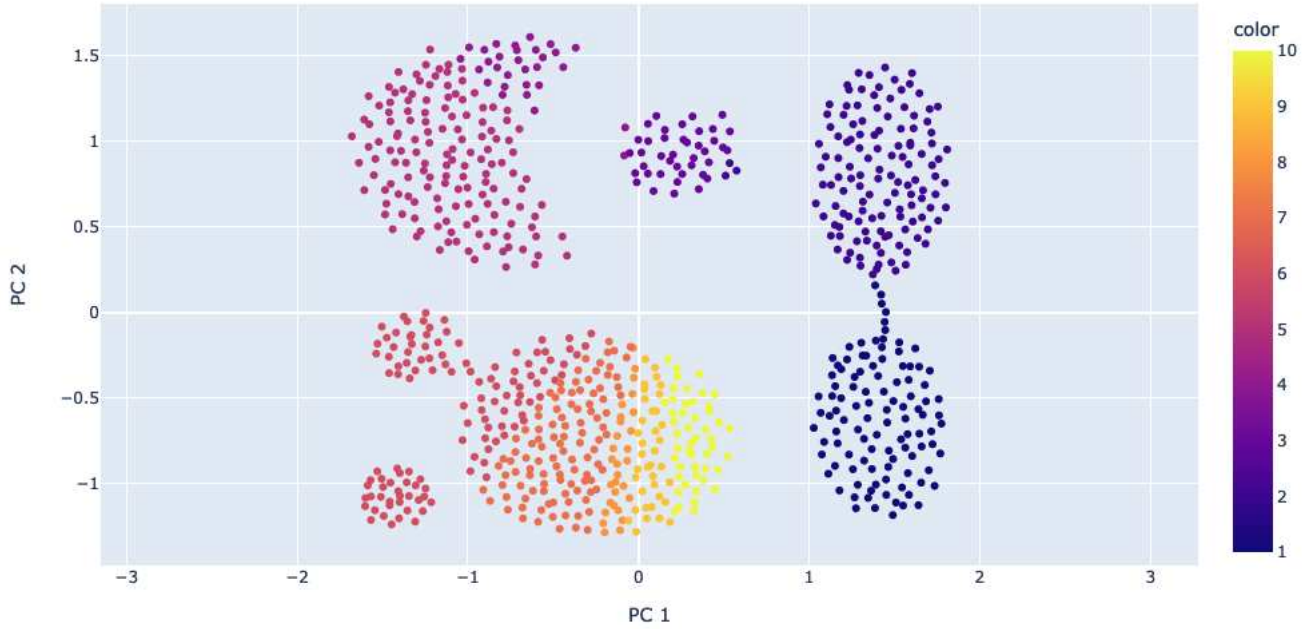
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.625 | 0.787 | 4.700 |
| KMeans | 0.937 | 0.945 | 1.037 |
| Spectral C. | 0.884 | 0.914 | 5.667 |
| Ward | 0.905 | 0.926 | 0.802 |
| DBscan | 0.0 | 0.0 | 0.060 |
| MeanShift | 0.108 | 0.325 | 28.888 |
| Optics | 0.004 | 0.297 | 3.749 |
| Birch | 0.802 | 0.890 | 0.122 |
| Affinity Prop. | 0.376 | 0.578 | 85.975 |

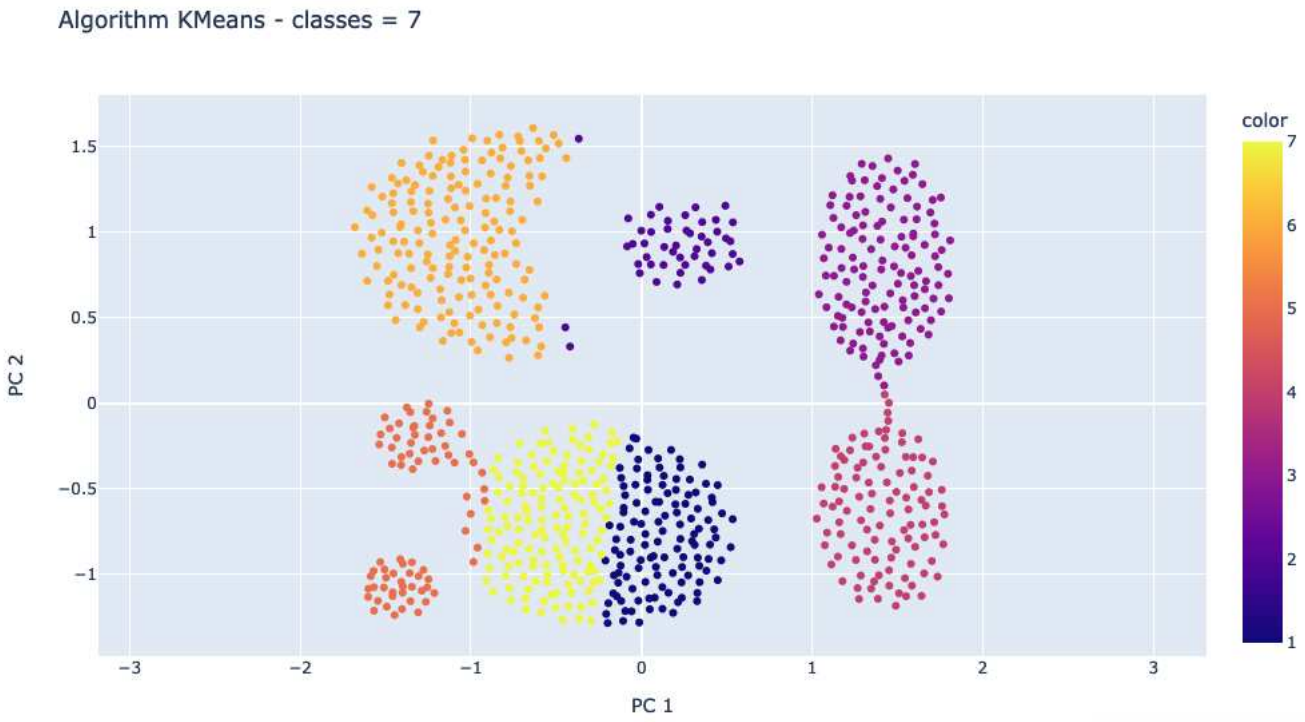
CircleClustering > DBSCAN, Meanshift, OPTICS, Affinity Propagation
CircleClustering < KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Birch

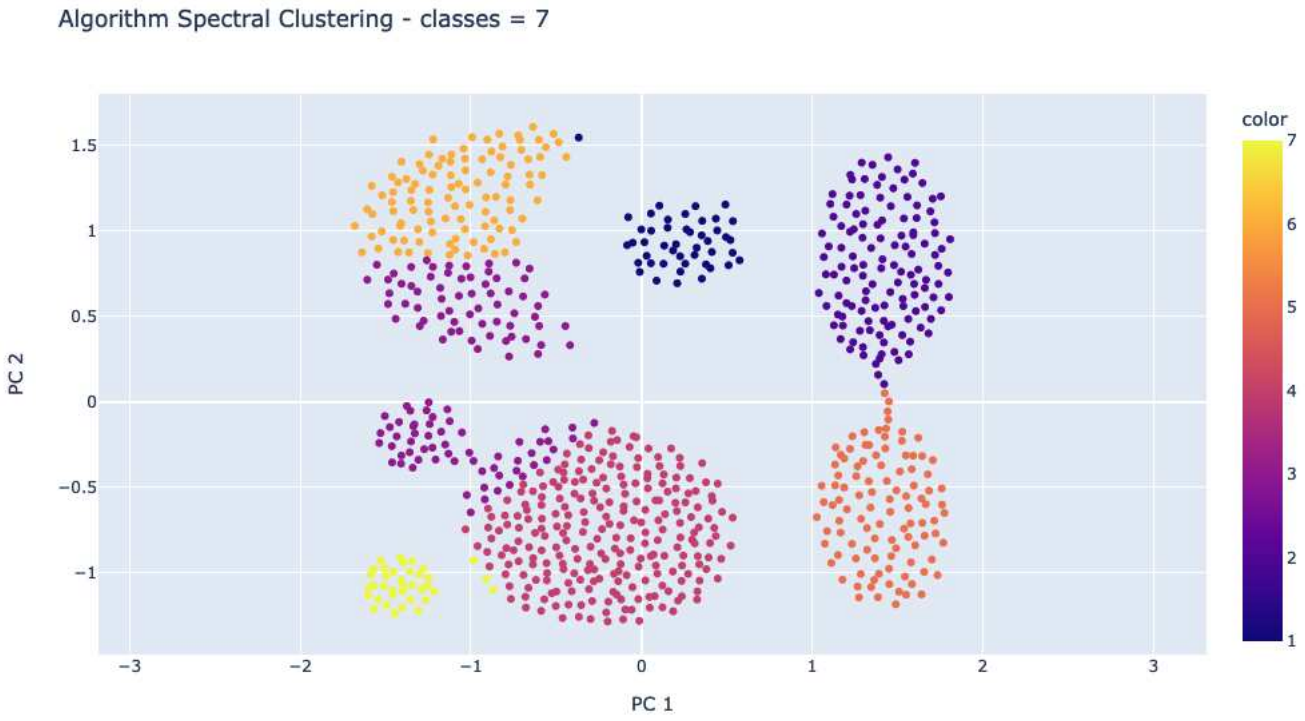
- Dataset: aggregation
  Samples = 788, Features = 2, Classes = 7

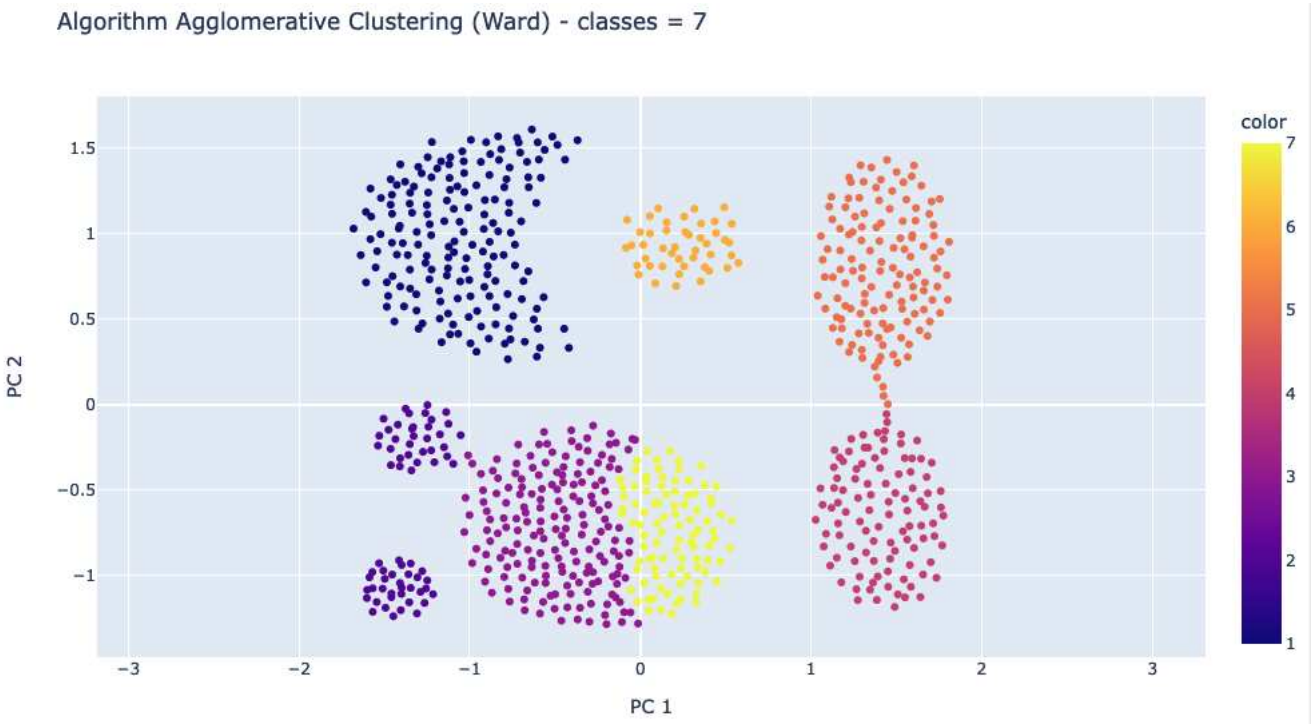


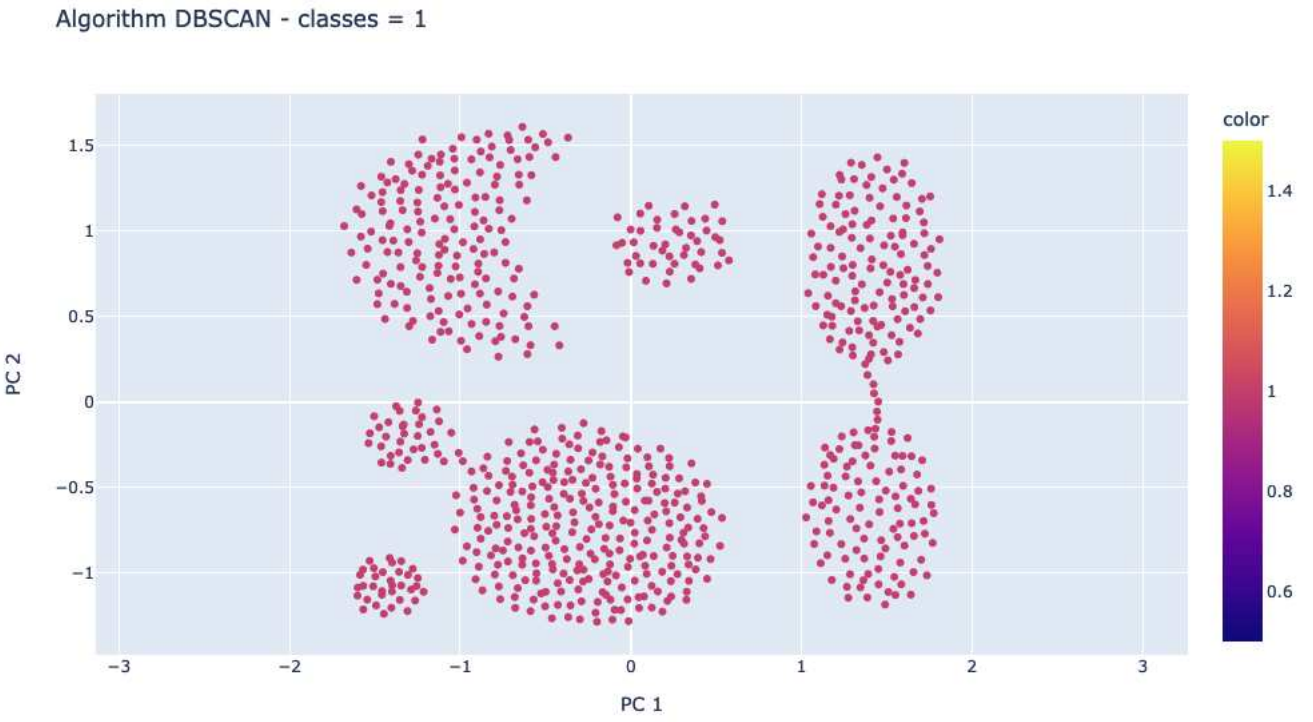

Algorithm CircleClustering - classes = 10

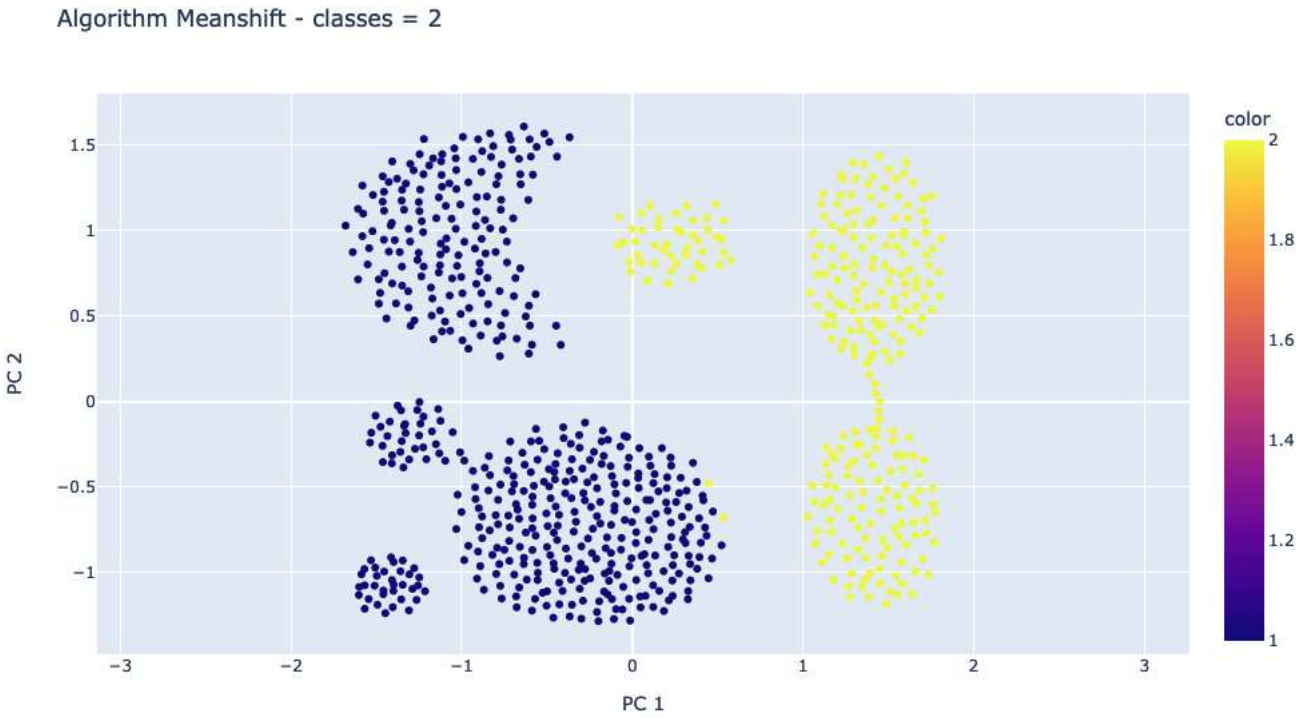


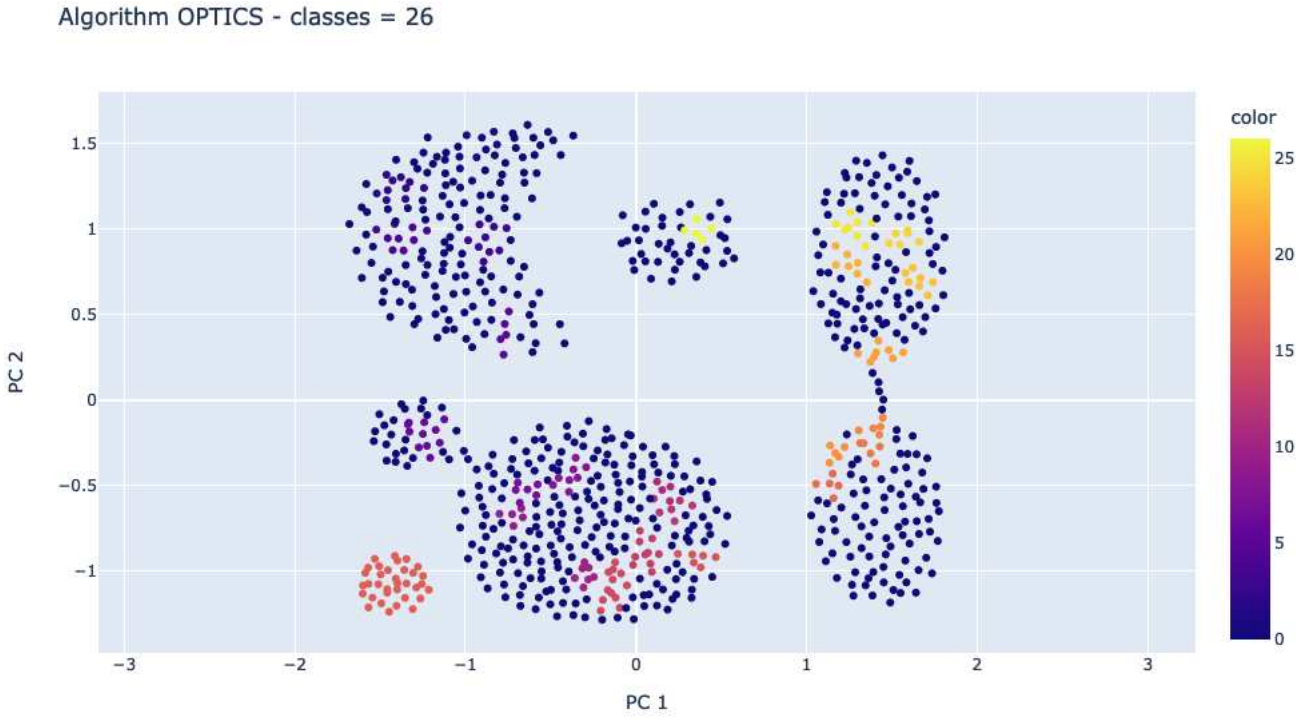

Algorithm KMeans - classes = 7
color
7
6
5
4
3
2
1
PC 2
1.5
1
0.5
0
−0.5
−1
−3
−2
−1
0
1
2
3
PC 1

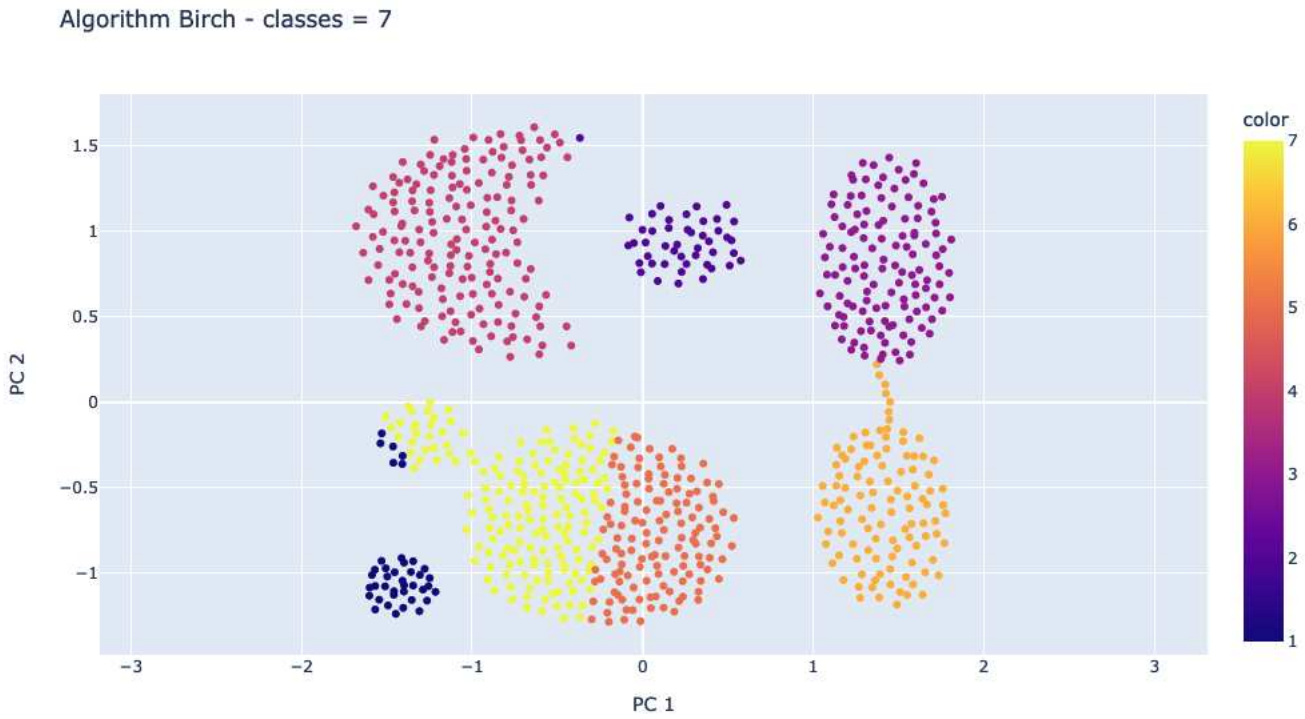

Algorithm Spectral Clustering - classes = 7
color
7
6
5
4
3
2
1
PC 2
1.5
1
0.5
0
−0.5
−1
−3
−2
−1
0
1
2
3
PC 1


Algorithm Agglomerative Clustering (Ward) - classes = 7
color
7
6
5
4
3
2
1
PC 2
PC 1
1.5
1
0.5
0
−0.5
−1
−3
−2
−1
0
1
2
3


Algorithm DBSCAN - classes = 1
color
1.4
1.2
1
0.8
0.6
PC 2
PC 1
1.5
1
0.5
0
−0.5
−1
−3
−2
−1
0
1
2
3


Algorithm Meanshift - classes = 2
color
2
1.8
1.6
1.4
1.2
1
1.5
1
0.5
0
−0.5
−1
PC 2
−3
−2
−1
0
1
2
3
PC 1


Algorithm OPTICS - classes = 26
color
25
20
15
10
5
0
1.5
1
0.5
0
−0.5
−1
PC 2
−3
−2
−1
0
1
2
3
PC 1


Algorithm Birch - classes = 7
color
7
6
5
4
3
2
1
PC 2
1.5
1
0.5
0
−0.5
−1
−3
−2
−1
0
1
2
3
PC 1

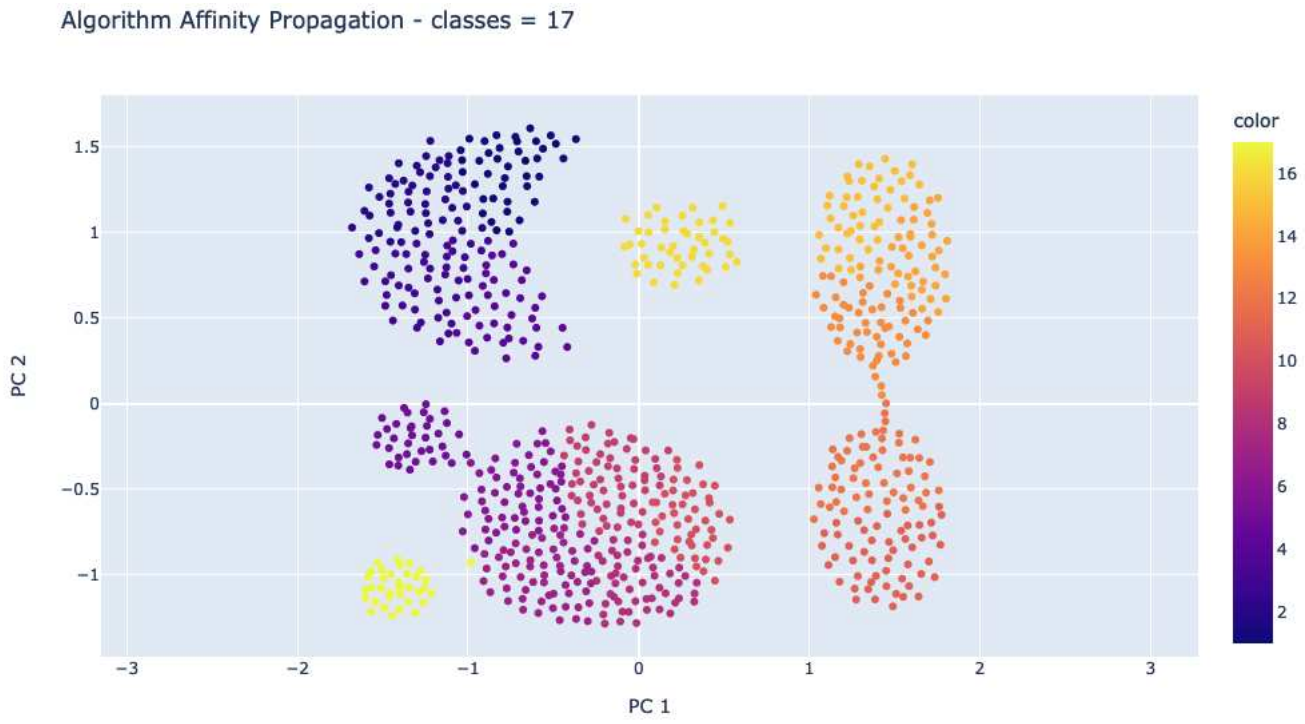

Algorithm Affinity Propagation - classes = 17
color
16
14
12
10
8
6
4
2
PC 2
1.5
1
0.5
0
−0.5
−1
−3
−2
−1
0
1
2
3
PC 1

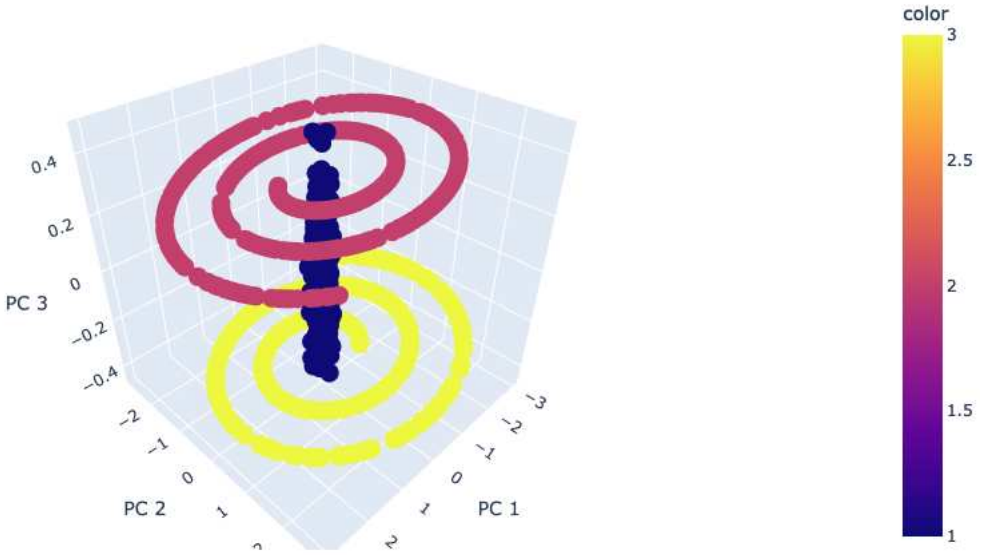
Tabella 3.16: aggregation

| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.575 | 0.767 | 0.304 |
| KMeans | 0.76 | 0.875 | 0.086 |
| Spectral C. | 0.805 | 0.858 | 0.283 |
| Ward | 0.813 | 0.919 | 0.010 |
| DBscan | 0.0 | 0.0 | 0.004 |
| MeanShift | 0.372 | 0.540 | 3.942 |
| Optics | 0.0447 | 0.2933 | 0.472 |
| Birch | 0.749 | 0.872 | 0.0134 |
| Affinity Prop. | 0.368 | 0.735 | 1.228 |

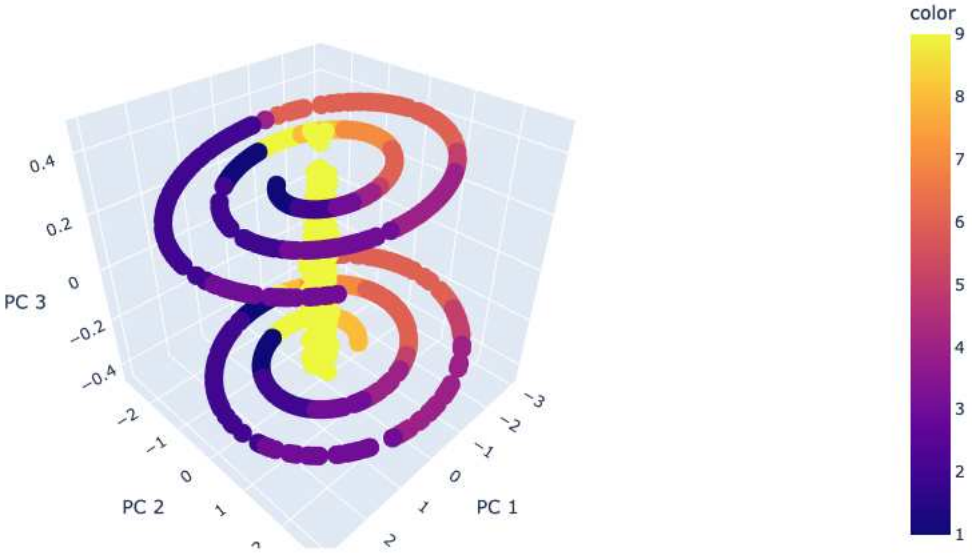
CircleClustering > DBSCAN, Meanshift, OPTICS, Affinity Propagation
CircleClustering < KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Birch

**Batteria: wut**

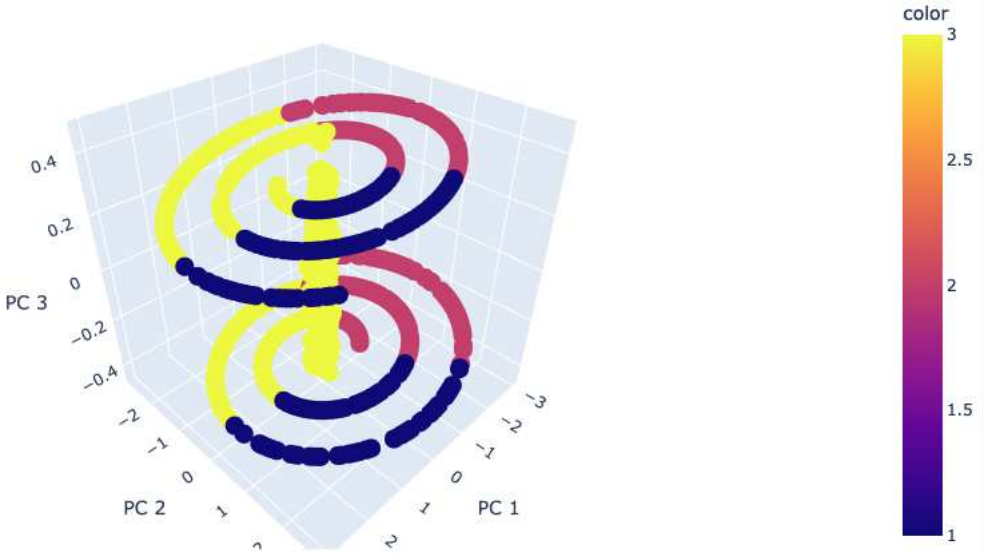
- Dataset: mk4
  Samples = 1500, Features = 3, Classes = 3



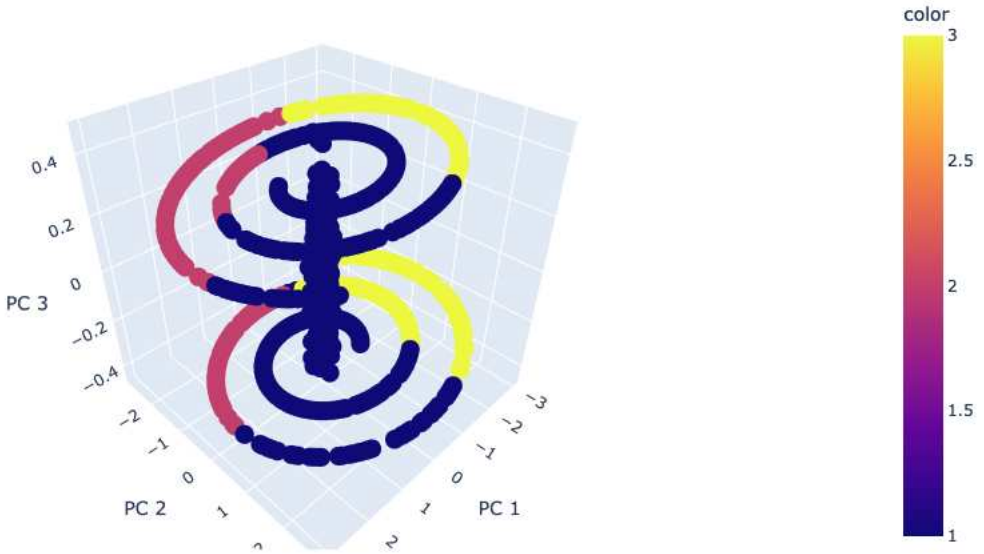
Algorithm CircleClustering - classes = 9



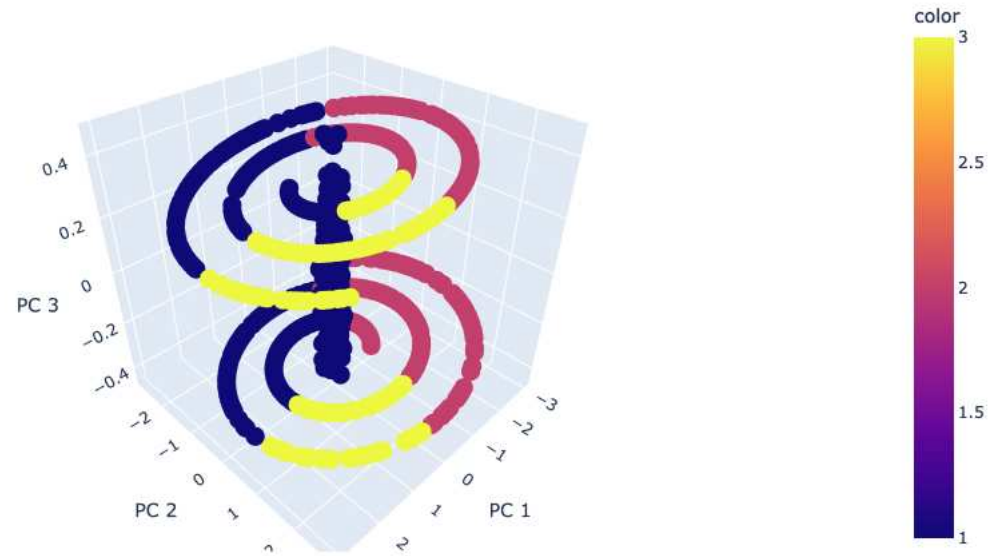

Algorithm KMeans - classes = 3



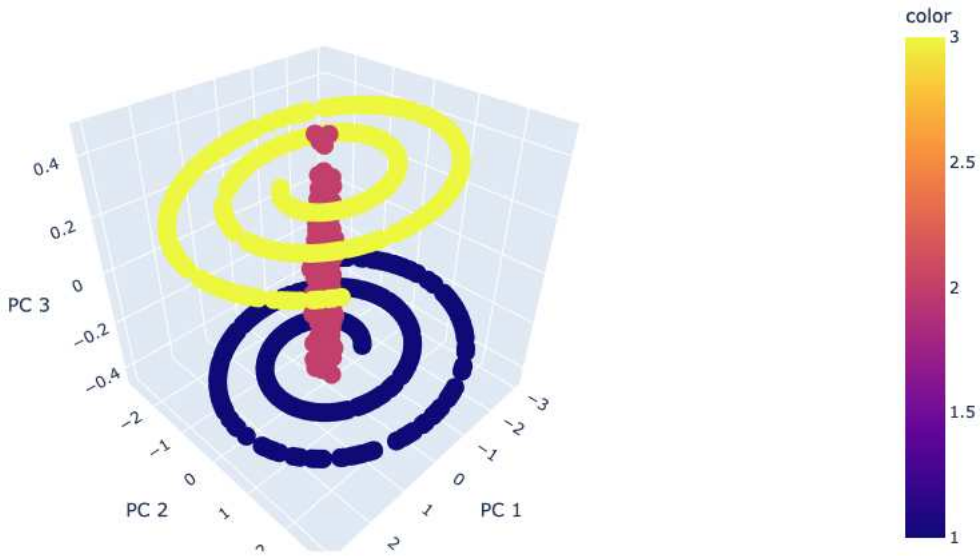
Algorithm Spectral Clustering - classes = 3



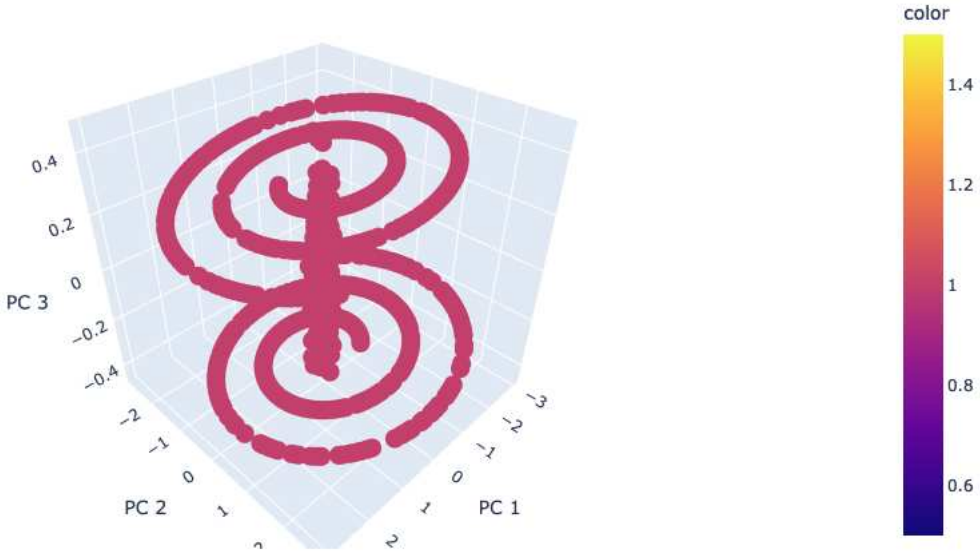

Algorithm Agglomerative Clustering (Ward) - classes = 3



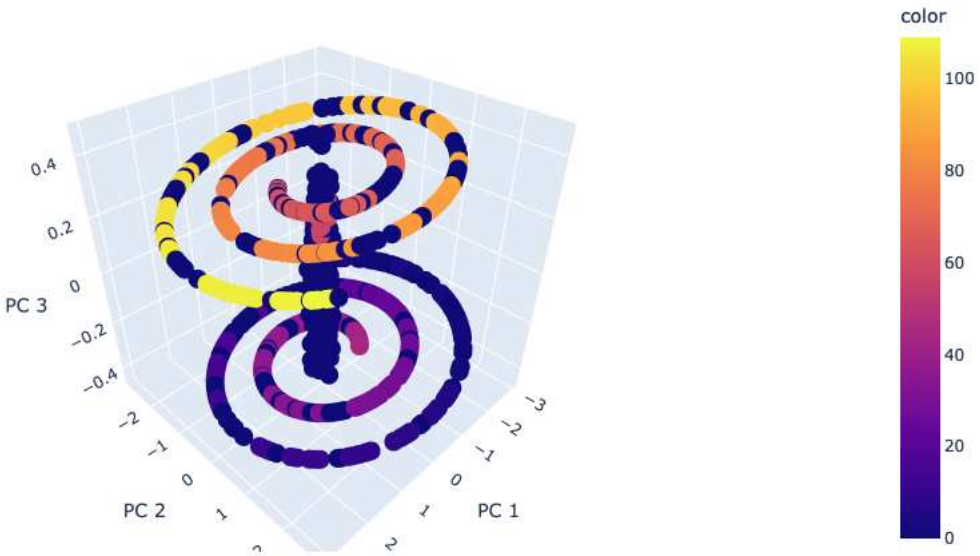
Algorithm DBSCAN - classes = 3



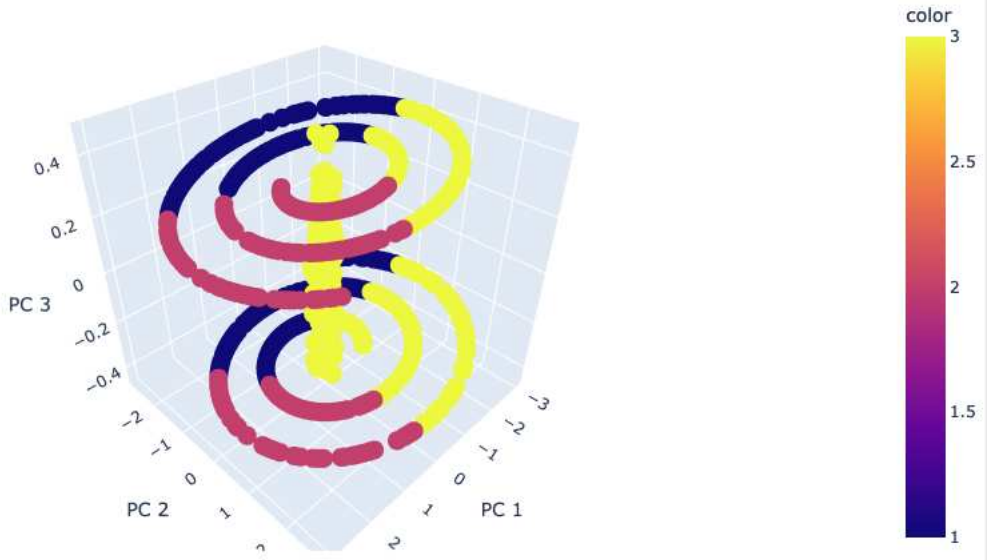

Algorithm Meanshift - classes = 1



Algorithm OPTICS - classes = 109




Algorithm Birch - classes = 3



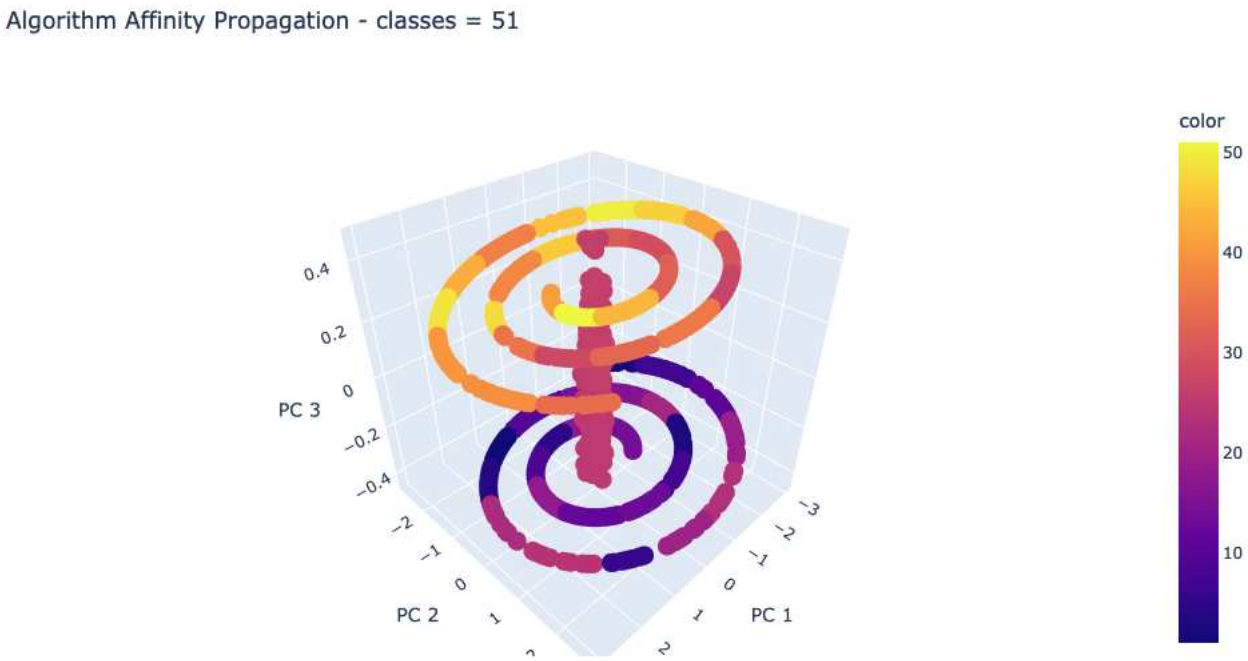
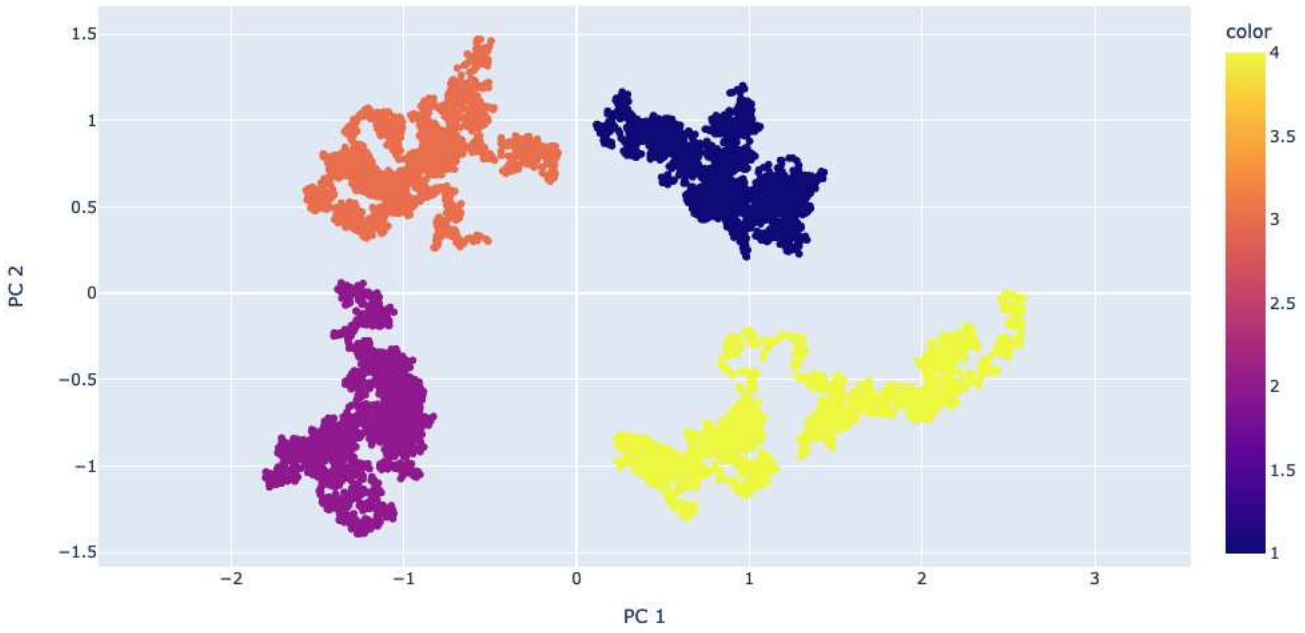



Tabella 3.17: aggregation

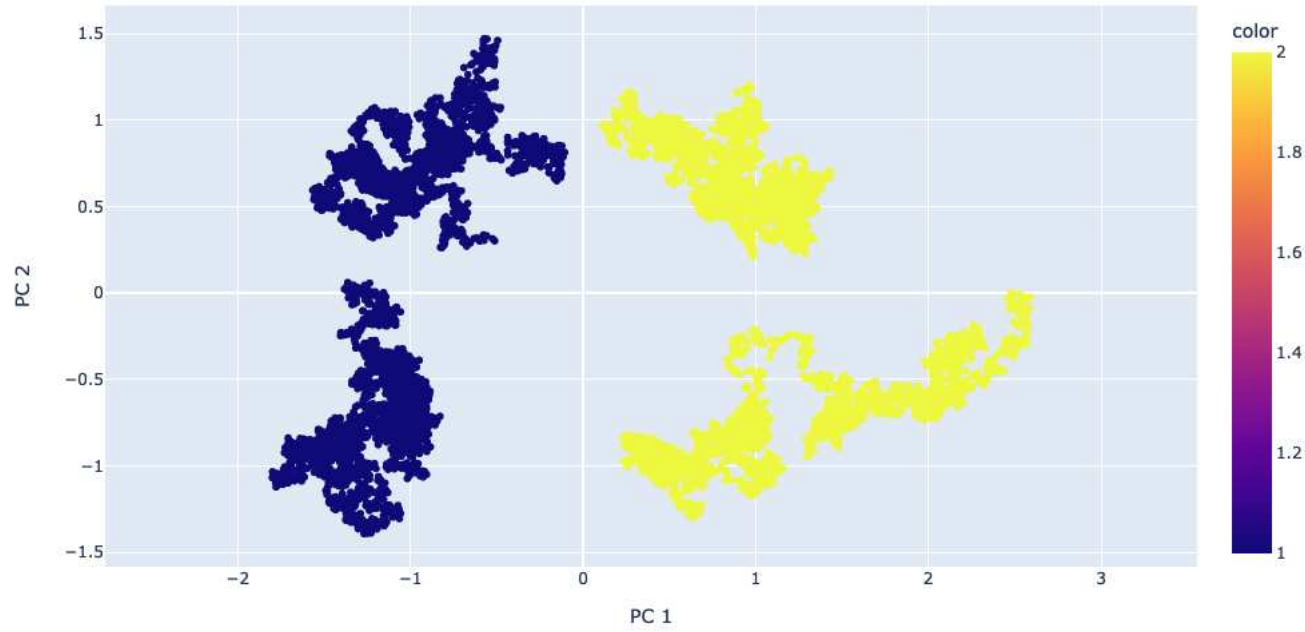
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.383 | 0.349 | 3.296 |
| KMeans | 0.201 | 0.245 | 0.171 |
| Spectral C. | 0.0859 | 0.1645 | 0.4934 |
| Ward | 0.1885 | 0.2357 | 0.045 |
| DBscan | 1.0 | 1.0 | 0.0134 |
| MeanShift | 0.0 | 0.0 | 7.467 |
| Optics | 0.1061 | 0.280 | 0.969 |
| Birch | 0.249 | 0.284 | 0.025 |
| Affinity Prop. | 0.244 | 0.477 | 7.81 |

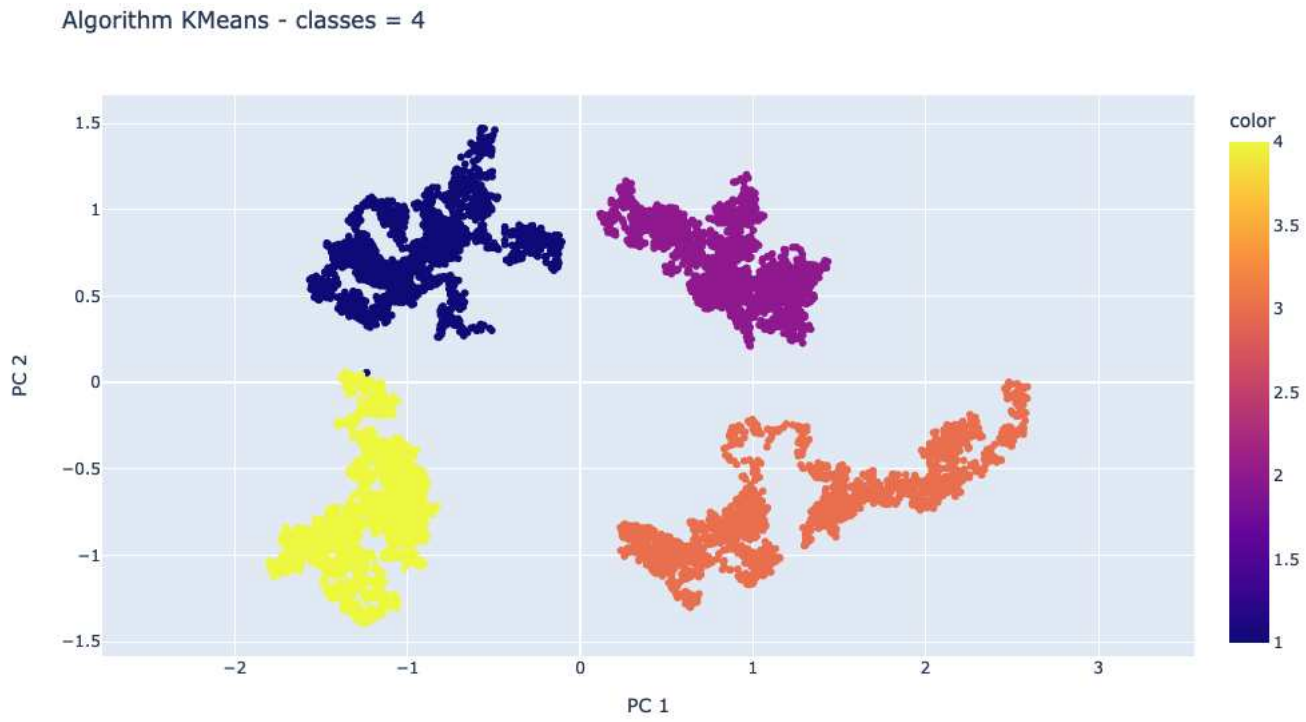
CircleClustering > KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Meanshift, Birch, OPTICS, Affinity Propagation CircleClustering < DBSCAN

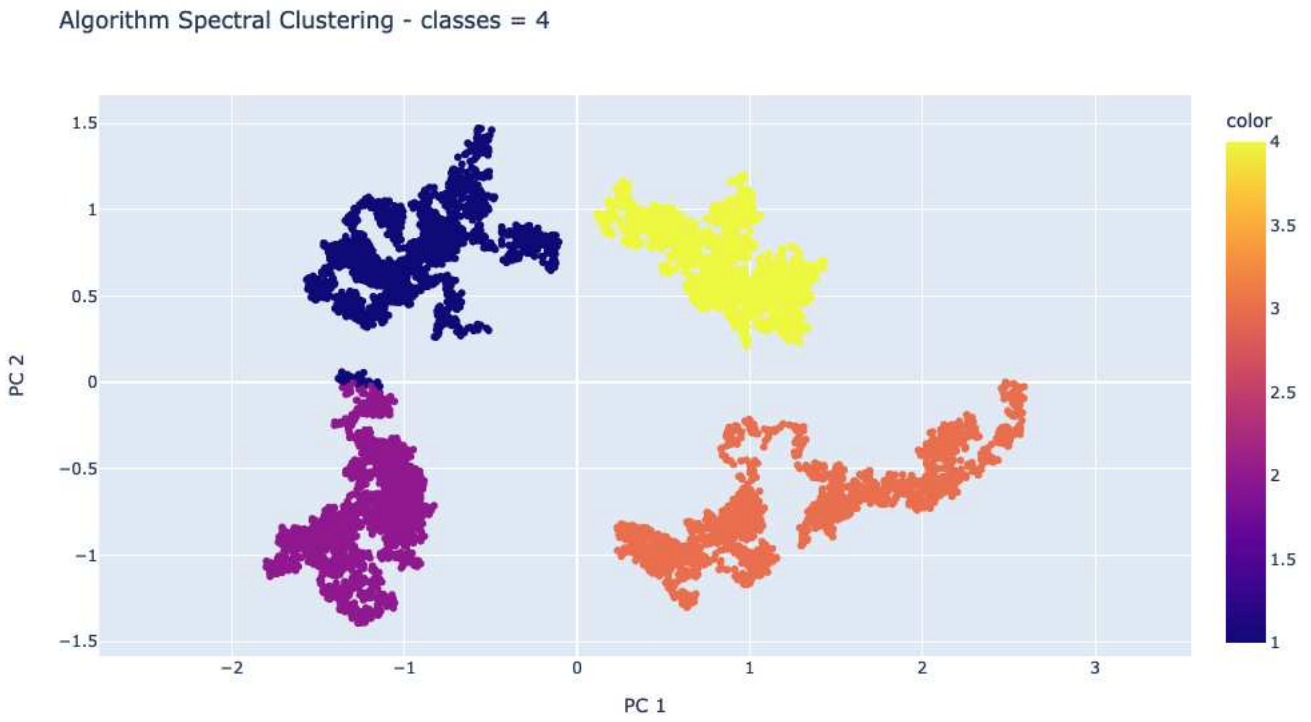
- Dataset: trajectories
  Samples = 10000, Features = 2, Classes = 4

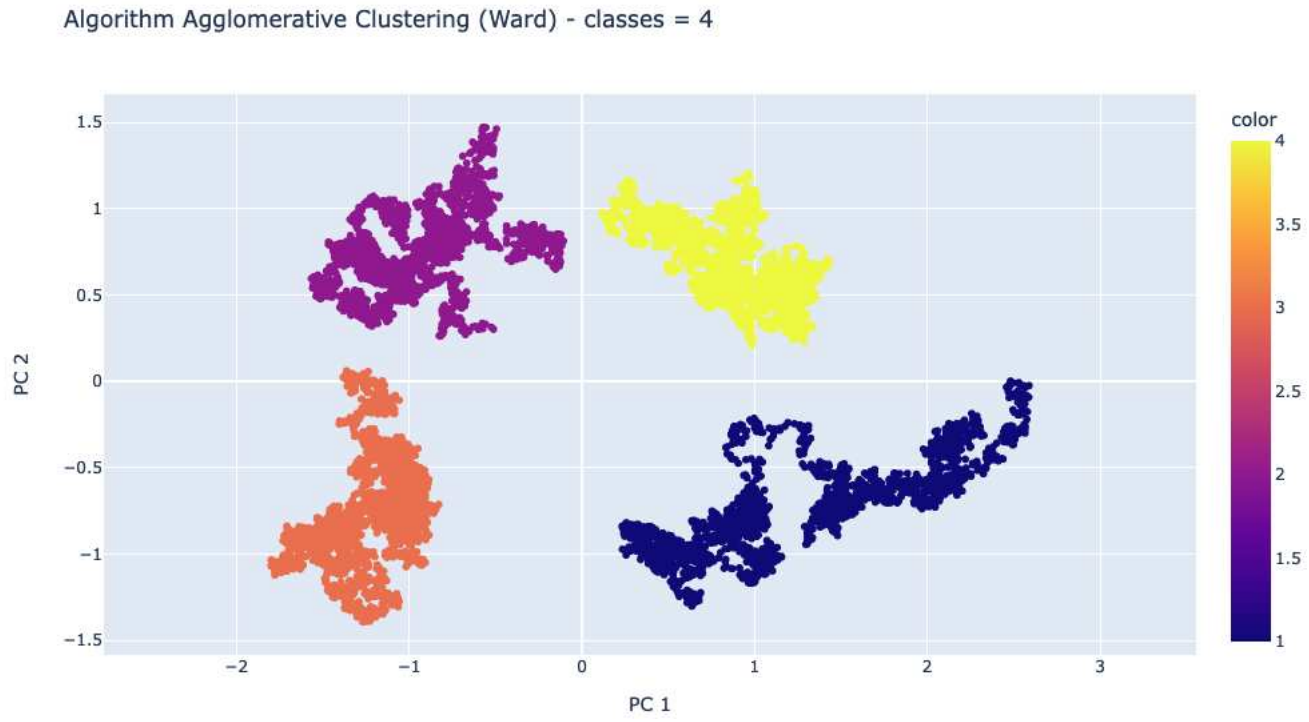


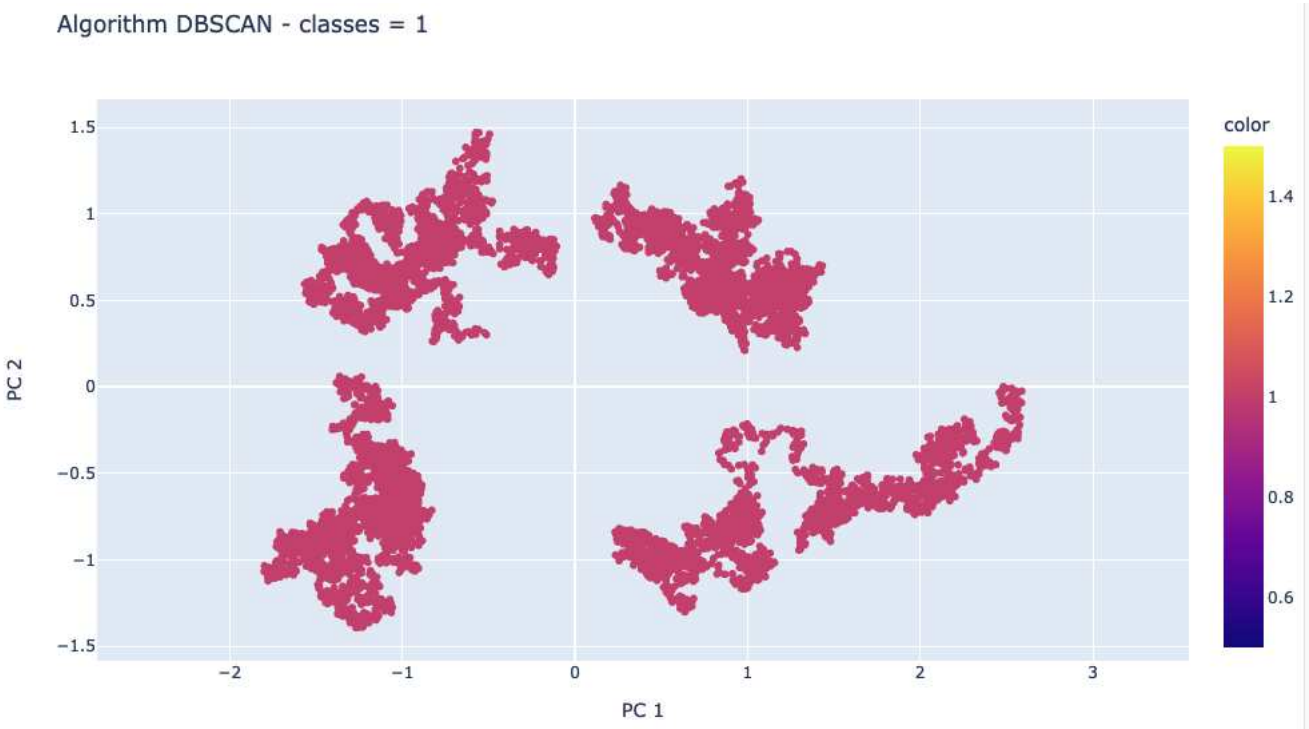

Algorithm CircleClustering - classes = 2

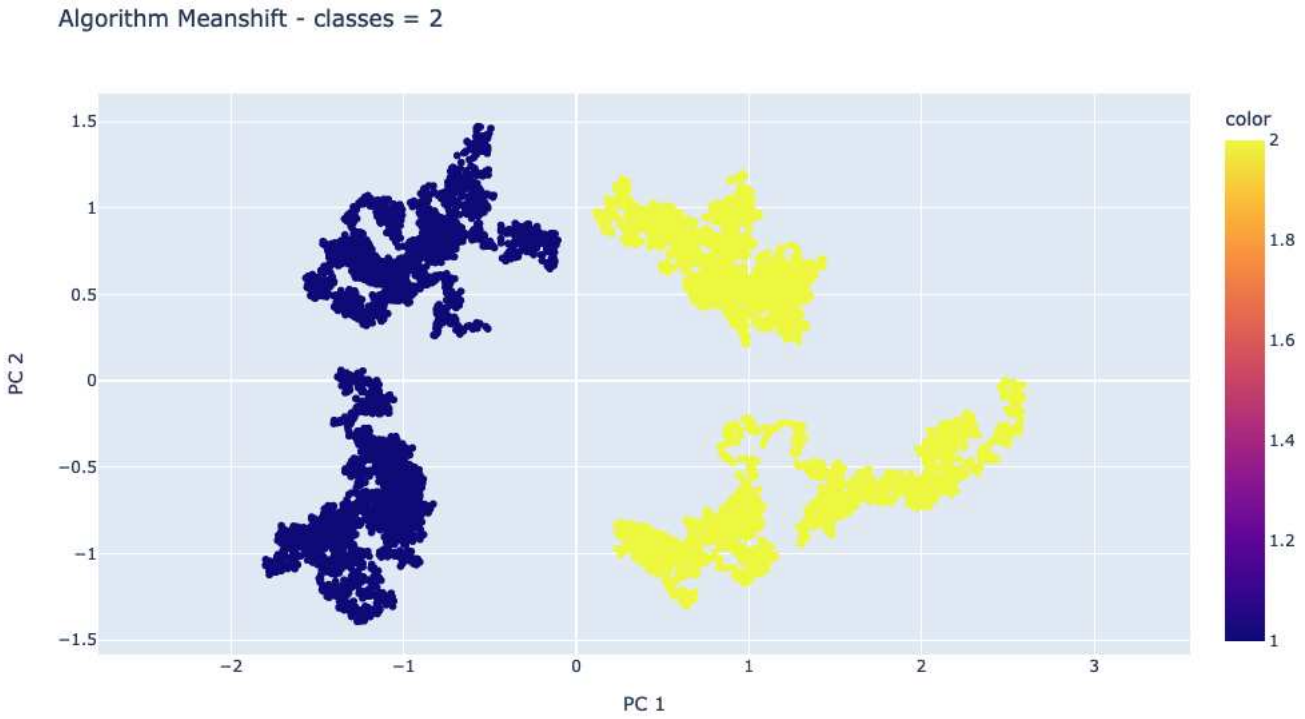


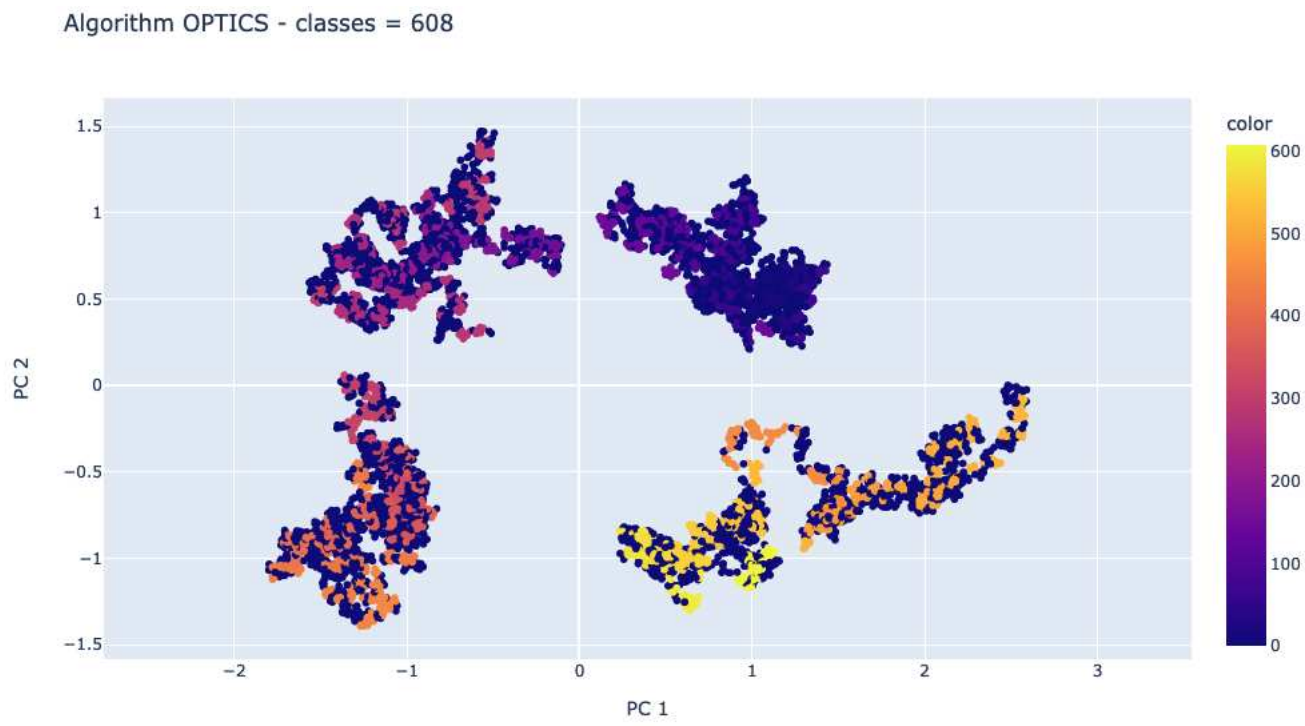

Algorithm KMeans - classes = 4
color
4
3.5
3
2.5
2
1.5
1
1.5
1
0.5
0
−0.5
−1
−1.5
PC 2
−2
−1
0
1
2
3
PC 1

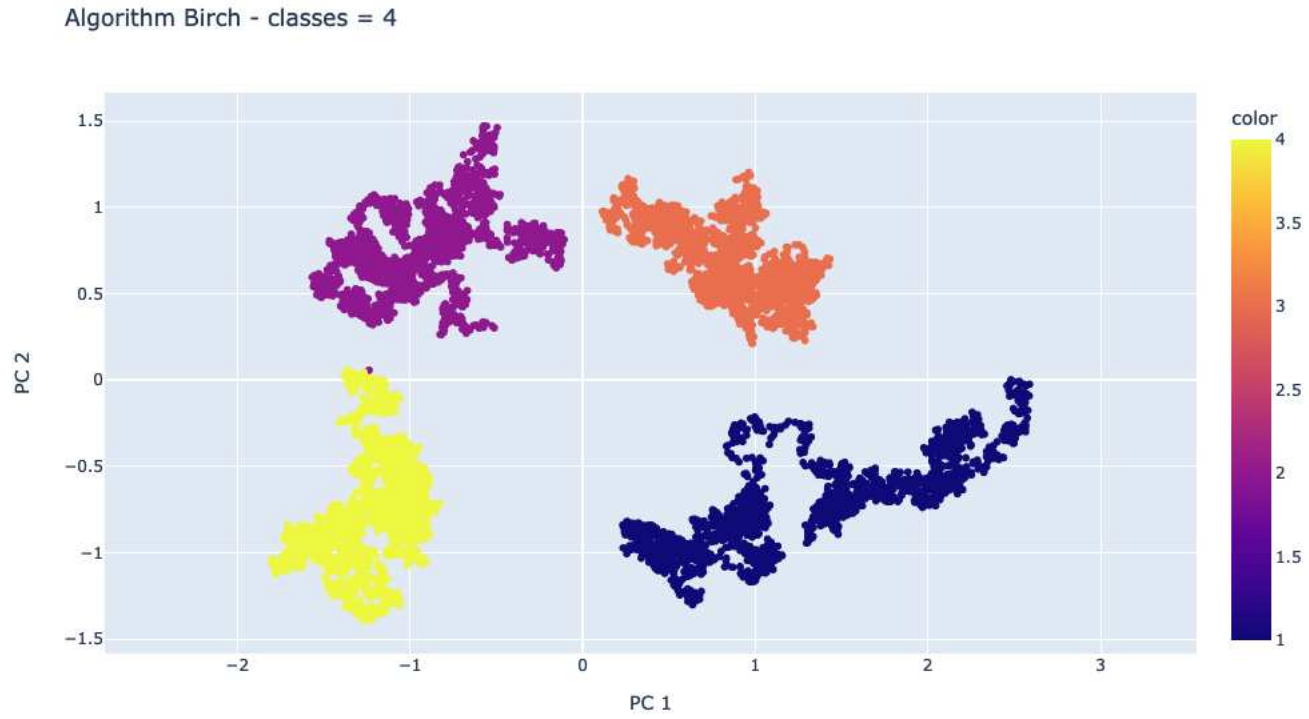

Algorithm Spectral Clustering - classes = 4
color
4
3.5
3
2.5
2
1.5
1
1.5
1
0.5
0
−0.5
−1
−1.5
PC 2
−2
−1
0
1
2
3
PC 1


Algorithm Agglomerative Clustering (Ward) - classes = 4
color
4
3.5
3
2.5
2
1.5
1
1.5
1
0.5
0
−0.5
−1
−1.5
PC 2
−2
−1
0
1
2
3
PC 1


Algorithm DBSCAN - classes = 1
color
1.4
1.2
1
0.8
0.6
1.5
1
0.5
0
−0.5
−1
−1.5
PC 2
−2
−1
0
1
2
3
PC 1


Algorithm Meanshift - classes = 2
color
2
1.8
1.6
1.4
1.2
1
PC 2
PC 1
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−1
0
1
2
3


Algorithm OPTICS - classes = 608
color
600
500
400
300
200
100
0
PC 2
PC 1
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−1
0
1
2
3


Algorithm Birch - classes = 4
color
4
3.5
3
2.5
2
1.5
1
PC 2
PC 1
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−1
0
1
2
3

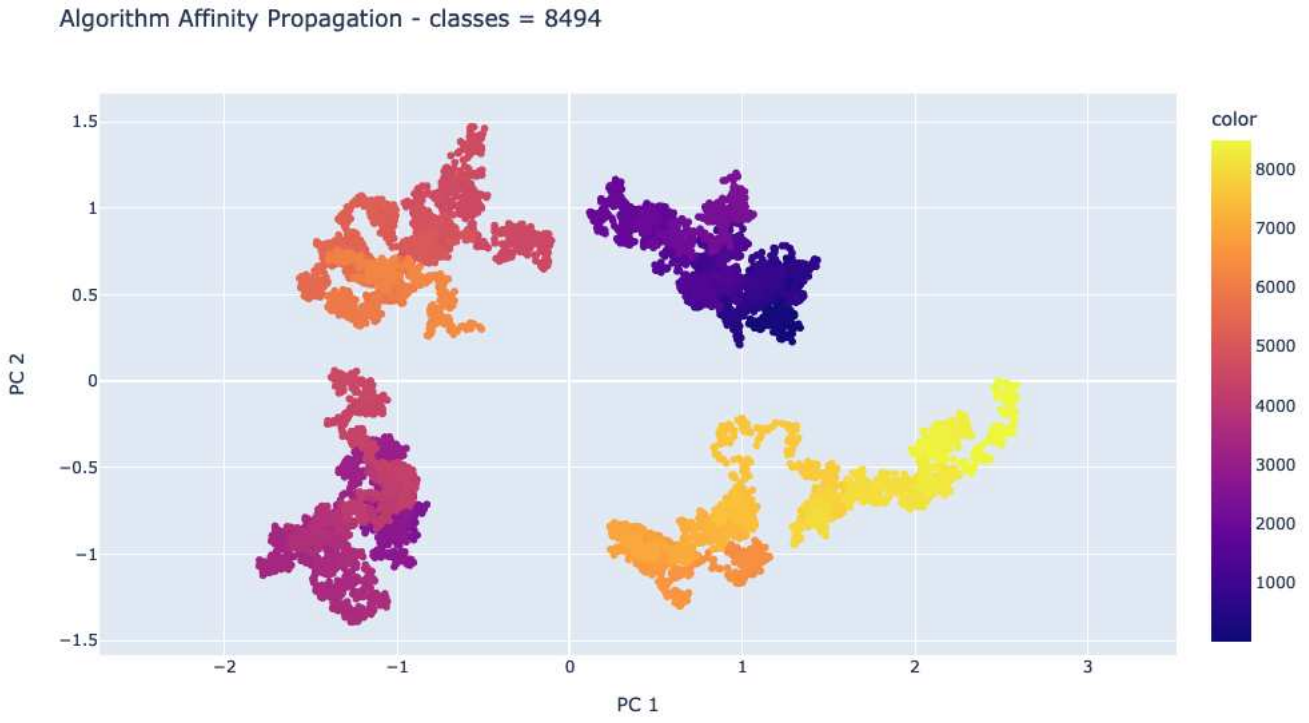
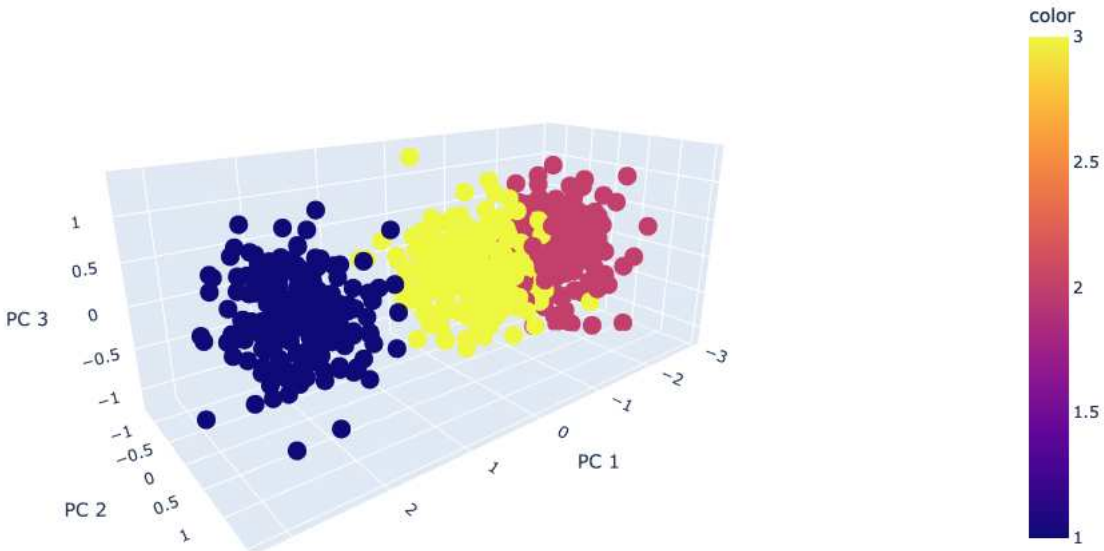

Algorithm Affinity Propagation - classes = 8494
color
8000
7000
6000
5000
4000
3000
2000
1000
PC 2
PC 1
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−1
0
1
2
3

Tabella 3.18: aggregation

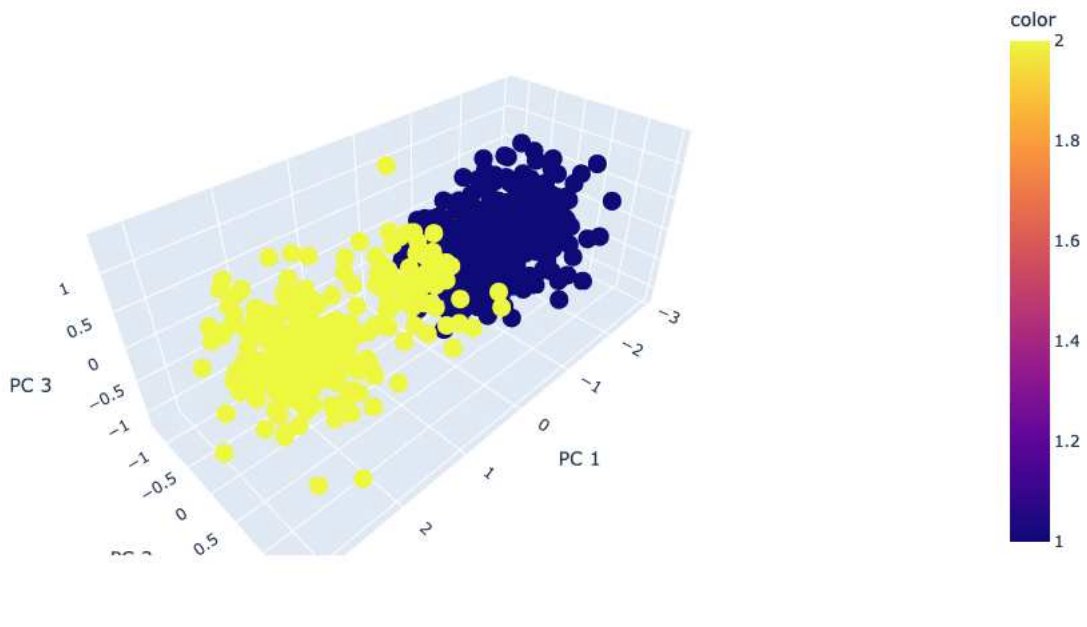
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.499 | 0.666 | 9.419 |
| KMeans | 0.999 | 0.999 | 0.666 |
| Spectral C. | 0.987 | 0.982 | 25.194 |
| Ward | 1,0 | 1.0 | 2.69 |
| DBscan | 1.0 | 1.0 | 0.342 |
| MeanShift | 0.499 | 0.666 | 58.998 |
| Optics | 0.002 | 0.232 | 9.174 |
| Birch | 0.999 | 0.999 | 0.149 |
| Affinity Prop. | 0.0042 | 0.0524 | 345.30 |

CircleClustering > DBSCAN, Meanshift, OPTICS, Affinity Propagation CircleClustering < KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Birch

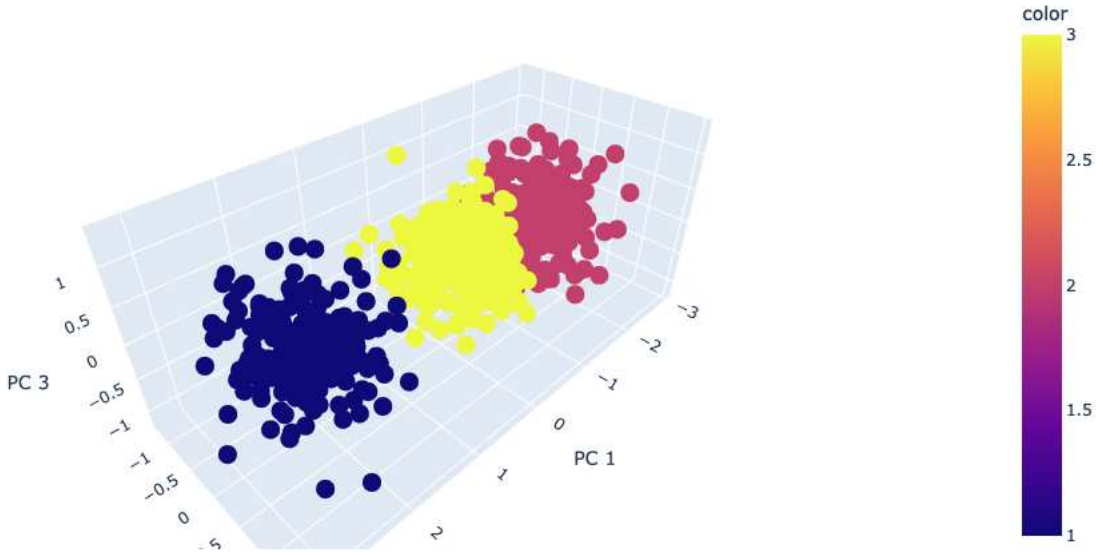
- Dataset: mk3
  Samples = 600, Features = 3, Classes = 3



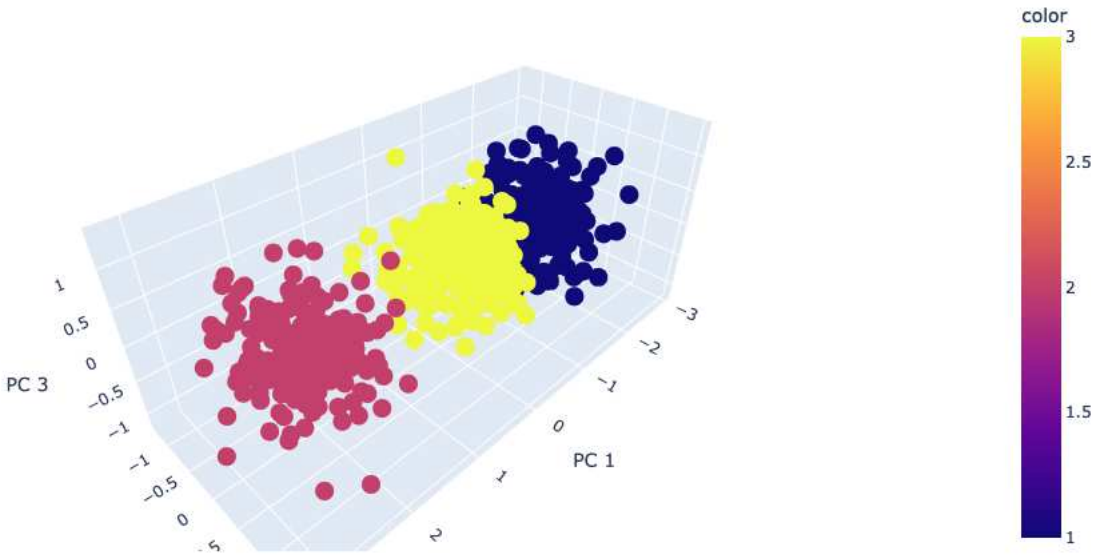
Algorithm CircleClustering - classes = 2



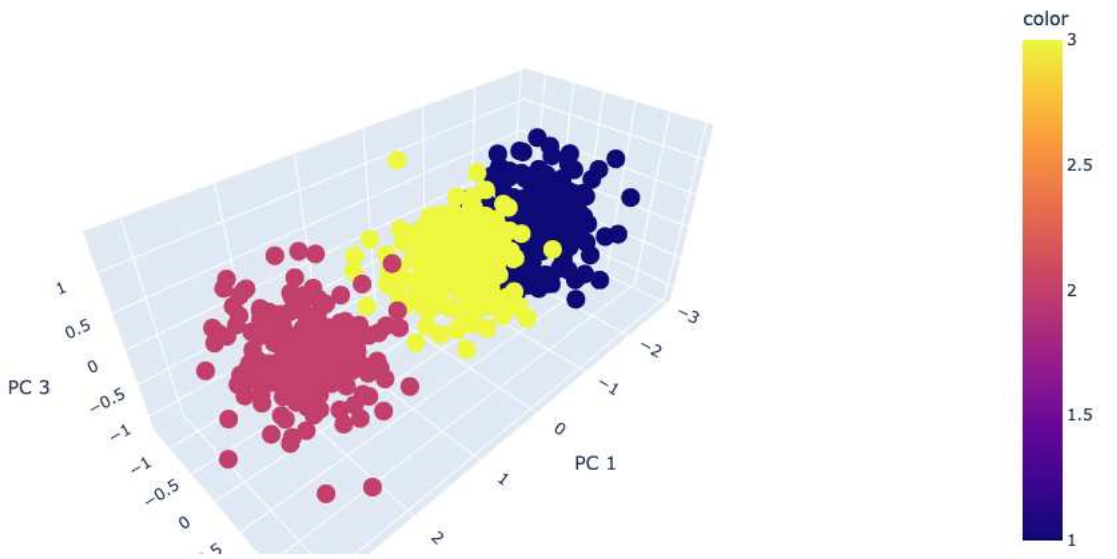

Algorithm KMeans - classes = 3



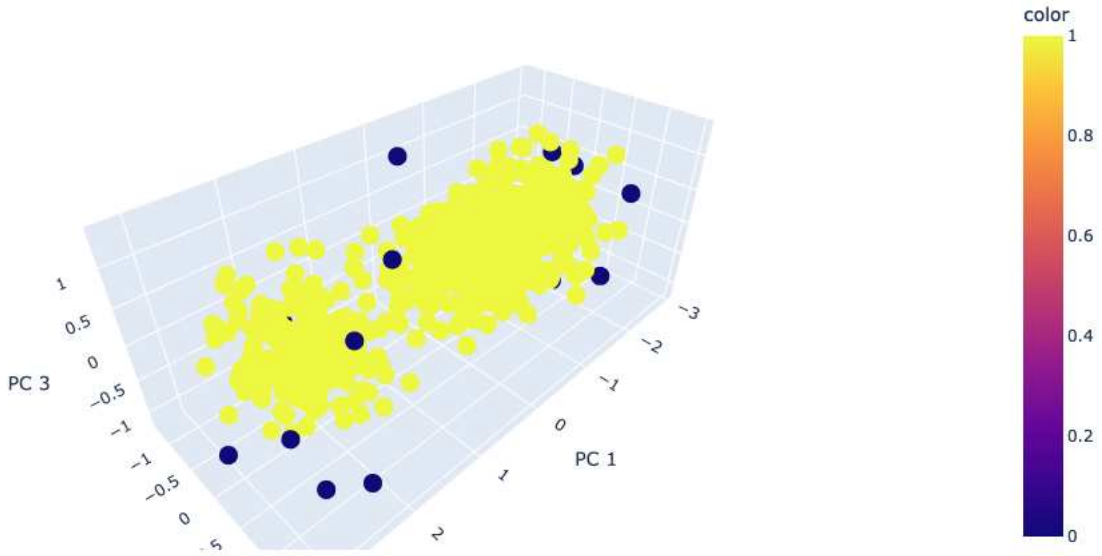
Algorithm Spectral Clustering - classes = 3



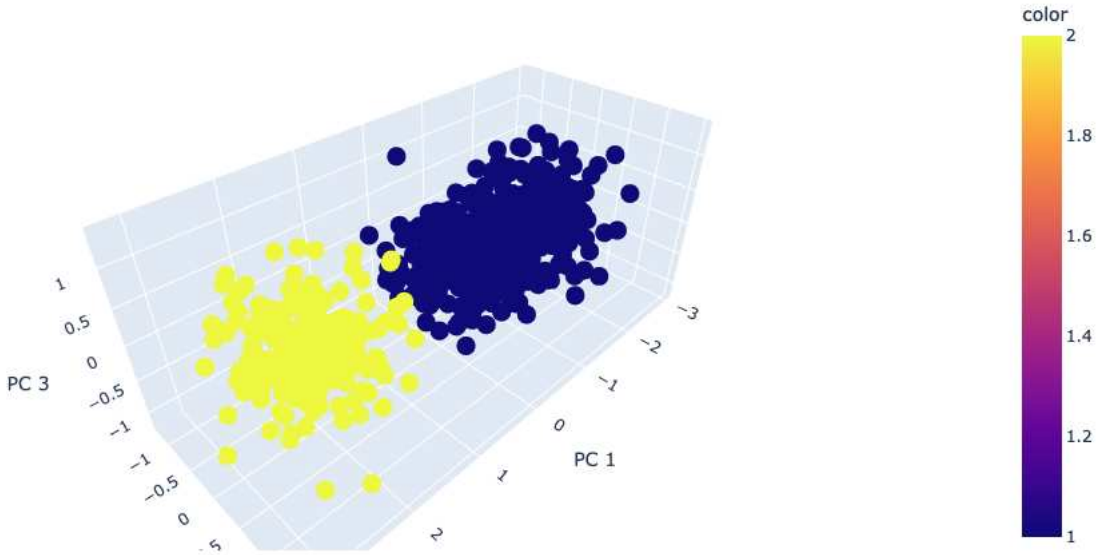

Algorithm Agglomerative Clustering (Ward) - classes = 3



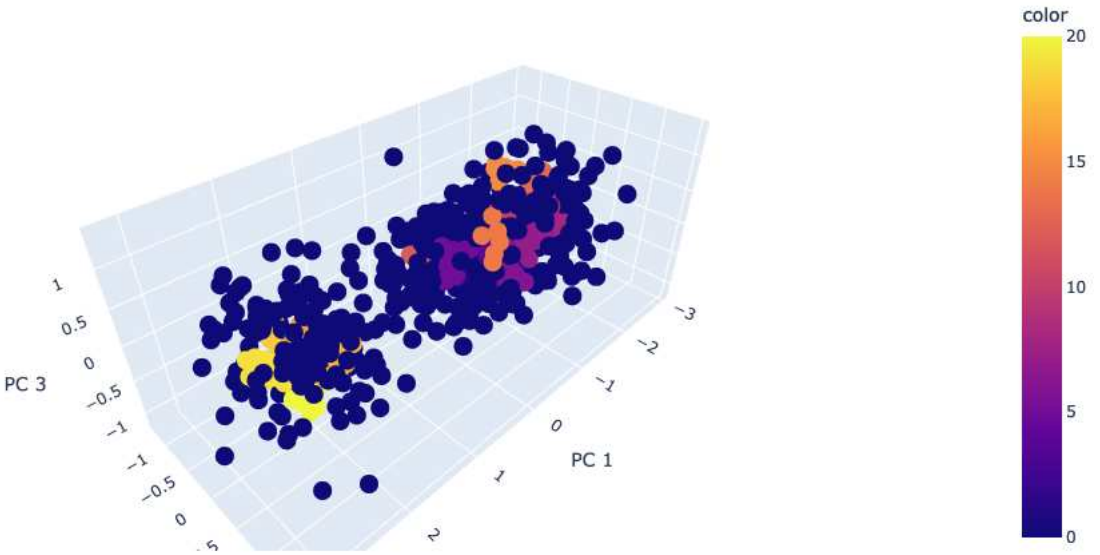
Algorithm DBSCAN - classes = 1



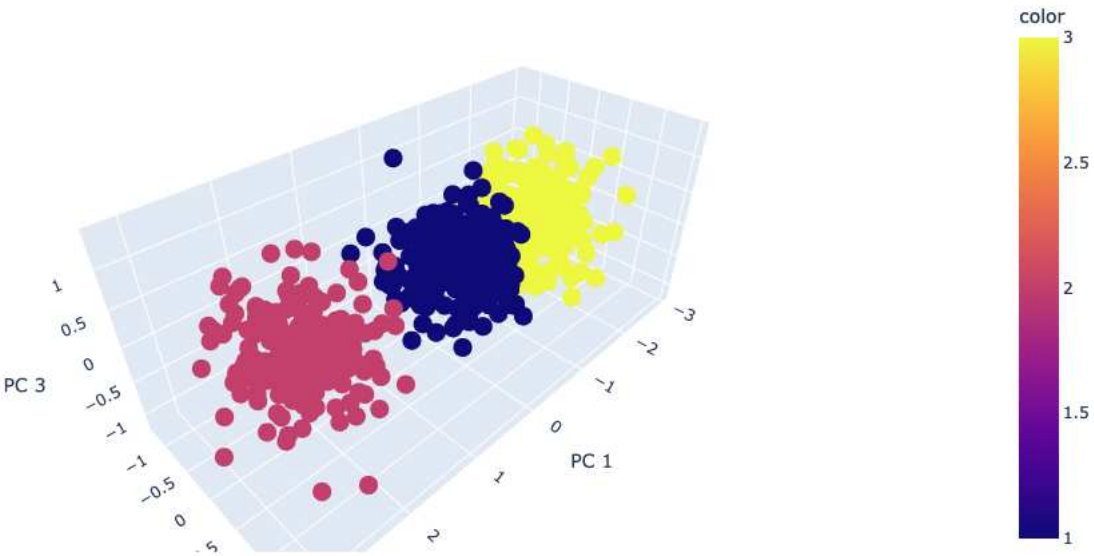

Algorithm Meanshift - classes = 2



Algorithm OPTICS - classes = 20




Algorithm Birch - classes = 3



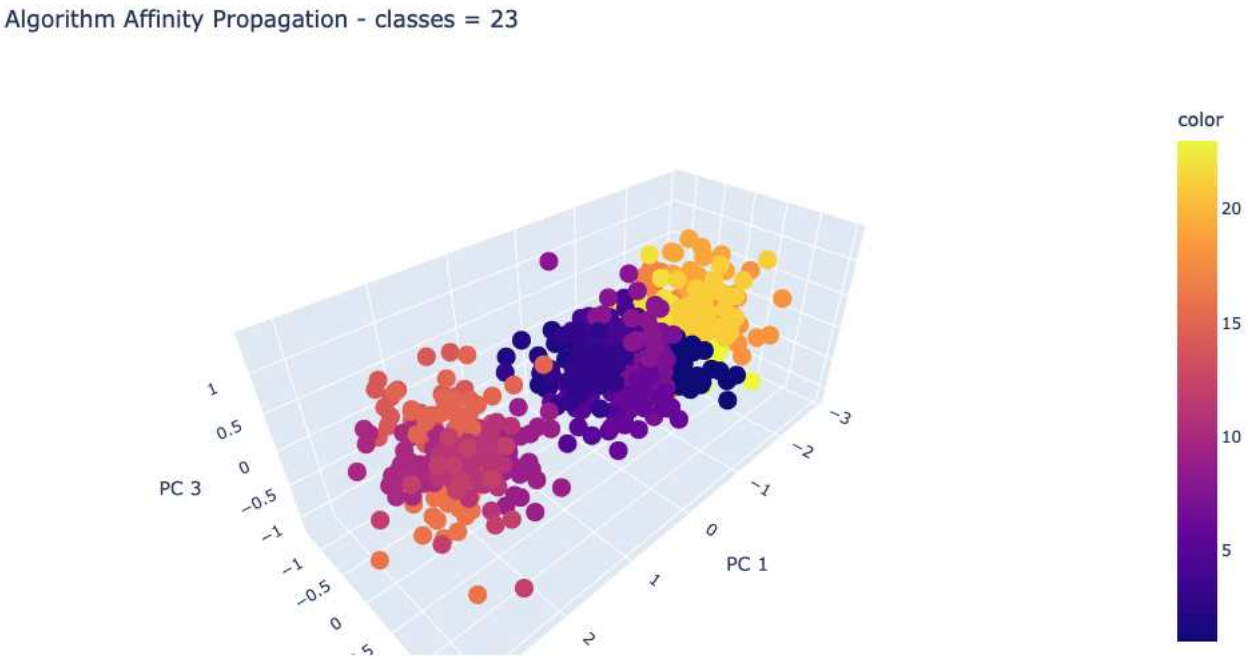


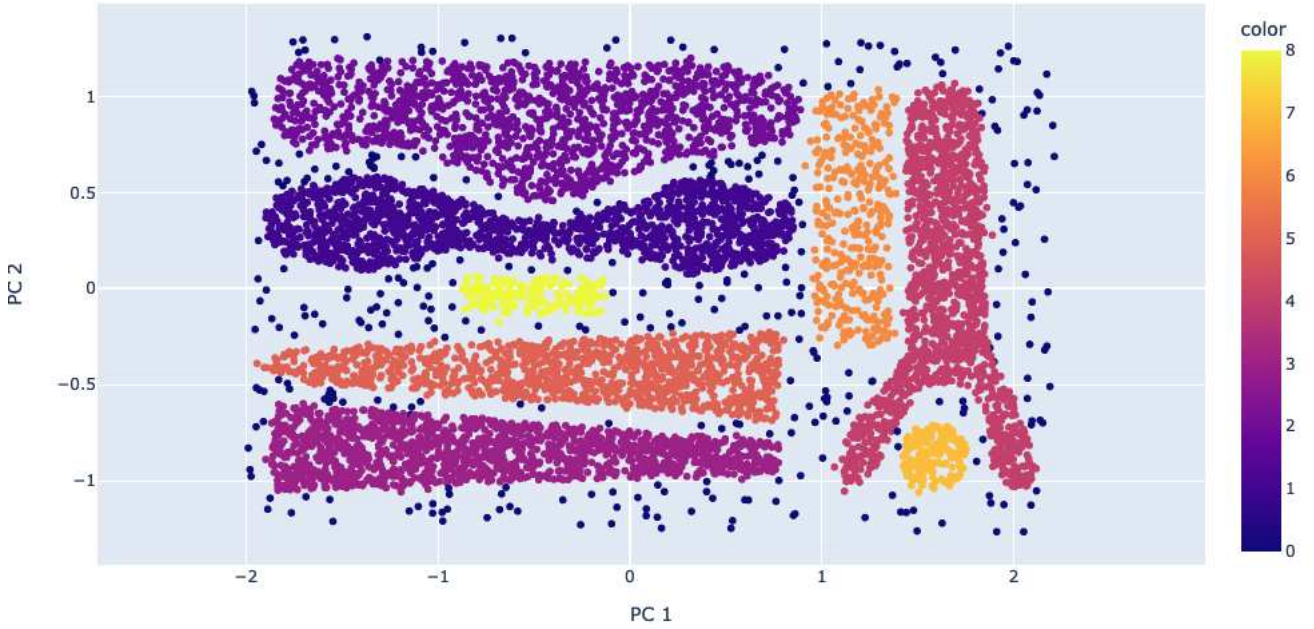

Tabella 3.19: aggregation

| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.4642 | 0.5383 | 0.306 |
| KMeans | 0.891 | 0.862 | 0.034 |
| Spectral C. | 0.873 | 0.846 | 0.277 |
| Ward | 0.864 | 0.837 | 0.006 |
| DBscan | 0.0002 | 0.004 | 0.003 |
| MeanShift | 0.558 | 0.692 | 3.258 |
| Optics | 0.007 | 0.184 | 0.362 |
| Birch | 0.869 | 0.844 | 0.011 |
| Affinity Prop. | 0.157 | 0.455 | 0.269 |

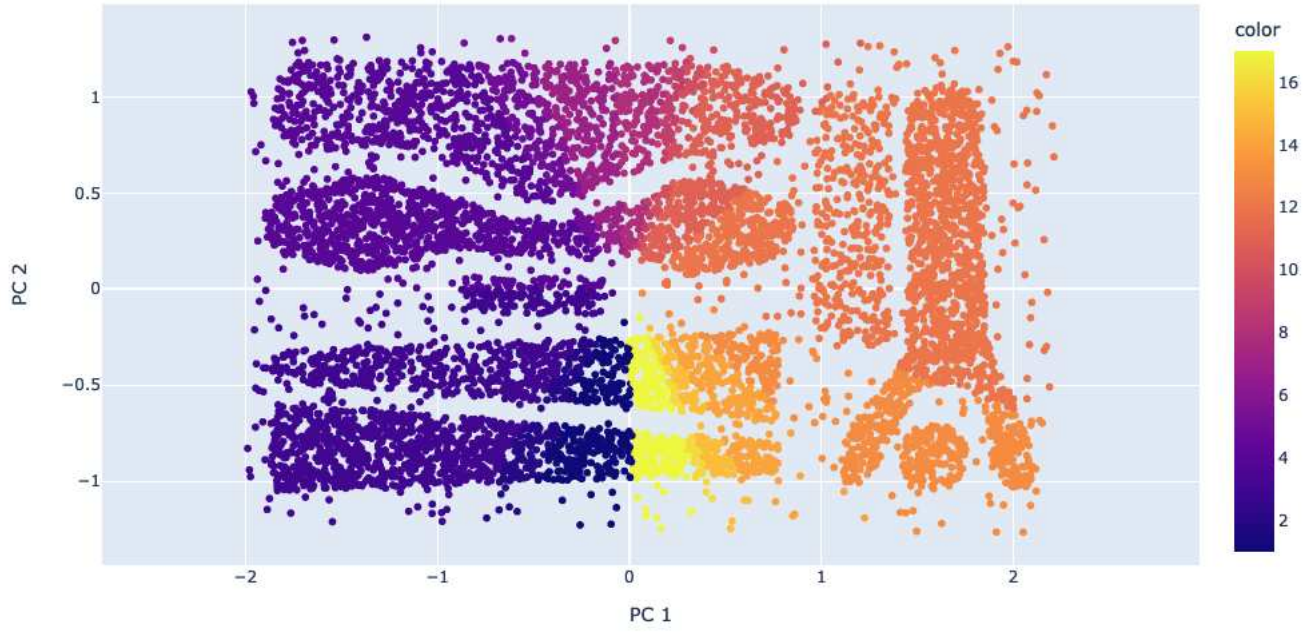
CircleClustering > DBSCAN, OPTICS, Affinity Propagation CircleClustering < KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Meanshift, Birch

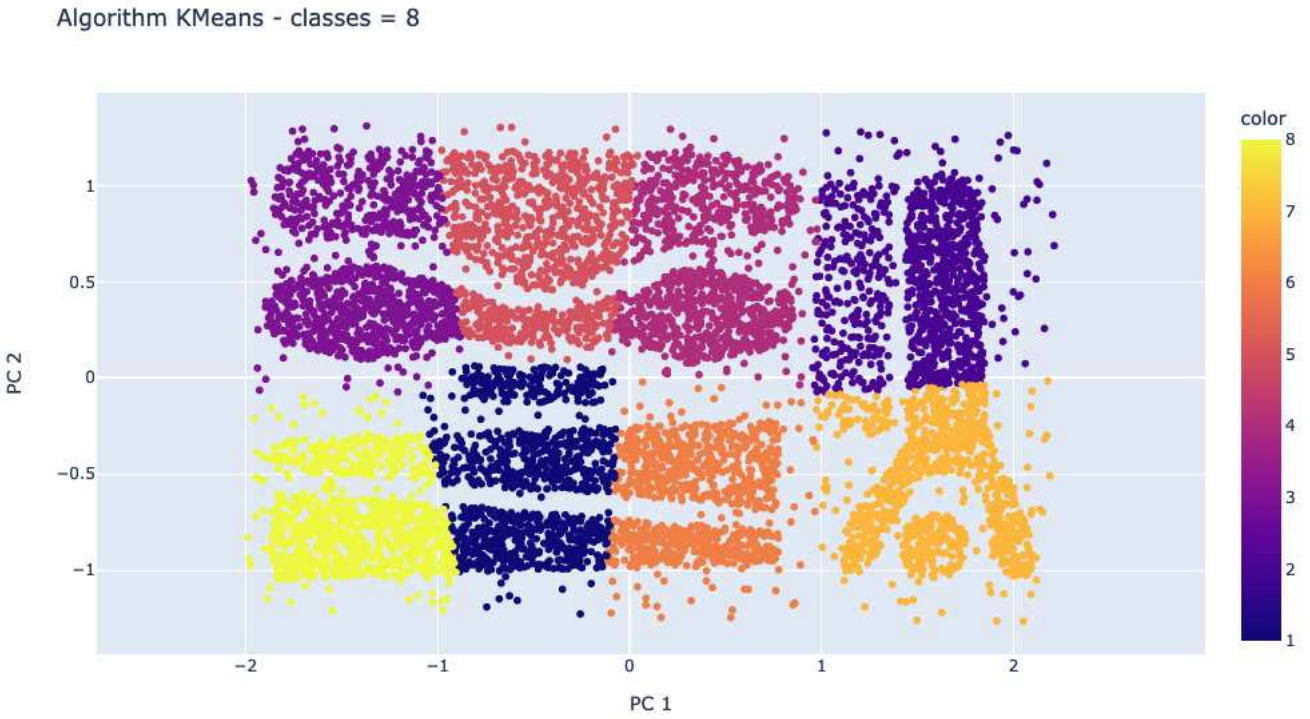
**Batteria: other**

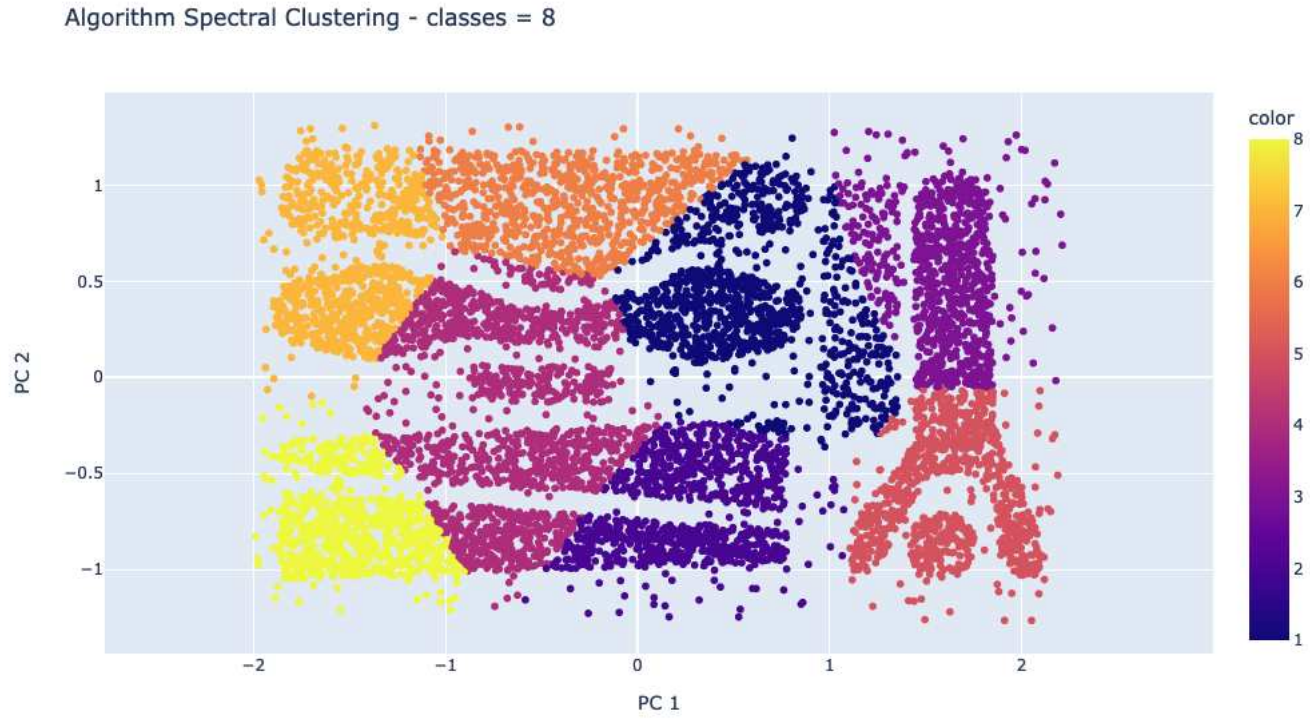
- Dataset: chameleon_t8_8k
  Samples = 8000, Features = 2, Classes = 8

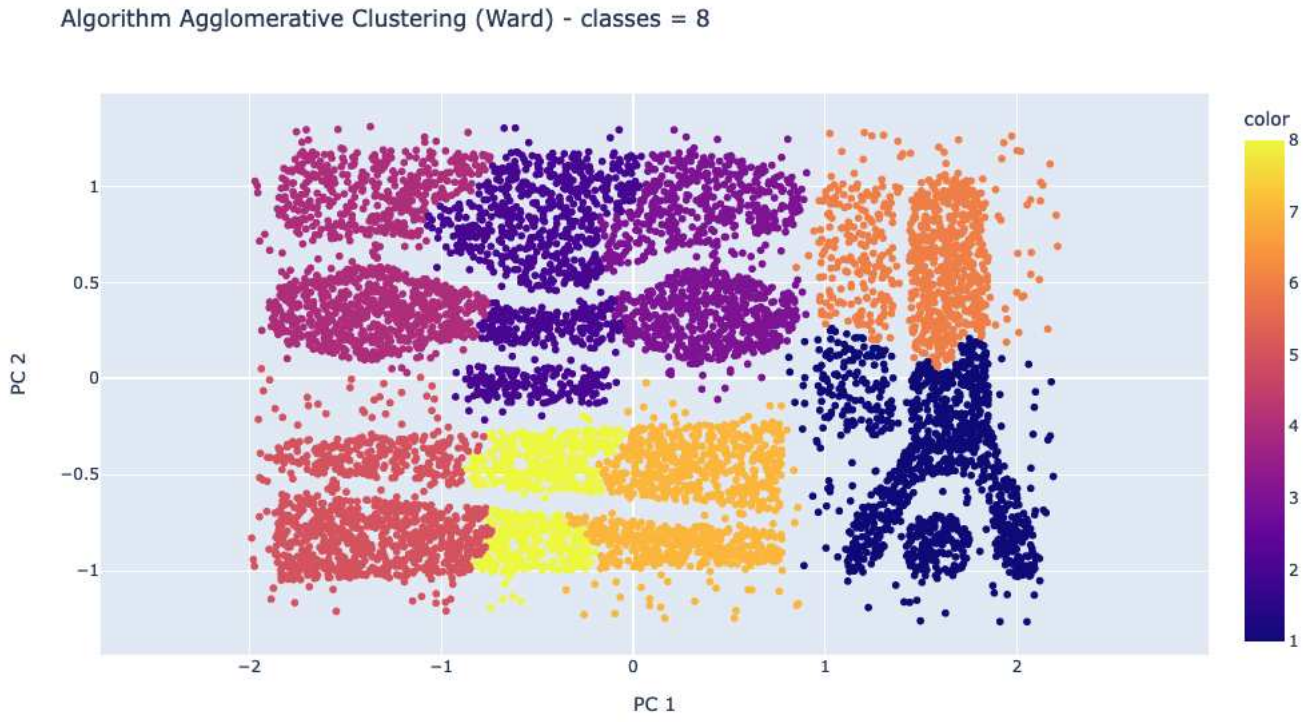


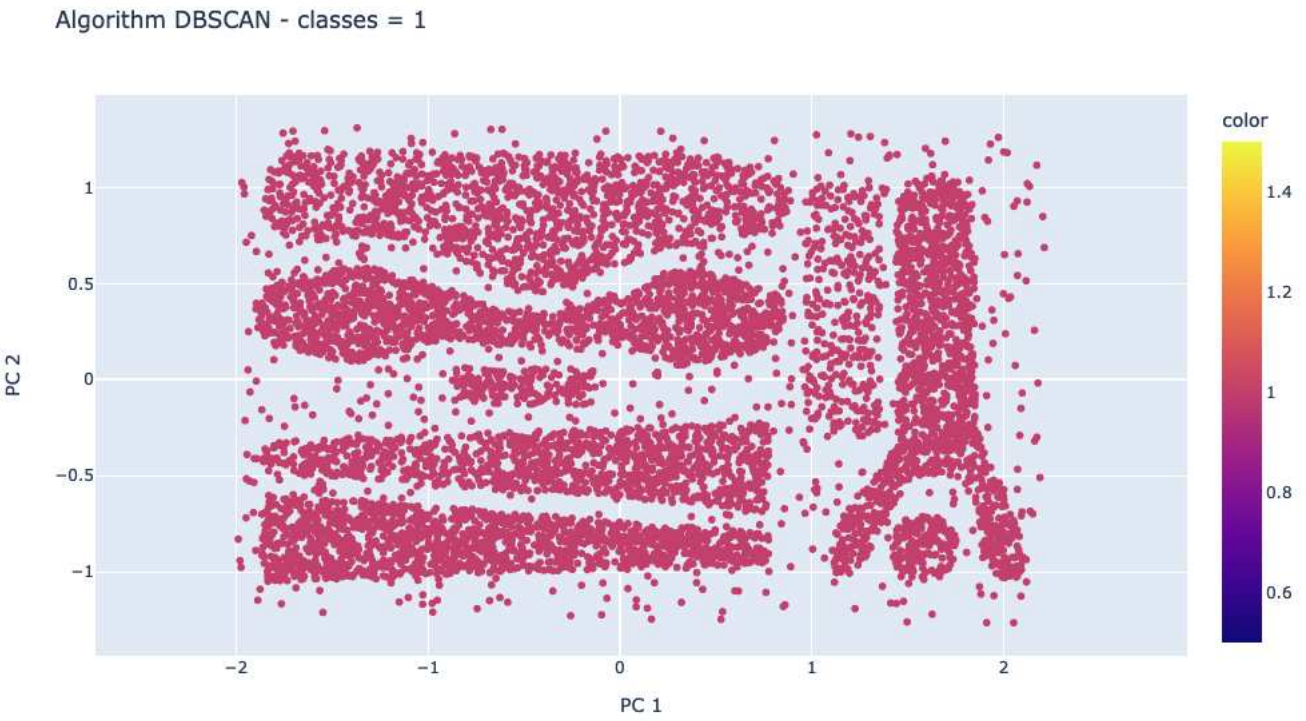

Algorithm CircleClustering - classes = 17

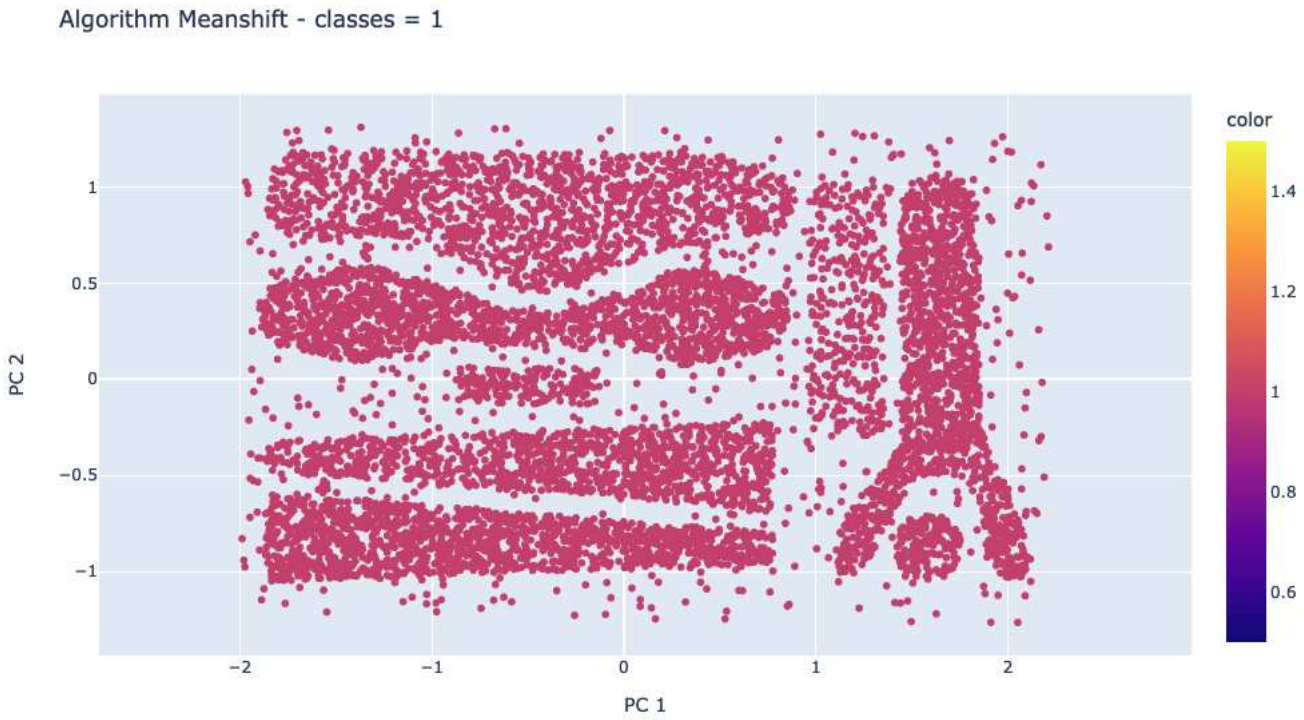


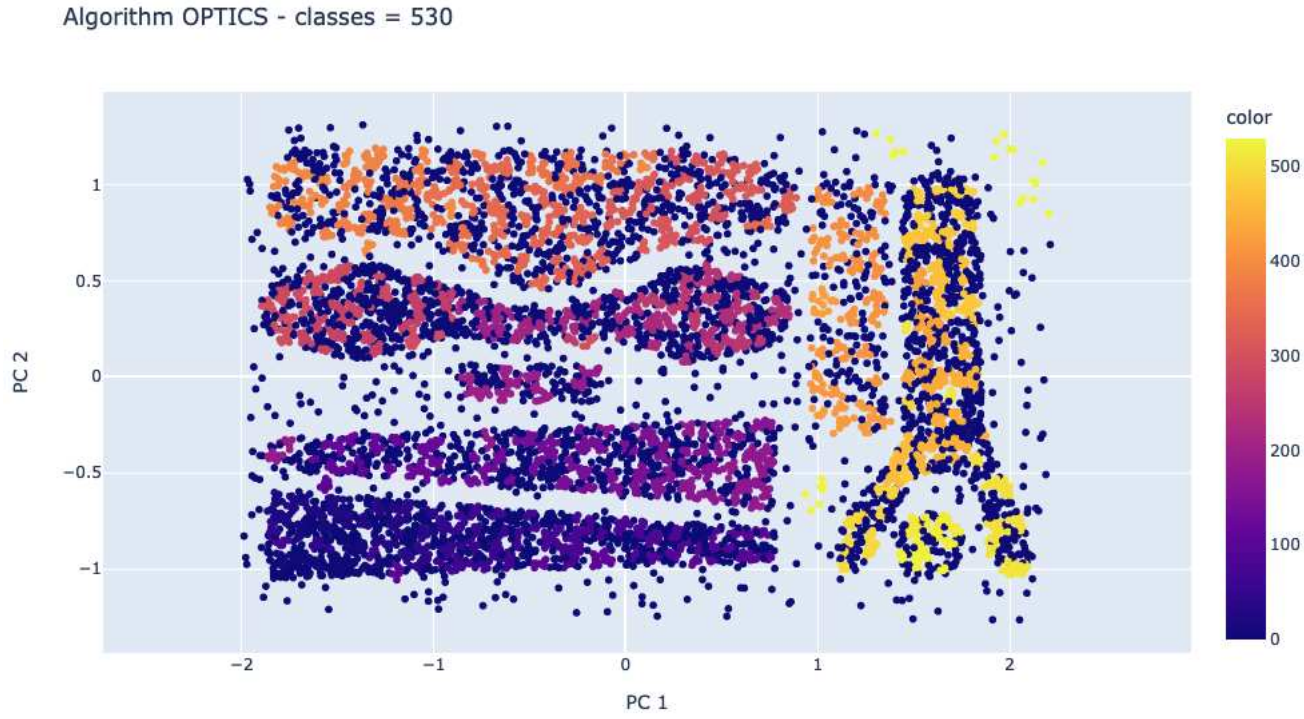

Algorithm KMeans - classes = 8
color
8
7
6
5
4
3
2
1
PC 2
1
0.5
0
−0.5
−1
−2
−1
0
1
2
PC 1

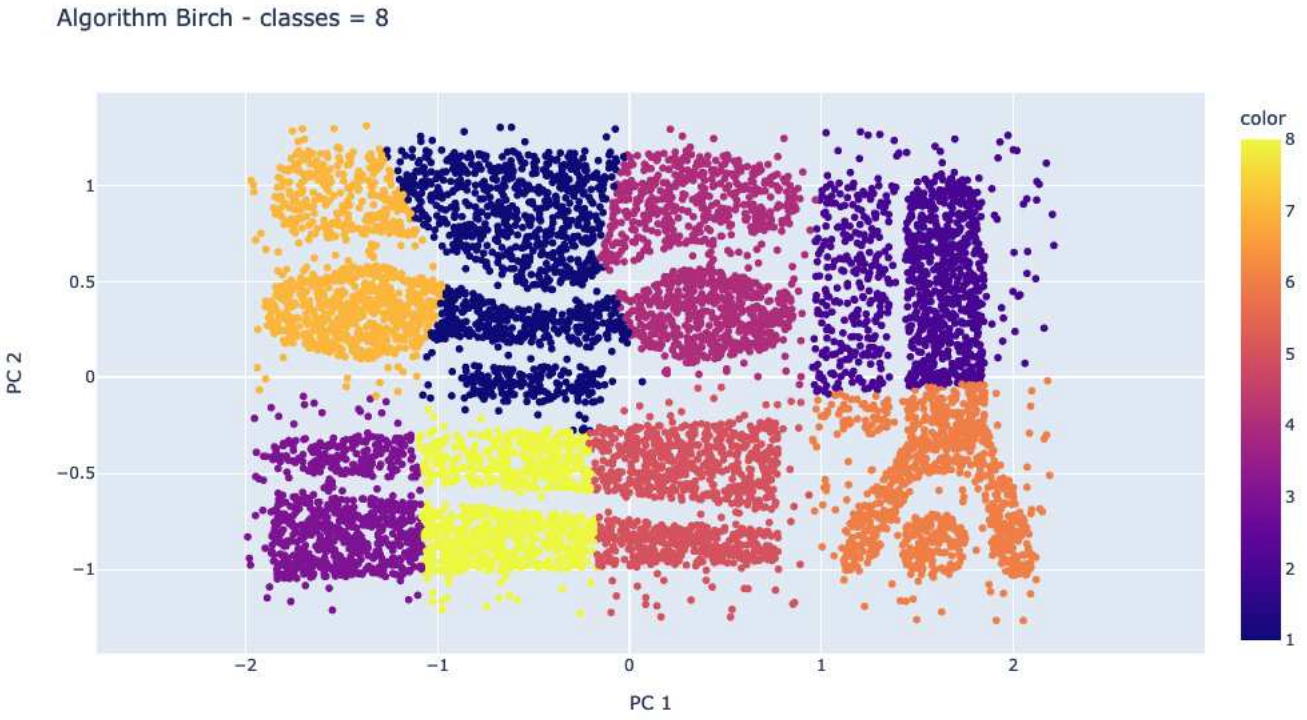

Algorithm Spectral Clustering - classes = 8
color
8
7
6
5
4
3
2
1
PC 2
1
0.5
0
−0.5
−1
−2
−1
0
1
2
PC 1


Algorithm Agglomerative Clustering (Ward) - classes = 8
color
8
7
6
5
4
3
2
1
PC 2
1
0.5
0
−0.5
−1
−2
−1
0
1
2
PC 1


Algorithm DBSCAN - classes = 1
color
1.4
1.2
1
0.8
0.6
PC 2
1
0.5
0
−0.5
−1
−2
−1
0
1
2
PC 1


Algorithm Meanshift - classes = 1
color
1.4
1.2
1
0.8
0.6
PC 2
PC 1
1
0.5
0
−0.5
−1
−2
−1
0
1
2


Algorithm OPTICS - classes = 530
color
500
400
300
200
100
0
PC 2
PC 1
1
0.5
0
−0.5
−1
−2
−1
0
1
2


Algorithm Birch - classes = 8
color
8
7
6
5
4
3
2
1
PC 2
1
0.5
0
−0.5
−1
−2
−1
0
1
2
PC 1

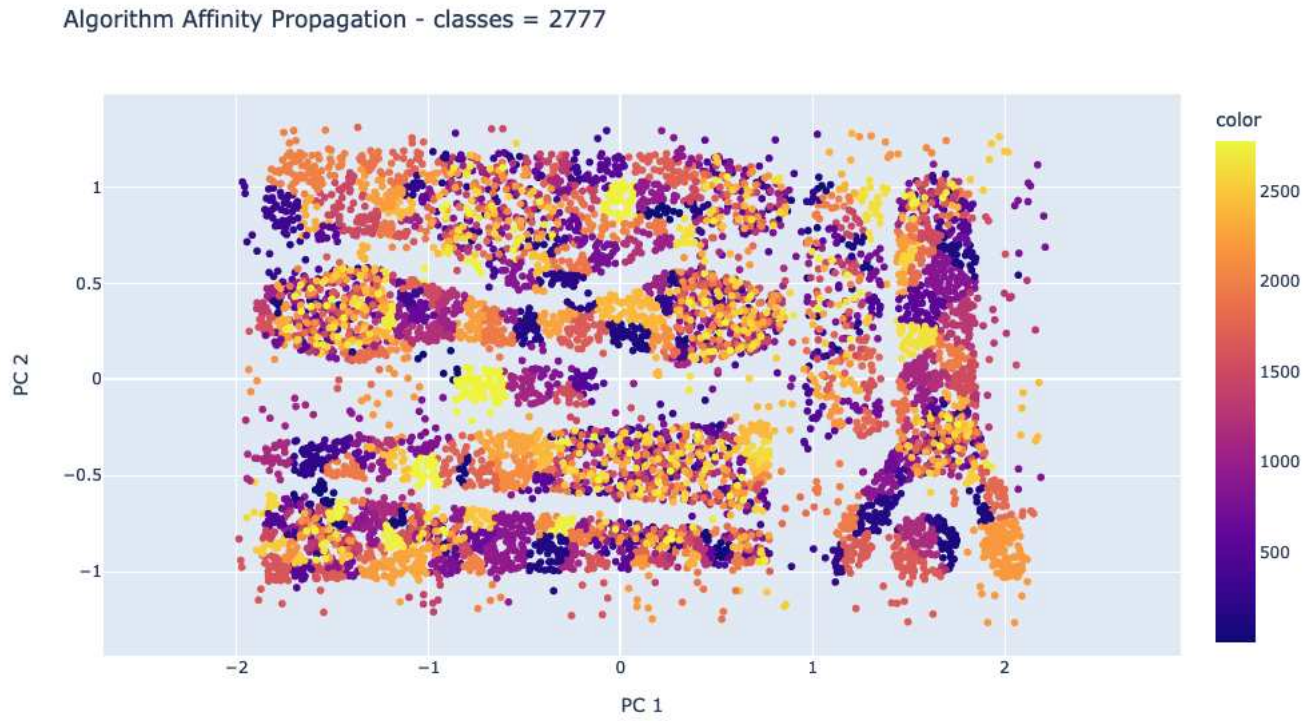

Algorithm Affinity Propagation - classes = 2777
color
2500
2000
1500
1000
500
PC 2
1
0.5
0
−0.5
−1
−2
−1
0
1
2
PC 1

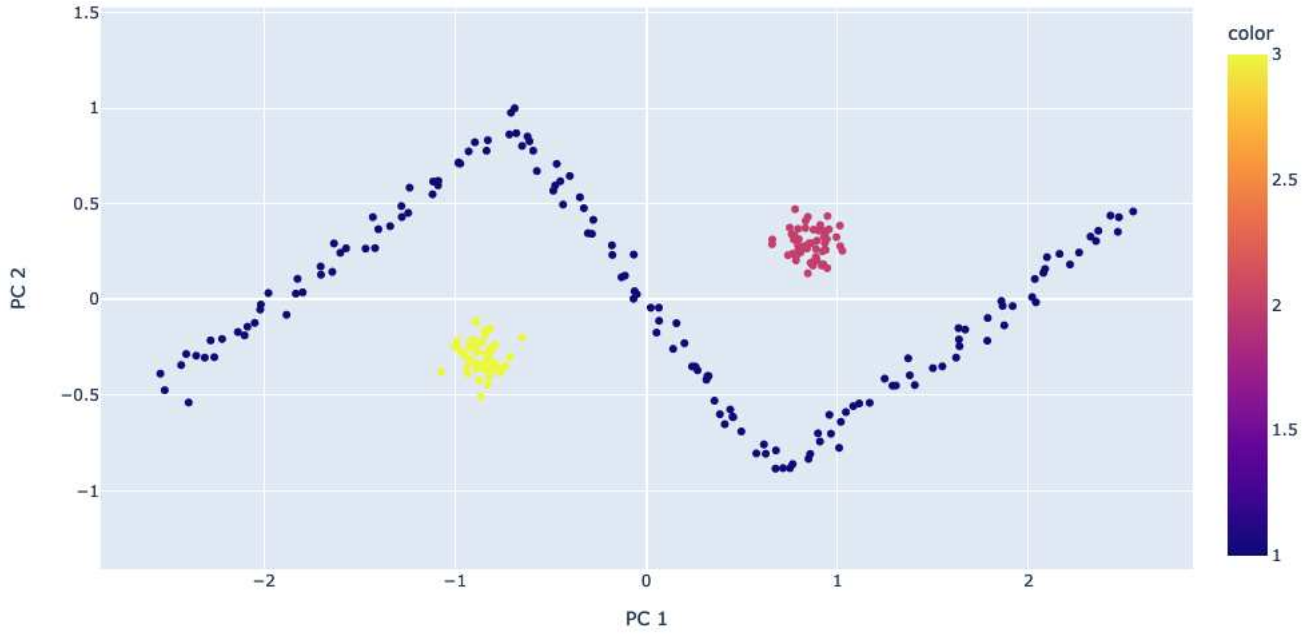
Tabella 3.20: chameleon_t8_8k

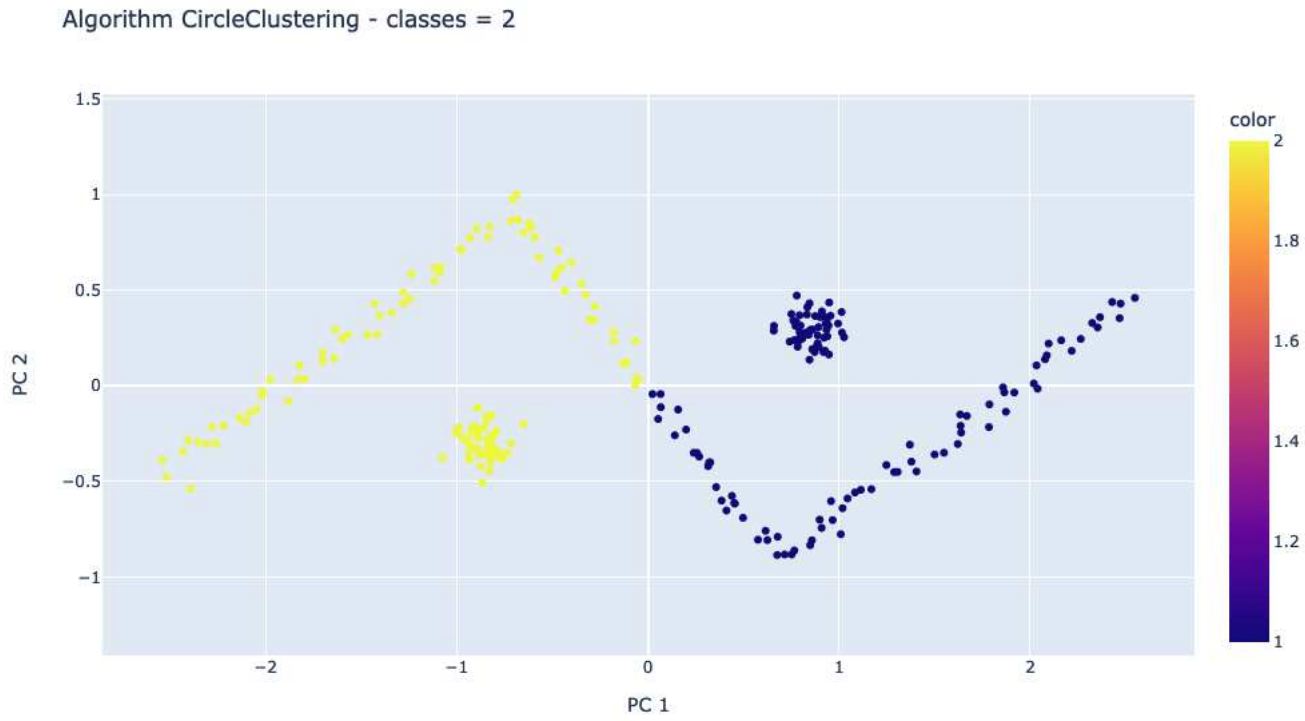
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.3043 | 0.4767 | 2.934 |
| KMeans | 0.347 | 0.551 | 1.095 |
| Spectral C. | 0.332 | 0.528 | 14.535 |
| Ward | 0.350 | 0.549 | 1.745 |
| DBscan | 0.0 | 0.0 | 0.178 |
| MeanShift | 0.0 | 0.0 | 173.51 |
| Optics | -0.0034 | 0.284 | 6.928 |
| Birch | 0.338 | 0.546 | 0.130 |
| Affinity Prop. | 0.025 | 0.302 | 217.01 |

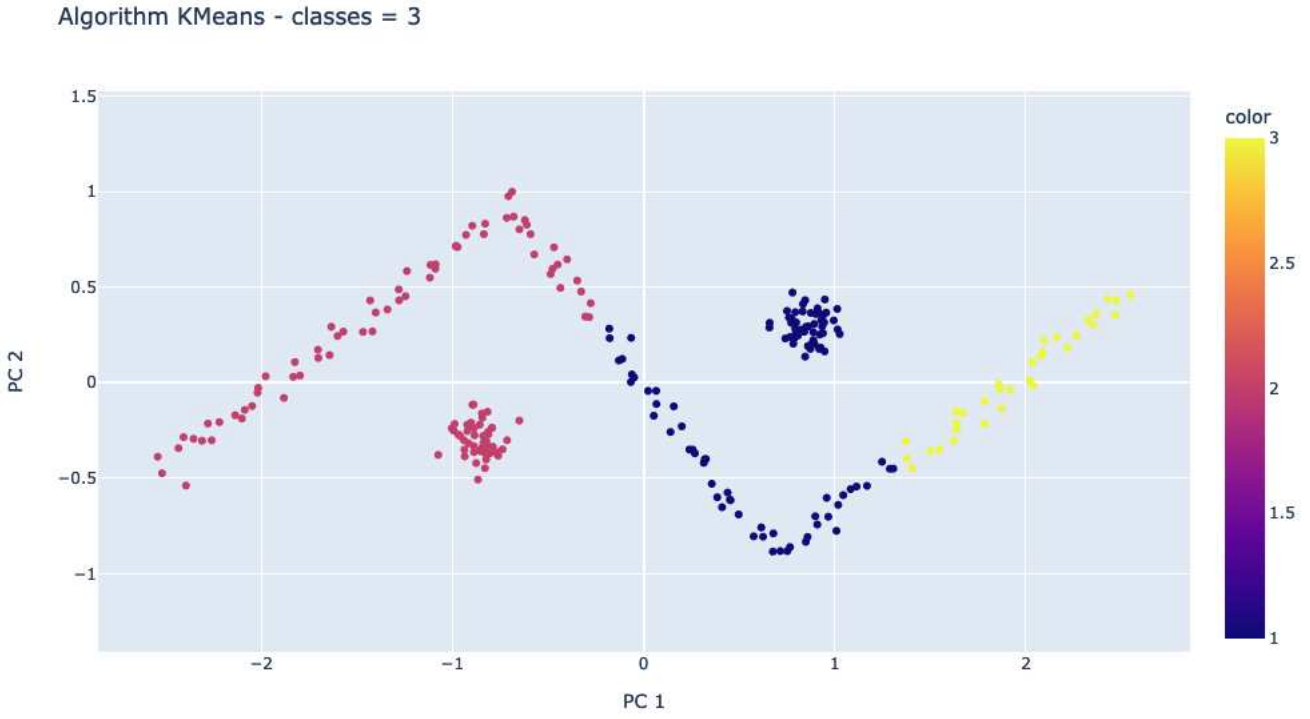
CircleClustering > DBSCAN, Meanshift, OPTICS, Affinity Propagation CircleClustering < KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Birch

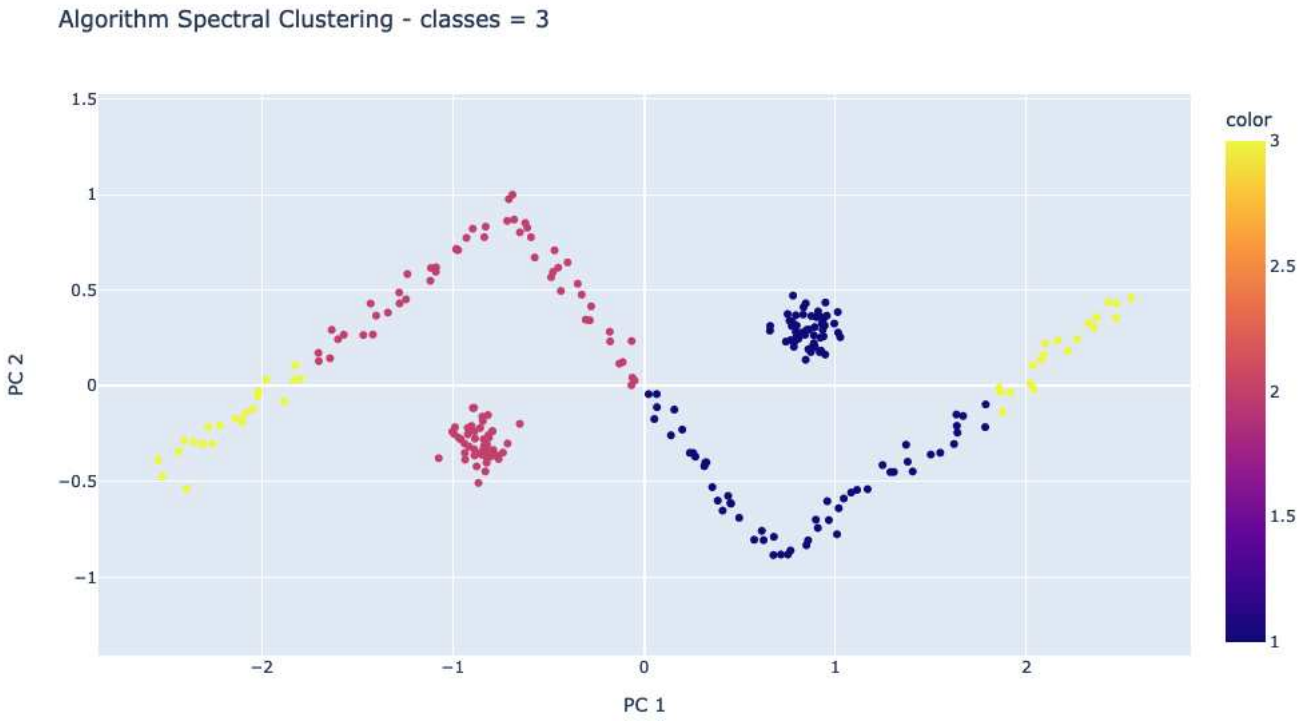
**Batteria: graves**

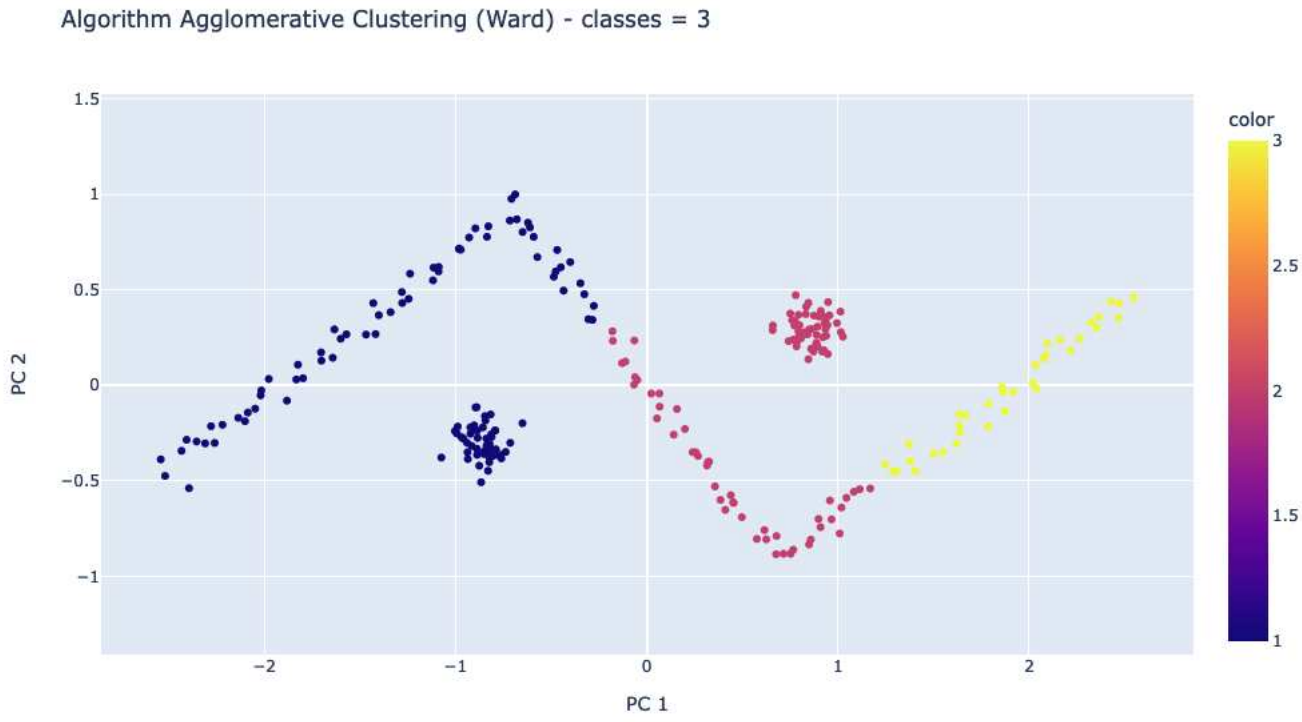
- Dataset: zigzag_t8_8k
  Samples = 250, Features = 2, Classes = 3

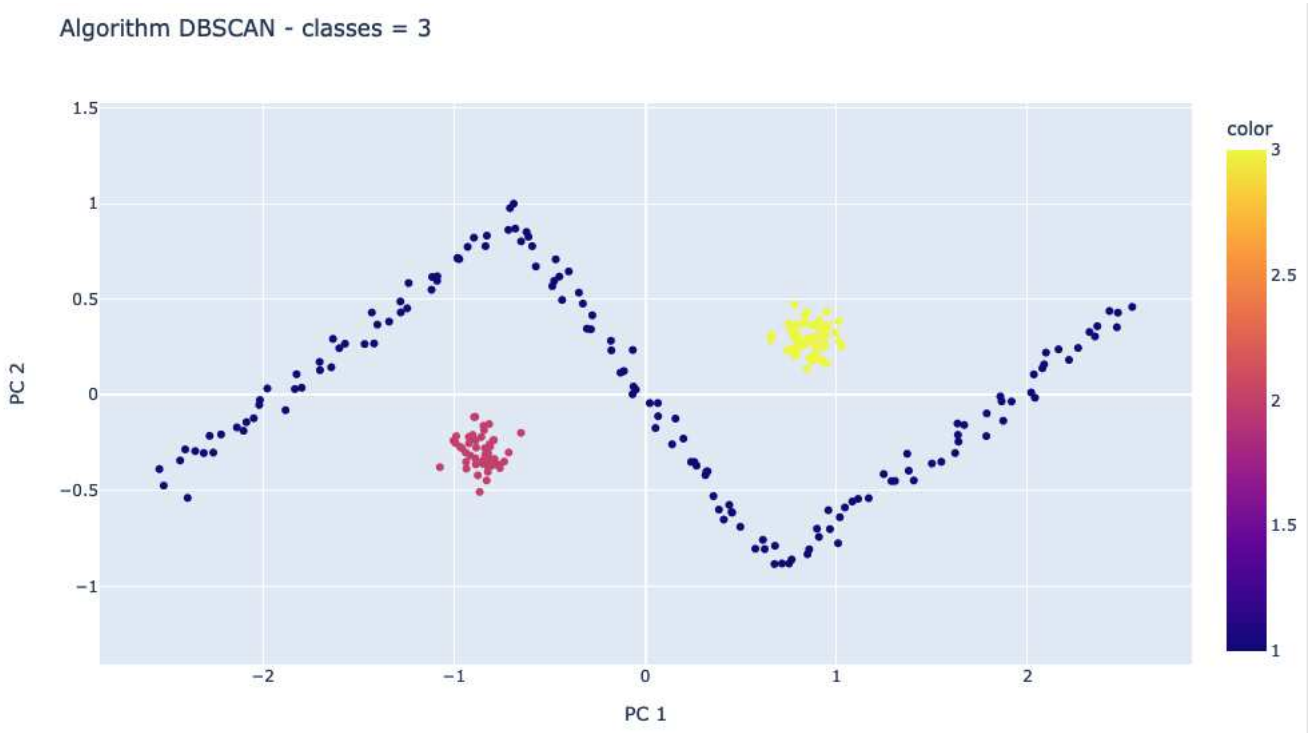


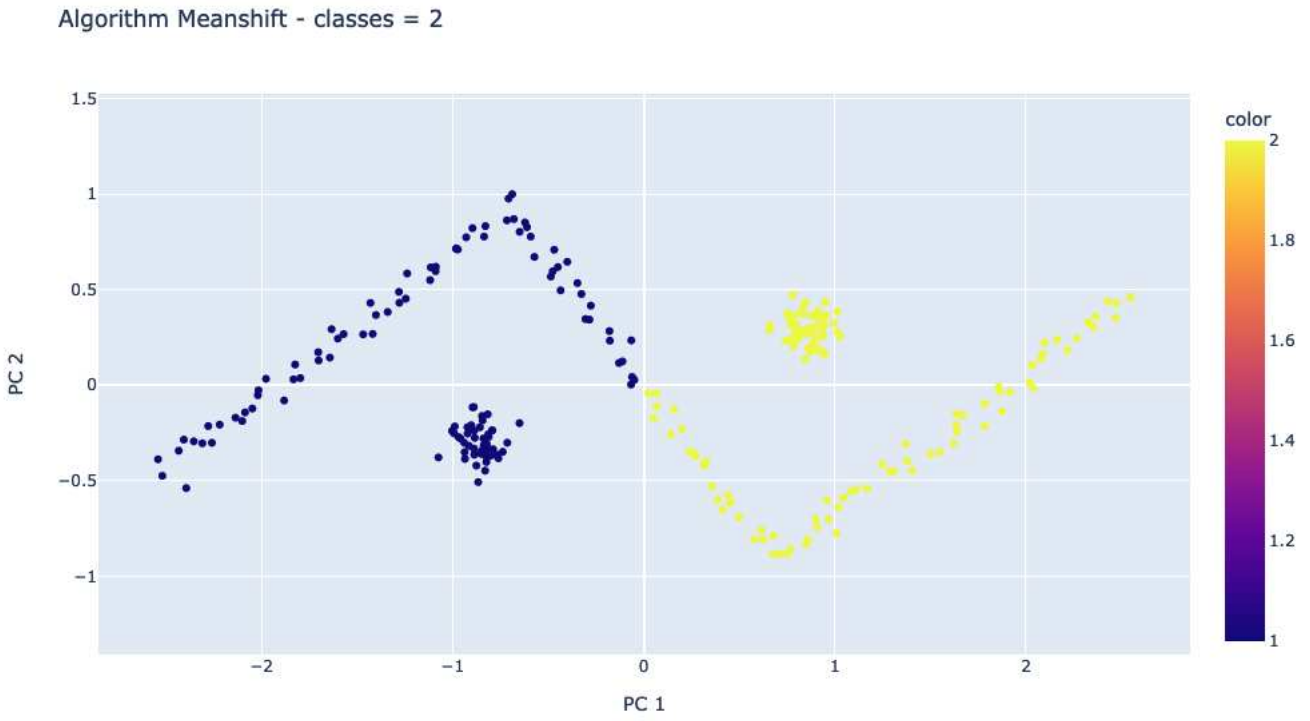

Algorithm CircleClustering - classes = 2
color
2
1.8
1.6
1.4
1.2
1
1.5
1
0.5
0
−0.5
−1
PC 2
−2
−1
0
1
2
PC 1

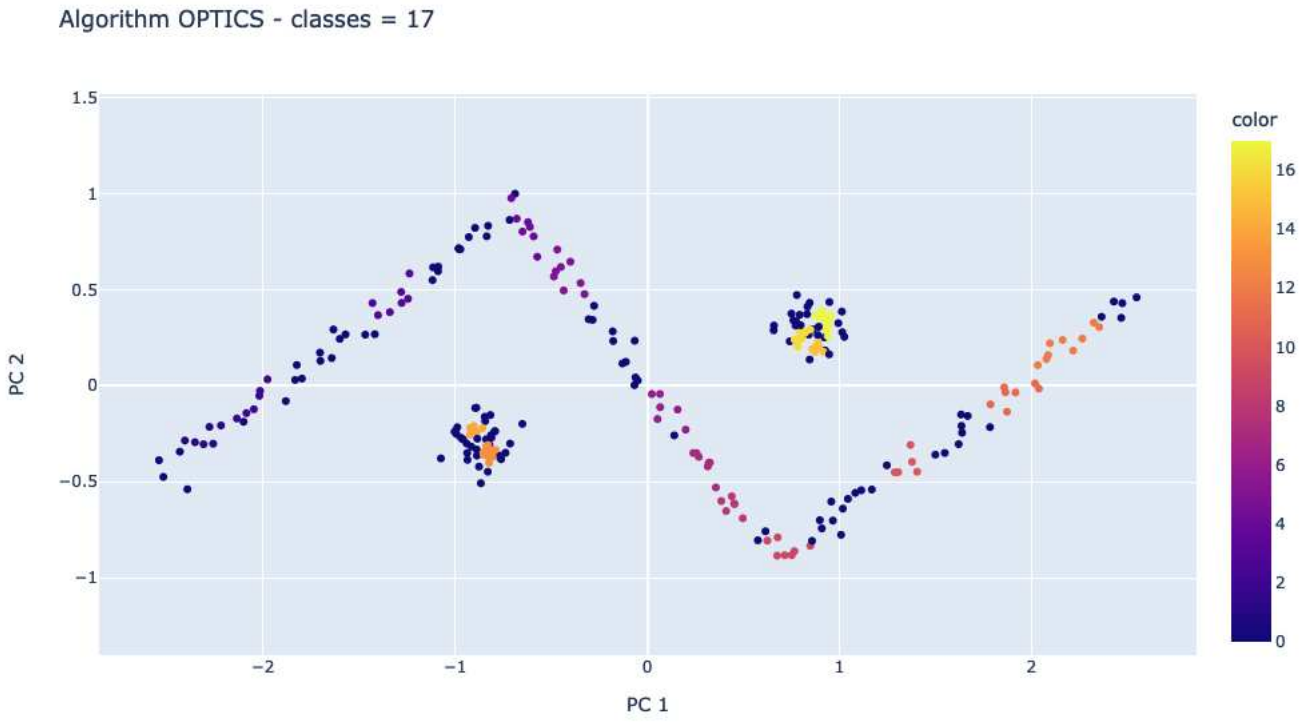

Algorithm KMeans - classes = 3
color
3
2.5
2
1.5
1
1.5
1
0.5
0
−0.5
−1
PC 2
−2
−1
0
1
2
PC 1

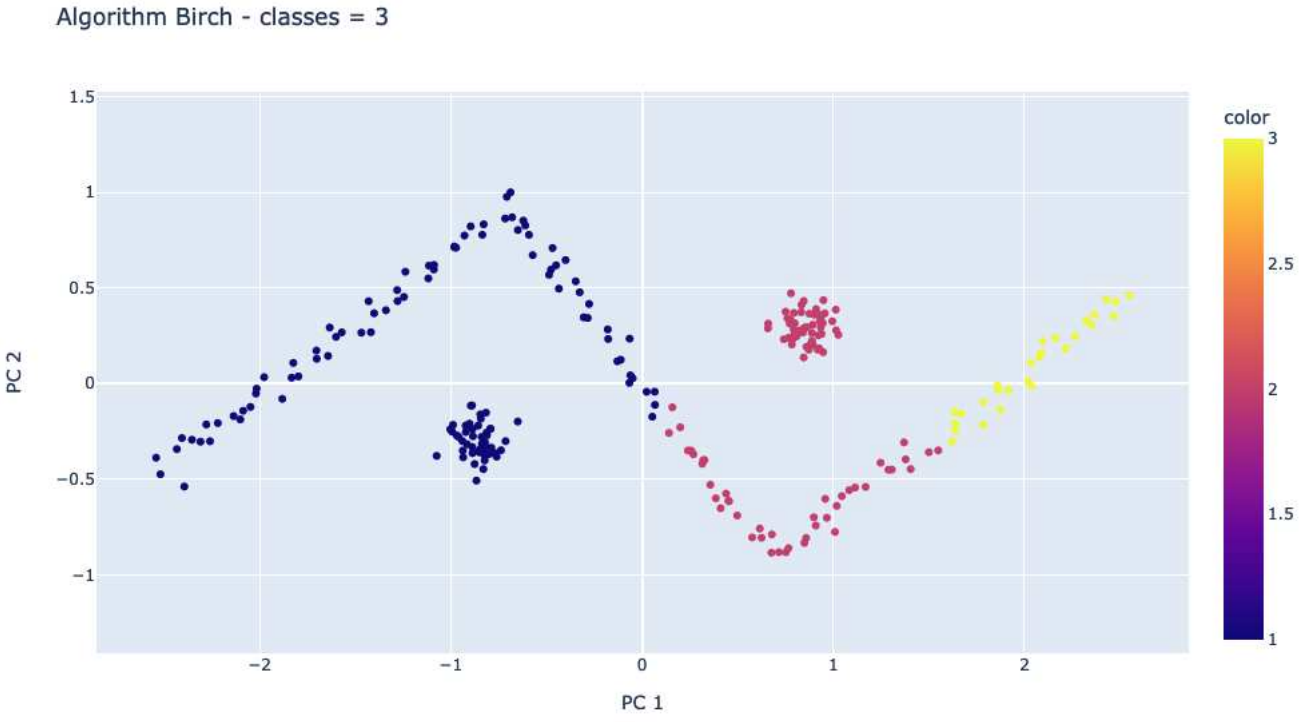

Algorithm Spectral Clustering - classes = 3
color
3
2.5
2
1.5
1
PC 2
PC 1
1.5
1
0.5
0
−0.5
−1
−2
−1
0
1
2


Algorithm Agglomerative Clustering (Ward) - classes = 3
color
3
2.5
2
1.5
1
PC 2
PC 1
1.5
1
0.5
0
−0.5
−1
−2
−1
0
1
2


Algorithm DBSCAN - classes = 3
color
3
2.5
2
1.5
1
1.5
1
0.5
0
−0.5
−1
PC 2
−2
−1
0
1
2
PC 1


Algorithm Meanshift - classes = 2
color
2
1.8
1.6
1.4
1.2
1
1.5
1
0.5
0
−0.5
−1
PC 2
−2
−1
0
1
2
PC 1


Algorithm OPTICS - classes = 17
color
PC 2
PC 1


Algorithm Birch - classes = 3
color
PC 2
PC 1

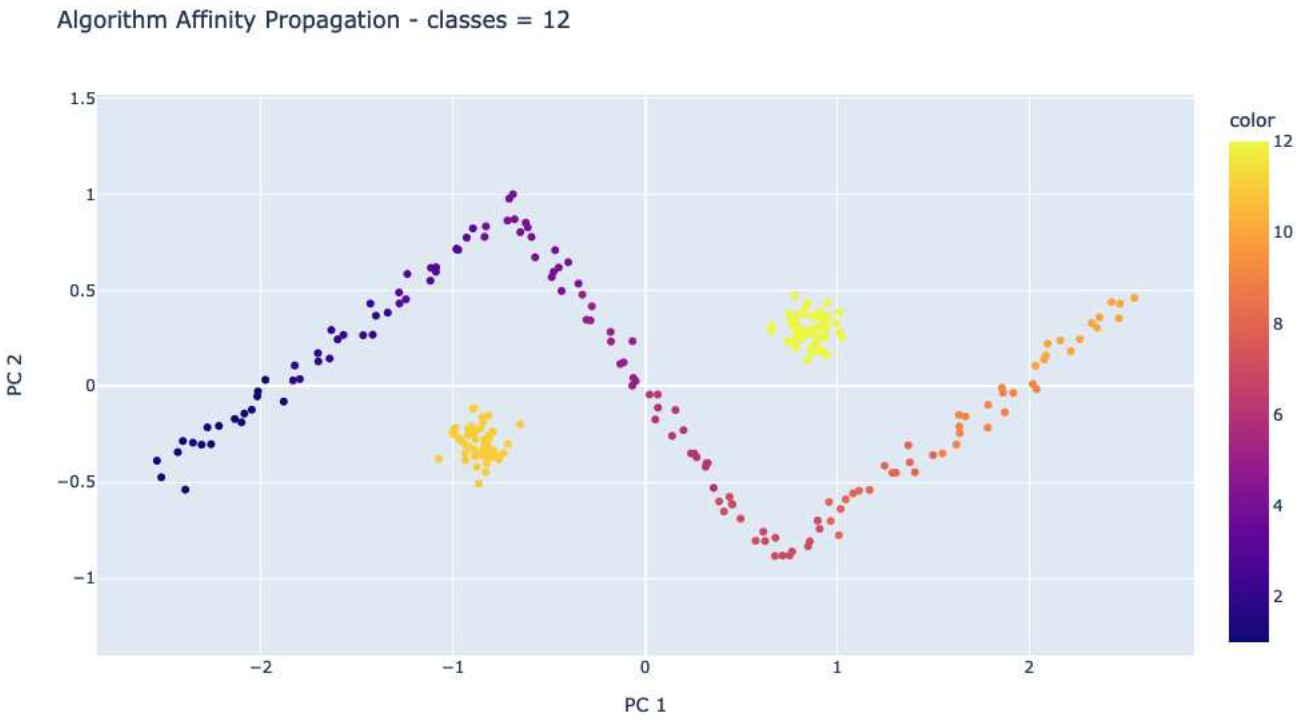
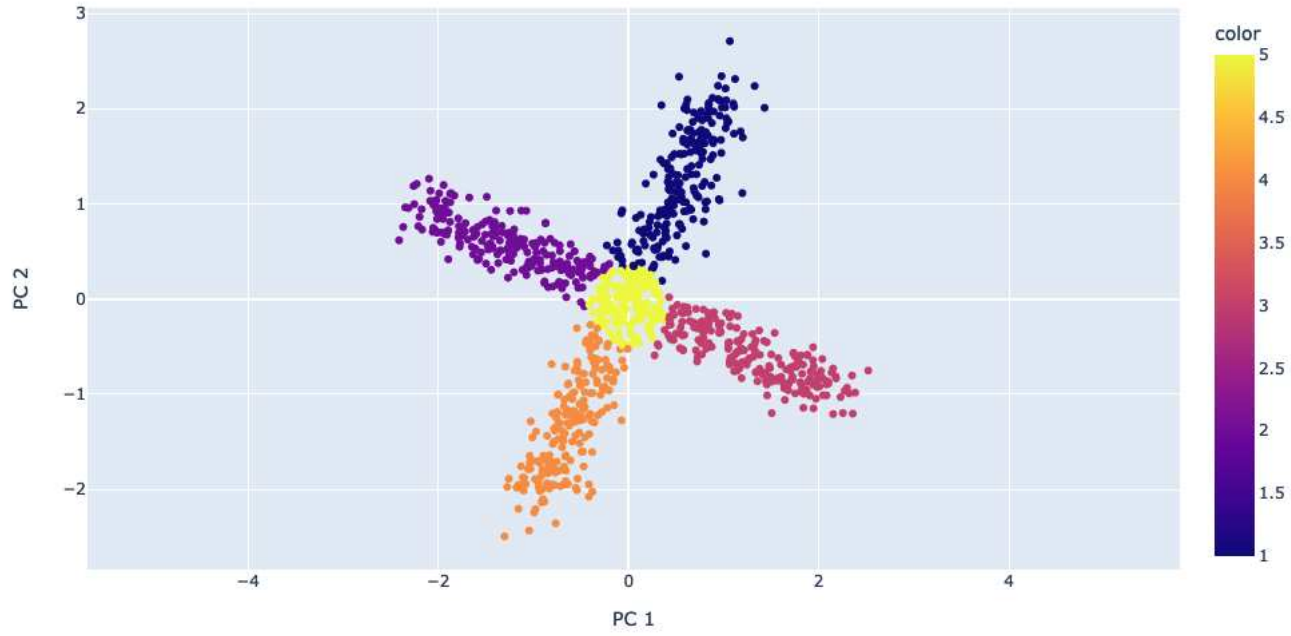



Tabella 3.21: zigzag

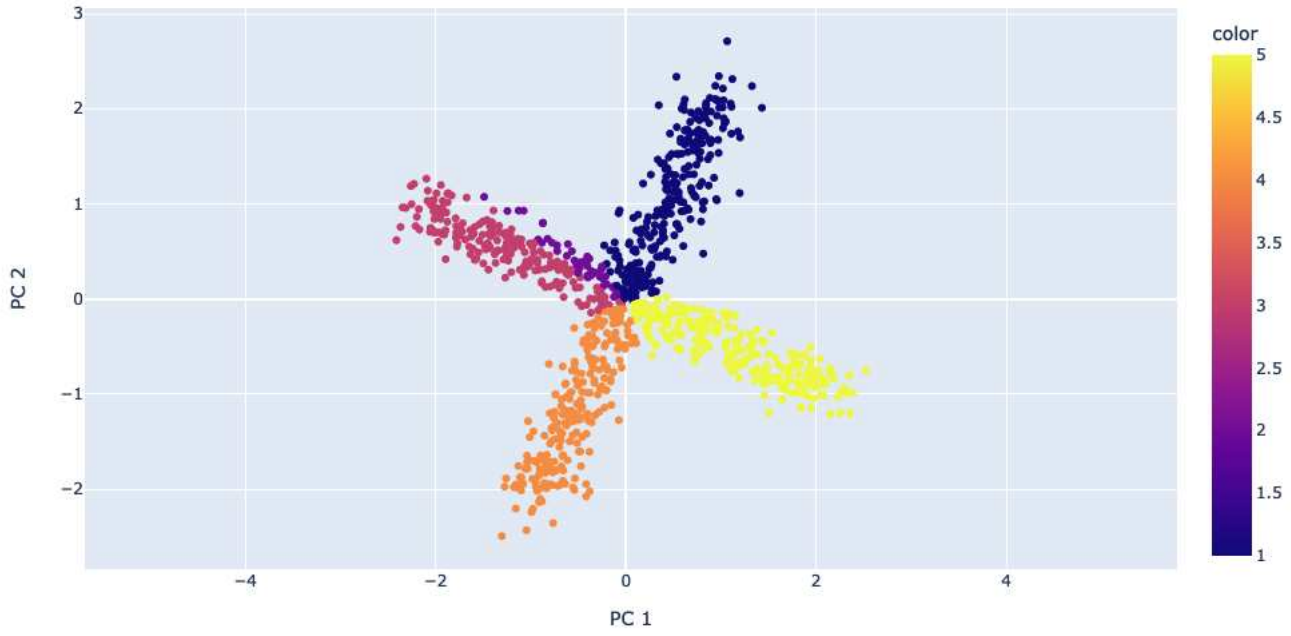
| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.156 | 0.334 | 0.168 |
| KMeans | 0.136 | 0.357 | 0.013 |
| Spectral C. | 0.144 | 0.3709 | 0.211 |
| Ward | 0.141 | 0.363 | 0.001 |
| DBscan | 1.0 | 1.0 | 0.001 |
| MeanShift | 0.1562 | 0.334 | 0.593 |
| Optics | -0.018 | 0.248 | 0.146 |
| Birch | 0.150 | 0.358 | 0.005 |
| Affinity Prop. | 0.281 | 0.567 | 0.106 |

CircleClustering > KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Meanshift, OPTICS, Birch CircleClustering < DBSCAN, Affinity Propagation

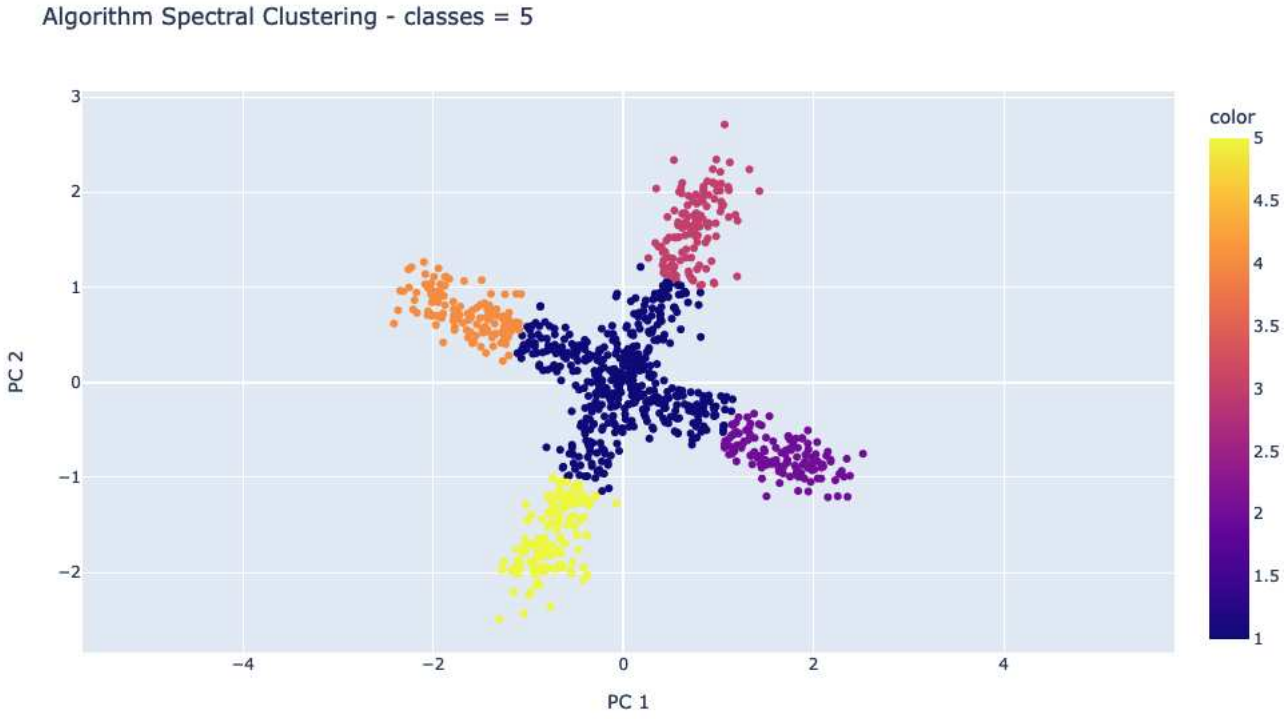
- Dataset: fuzzyx
  Samples = 1000, Features = 2, Classes = 5




Algorithm CircleClustering - classes = 5



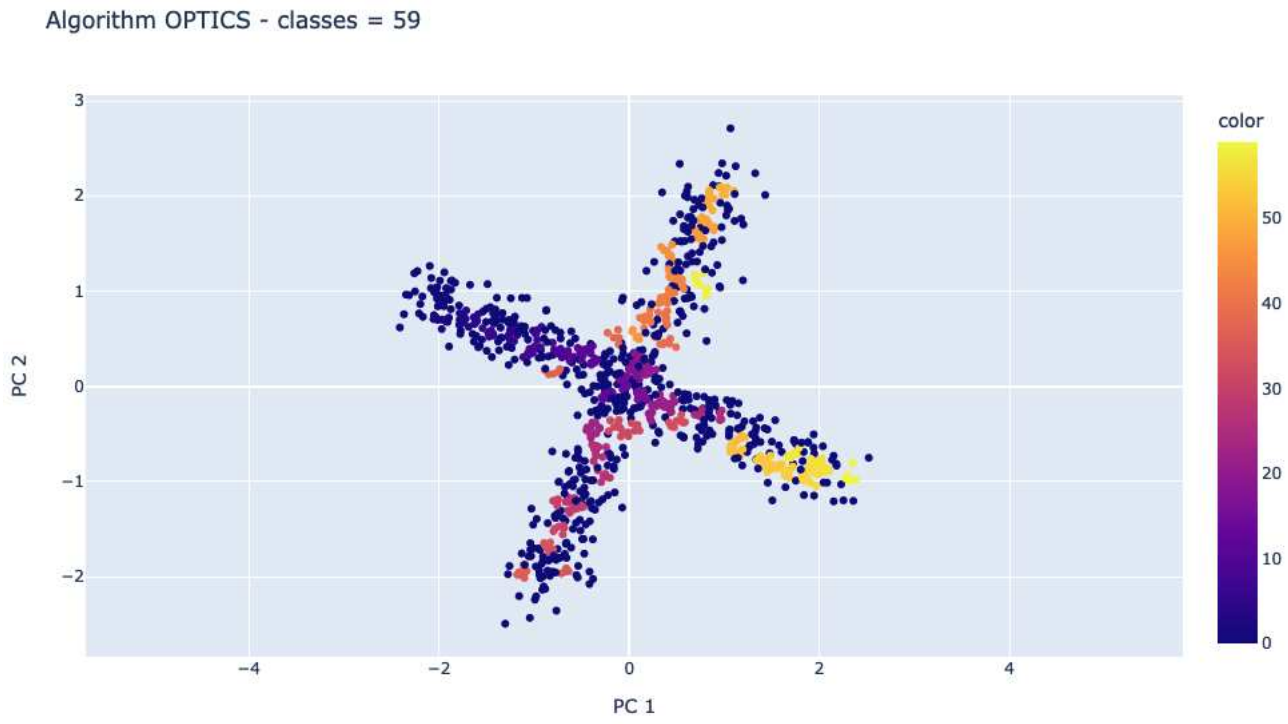

Algorithm KMeans - classes = 5
PC 2
PC 1
color
5
4.5
4
3.5
3
2.5
2
1.5
1

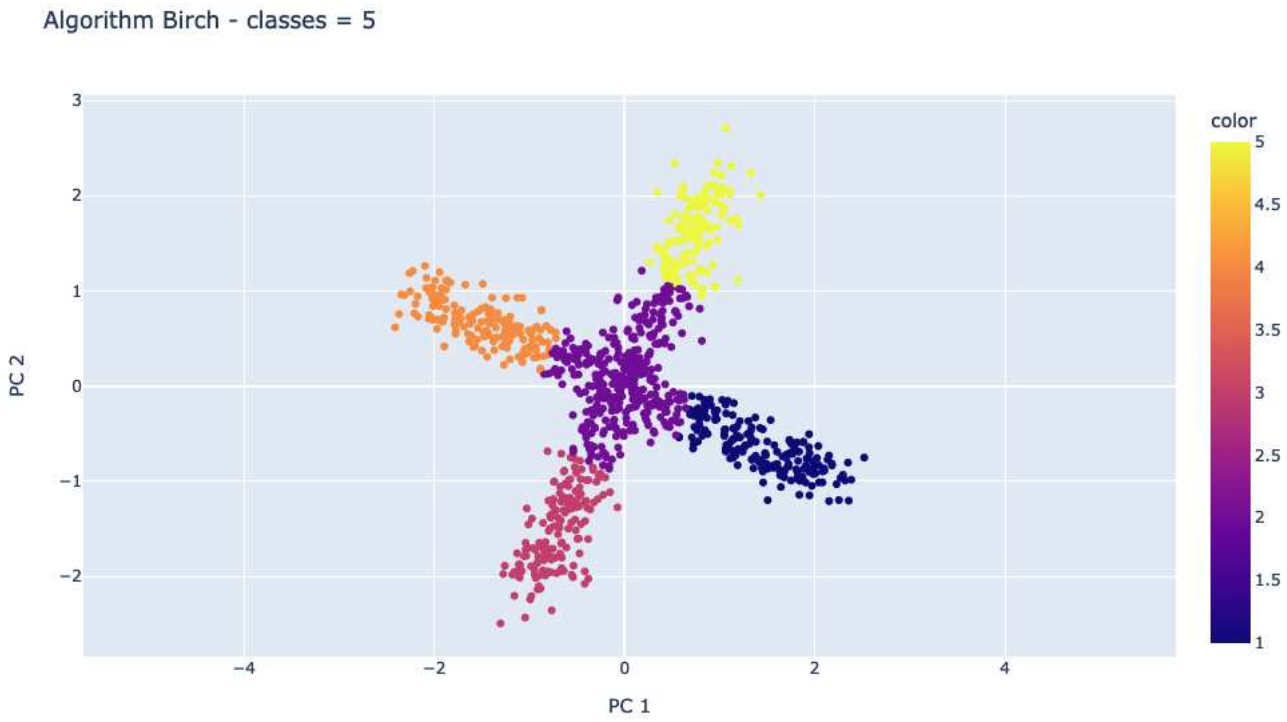

Algorithm Spectral Clustering - classes = 5
PC 2
PC 1
color
5
4.5
4
3.5
3
2.5
2
1.5
1


Algorithm Agglomerative Clustering (Ward) - classes = 5
color
5
4.5
4
3.5
3
2.5
2
1.5
1
PC 2
PC 1
3
2
1
0
−1
−2
−4
−2
0
2
4


Algorithm DBSCAN - classes = 1
color
1.4
1.2
1
0.8
0.6
PC 2
PC 1
3
2
1
0
−1
−2
−4
−2
0
2
4


Algorithm Meanshift - classes = 1
color
1.4
1.2
1
0.8
0.6
PC 2
PC 1
3
2
1
0
−1
−2
−4
−2
0
2
4


Algorithm OPTICS - classes = 59
color
50
40
30
20
10
0
PC 2
PC 1
3
2
1
0
−1
−2
−4
−2
0
2
4


Algorithm Birch - classes = 5
PC 2
PC 1
color
5
4.5
4
3.5
3
2.5
2
1.5
1


Algorithm Affinity Propagation - classes = 19
PC 2
PC 1
color
18
16
14
12
10
8
6
4
2

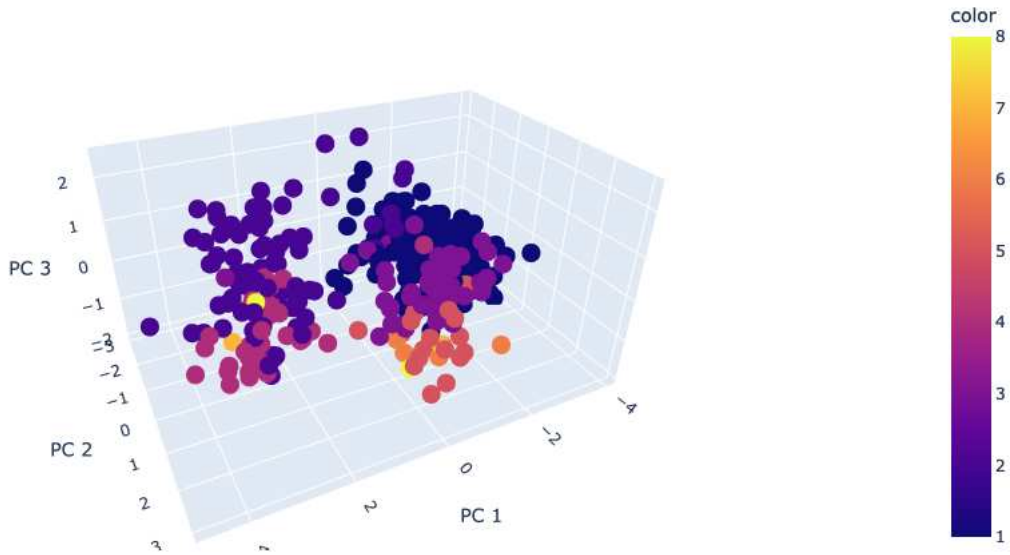
Tabella 3.22: fuzzyx

| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.719 | 0.746 | 1.294 |
| KMeans | 0.597 | 0.718 | 0.024 |
| Spectral C. | 0.308 | 0.565 | 0.264 |
| Ward | 0.637 | 0.736 | 0.017 |
| DBscan | 0.0 | 0.0 | 0.006 |
| MeanShift | 0.0 | 0.0 | 5.335 |
| Optics | 0.018 | 0.301 | 0.614 |
| Birch | 0.534 | 0.689 | 0.017 |
| Affinity Prop. | 0.325 | 0.641 | 1.659 |

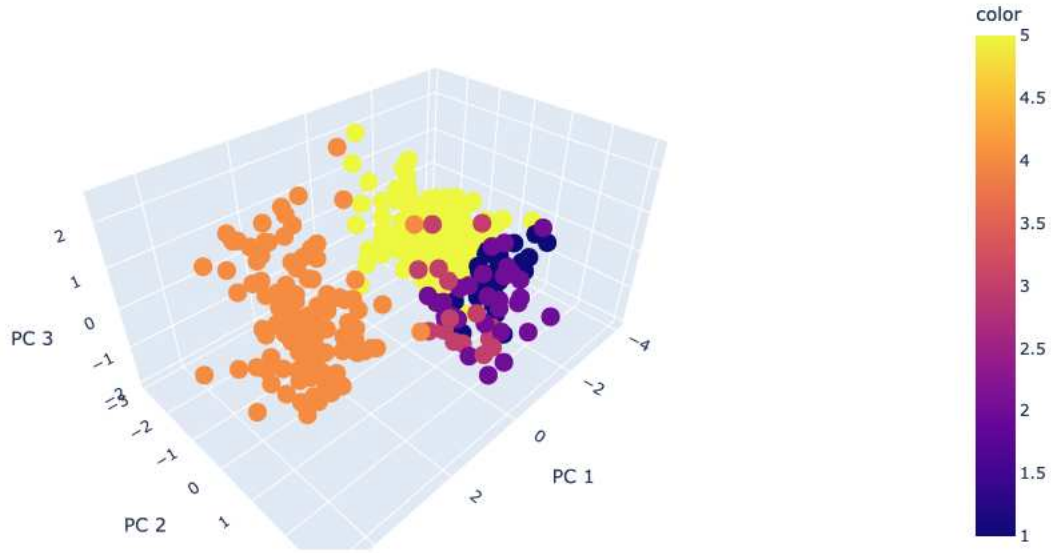
CircleClustering > KMeans, Spectral Clustering, Agglomerative Clustering (Ward), DBSCAN, Meanshift, OPTICS, Birch, Affinity Propagation

**Batteria: uci**

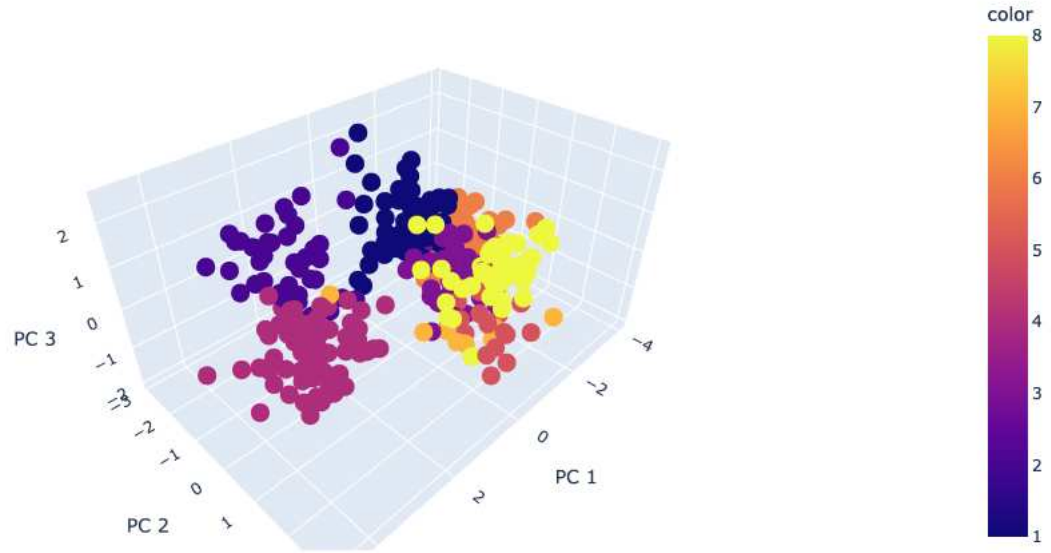
- Dataset: ecoli
  Samples = 336, Features = 7, Classes = 8



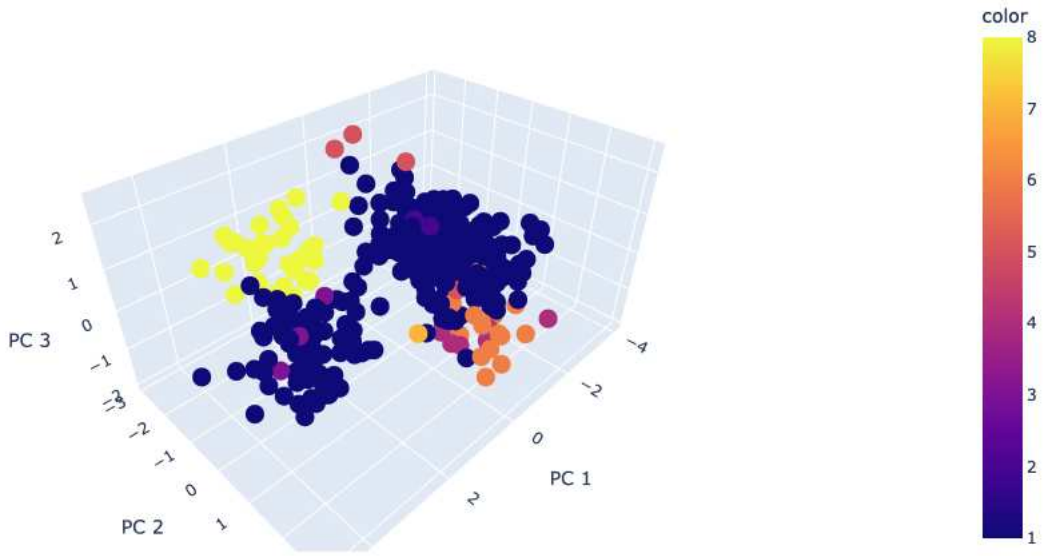
Algorithm CircleClustering - classes = 5



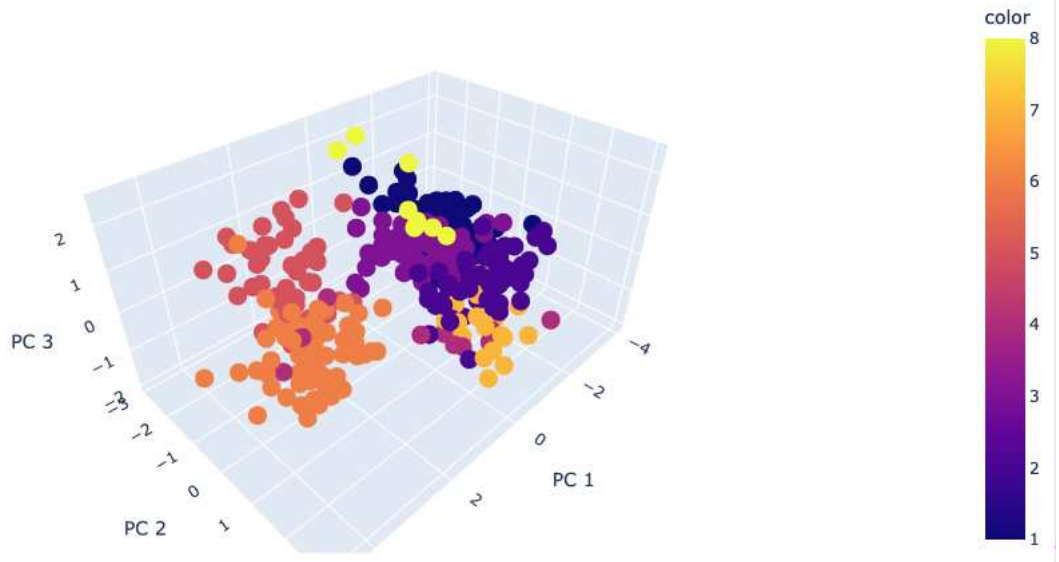

Algorithm KMeans - classes = 8



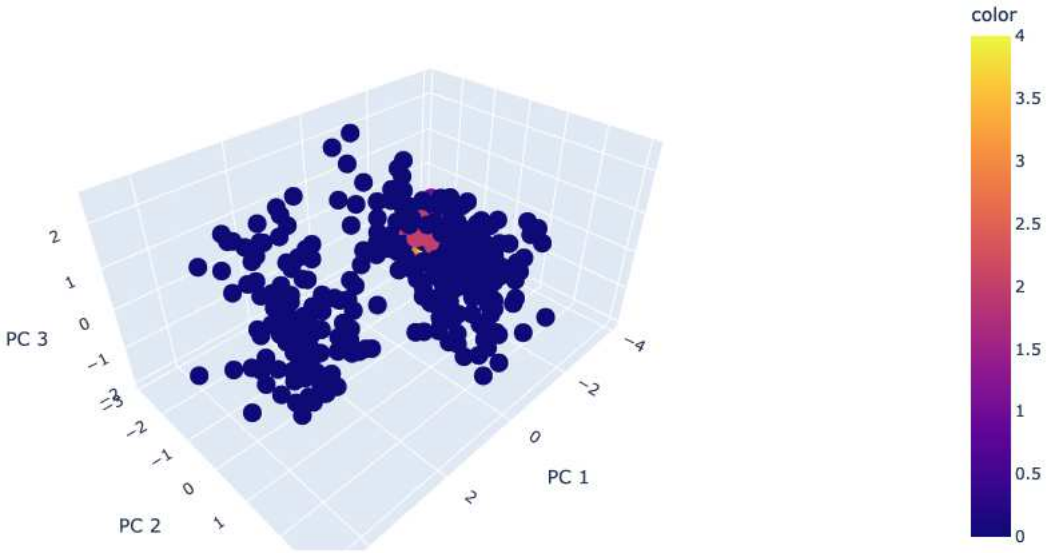
Algorithm Spectral Clustering - classes = 8



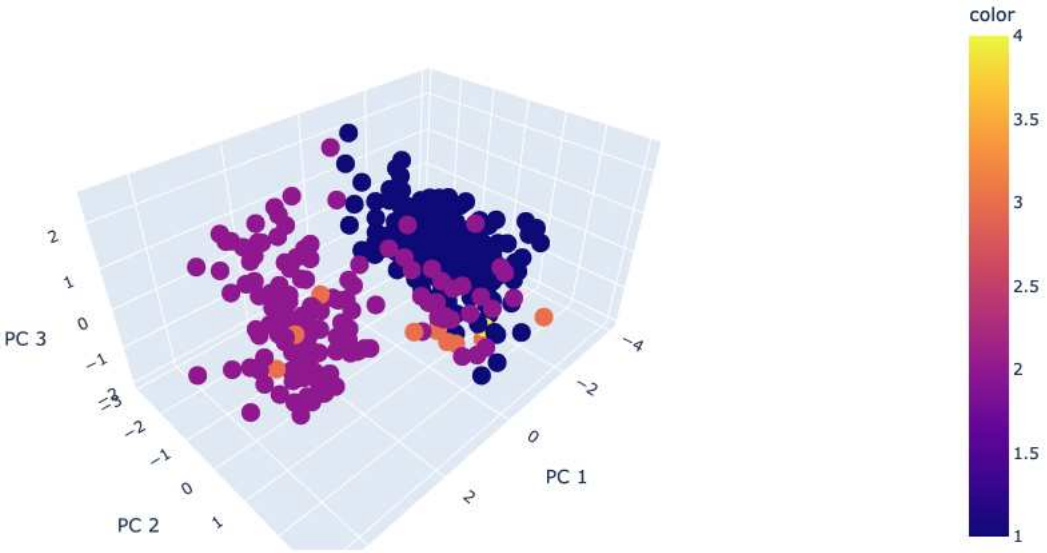

Algorithm Agglomerative Clustering (Ward) - classes = 8



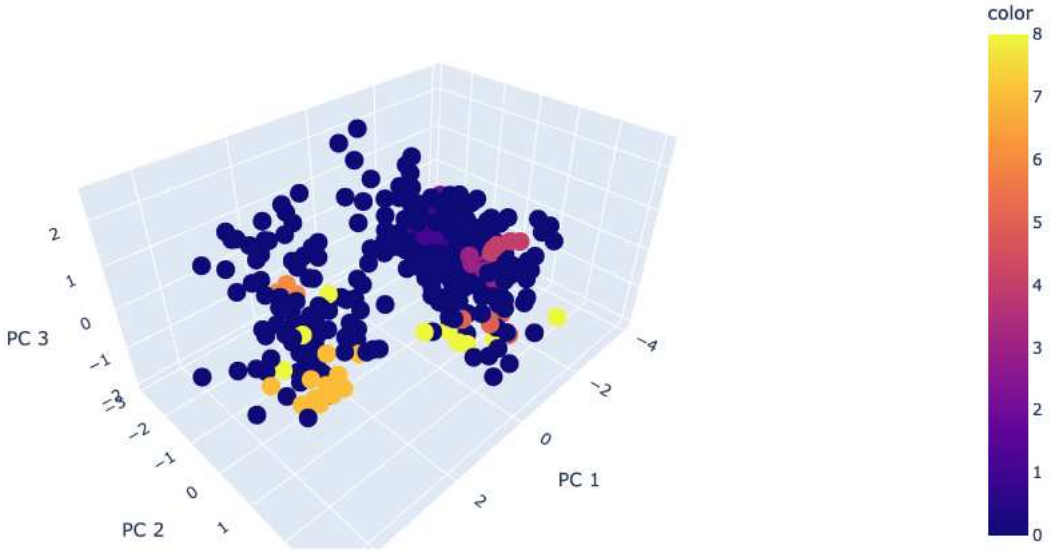
Algorithm DBSCAN - classes = 4



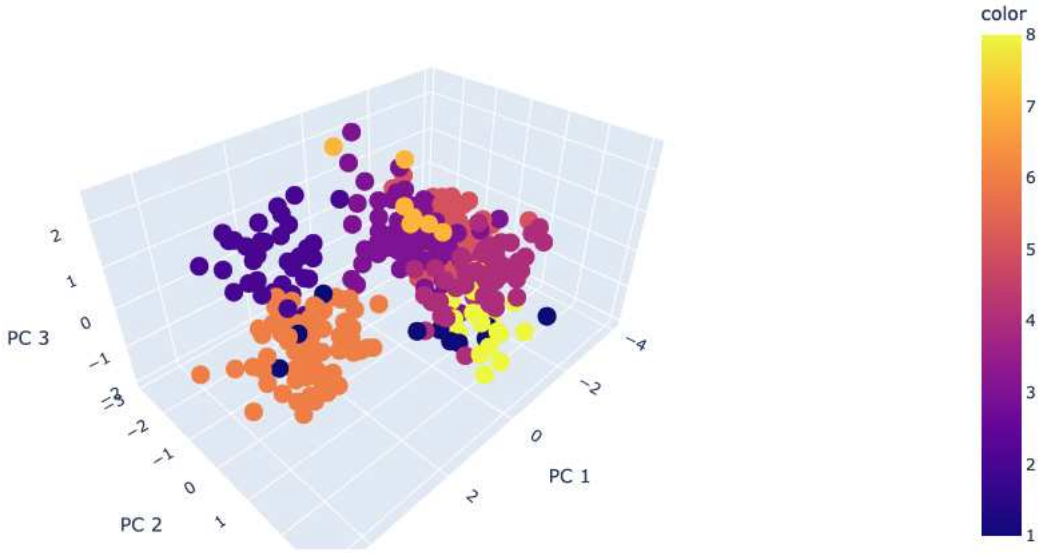

Algorithm Meanshift - classes = 4



Algorithm OPTICS - classes = 8




Algorithm Birch - classes = 8



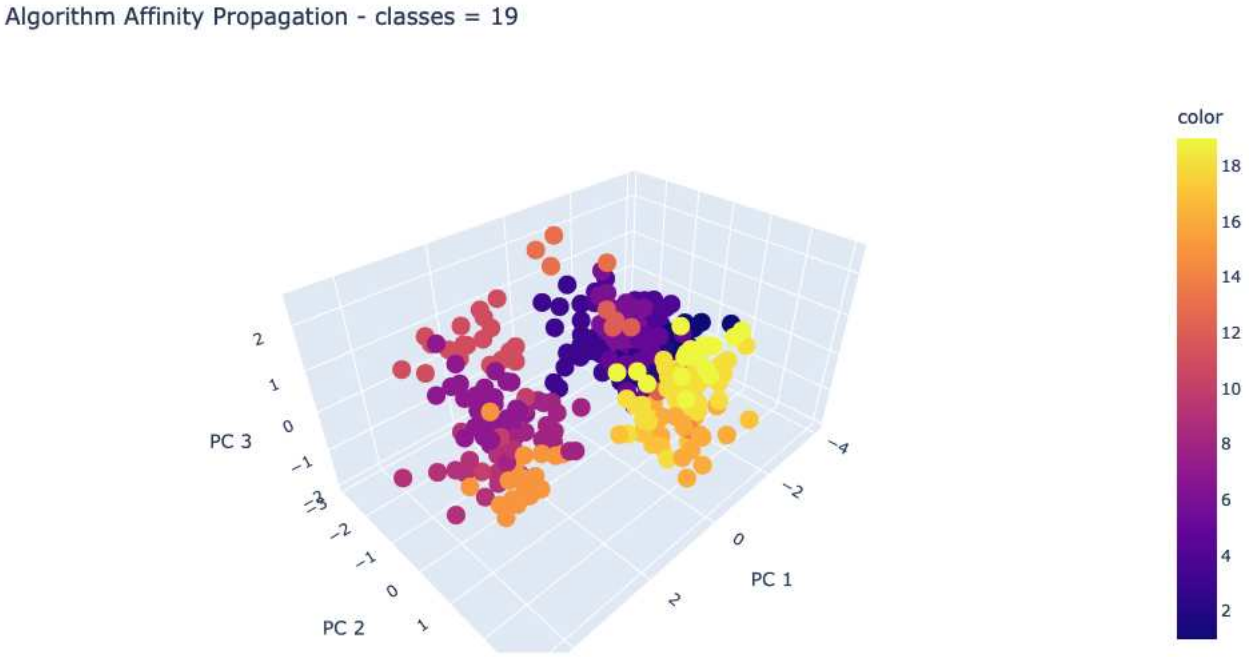



Tabella 3.23: fuzzyx

| | adjusted rand score | adjusted mutual info score | time cpu(s) |
|---|---|---|---|
| CircleClustering | 0.693 | 0.604 | 0.199 |
| KMeans | 0.411 | 0.579 | 0.041 |
| Spectral C. | 0.196 | 0.357 | 0.254 |
| Ward | 0.485 | 0.618 | 0.002 |
| DBscan | -0.062 | 0.052 | 0.002 |
| MeanShift | 0.425 | 0.402 | 1.064 |
| Optics | 0.071 | 0.1981 | 0.201 |
| Birch | 0.514 | 0.648 | 0.016 |
| Affinity Prop. | 0.238 | 0.505 | 0.112 |

CircleClustering > KMeans, Spectral Clustering, Agglomerative Clustering (Ward), DBSCAN, Meanshift, OPTICS, Birch, Affinity Propagation

### 3.2.4 Risultati globali

Sopra sono riportati solo alcuni esempi significativi della moltitudine di dataset analizzati. Sono ora esposti i risultati globali dell'algoritmo CircleClustering per ogni batteria, evidenziando la percentuale di volte che l'algoritmo è stato in grado di performare meglio delle altre tecniche.

**Batteria: fcps** ← CircleClustering ha raggiunto uno "adjusted rand score" superiore:

KMeans 44.44% delle volte

Spectral Clustering 33.33% delle volte

Agglomerative Clustering (Ward) 44.44% delle volte

DBSCAN 44.44% delle volte

Meanshift 88.89% delle volte

OPTICS 100.0% delle volte

birch 66.67% delle volte

Affinity Propagation 100.0% delle volte

**Batteria: sipu** ← CircleClustering ha raggiunto uno "adjusted rand score" superiore:

KMeans 6.25% delle volte

Spectral Clustering 12.5% delle volte

Agglomerative Clustering (Ward) 6.25% delle volte

DBSCAN 87.5% delle volte

Meanshift 81.25% delle volte

OPTICS 93.75% delle volte

birch 18.75% delle volte

Affinity Propagation 62.5% delle volte

**Batteria: wut** ← CircleClustering ha raggiunto uno "adjusted rand score" superiore:

KMeans 22.73% delle volte

Spectral Clustering 27.27% delle volte

Agglomerative Clustering (Ward) 22.73% delle volte

DBSCAN 72.73% delle volte

Meanshift 68.18% delle volte

OPTICS 90.91% delle volte

birch 31.82% delle volte

Affinity Propagation 77.27% delle volte

**Batteria: other** ← CircleClustering ha raggiunto uno "adjusted rand score" superiore:

KMeans 0.0% delle volte

Spectral Clustering 12.5% delle volte

Agglomerative Clustering (Ward) 0.0% delle volte

DBSCAN 87.5% delle volte

Meanshift 75.0% delle volte

OPTICS 87.5% delle volte

birch 0.0% delle volte

Affinity Propagation 87.5% delle volte

**Batteria: graves** ← CircleClustering ha raggiunto uno "adjusted rand score" superiore:

KMeans 80.0% delle volte

Spectral Clustering 70.0% delle volte

Agglomerative Clustering (Ward) 70.0% delle volte

DBSCAN 40.0% delle volte

Meanshift 80.0% delle volte

OPTICS 100.0% delle volte

birch 80.0% delle volte

Affinity Propagation 60.0% delle volte

**Batteria: uci** ← CircleClustering ha raggiunto uno "adjusted rand score" superiore:

KMeans 62.5% delle volte

Spectral Clustering 75.0% delle volte

Agglomerative Clustering (Ward) 62.5% delle volte

DBSCAN 100.0% delle volte

Meanshift 87.5% delle volte

OPTICS 87.5% delle volte

birch 87.5% delle volte

Affinity Propagation 87.5% delle volte

**Batteria: h2mg** ← CircleClustering ha raggiunto uno "adjusted rand score" superiore:

KMeans 20.63% delle volte

Spectral Clustering 38.1% delle volte

Agglomerative Clustering (Ward) 28.57% delle volte

DBSCAN 55.56% delle volte

Meanshift 88.89% delle volte

OPTICS 58.73% delle volte

birch 34.92% delle volte

Affinity Propagation 52.38% delle volte

**Batteria: g2mg** ← CircleClustering ha raggiunto uno "adjusted rand score" superiore:

KMeans 38.1% delle volte

Spectral Clustering 65.08% delle volte

Agglomerative Clustering (Ward) 88.89% delle volte

DBSCAN 96.83% delle volte

Meanshift 98.41% delle volte

OPTICS 82.54% delle volte

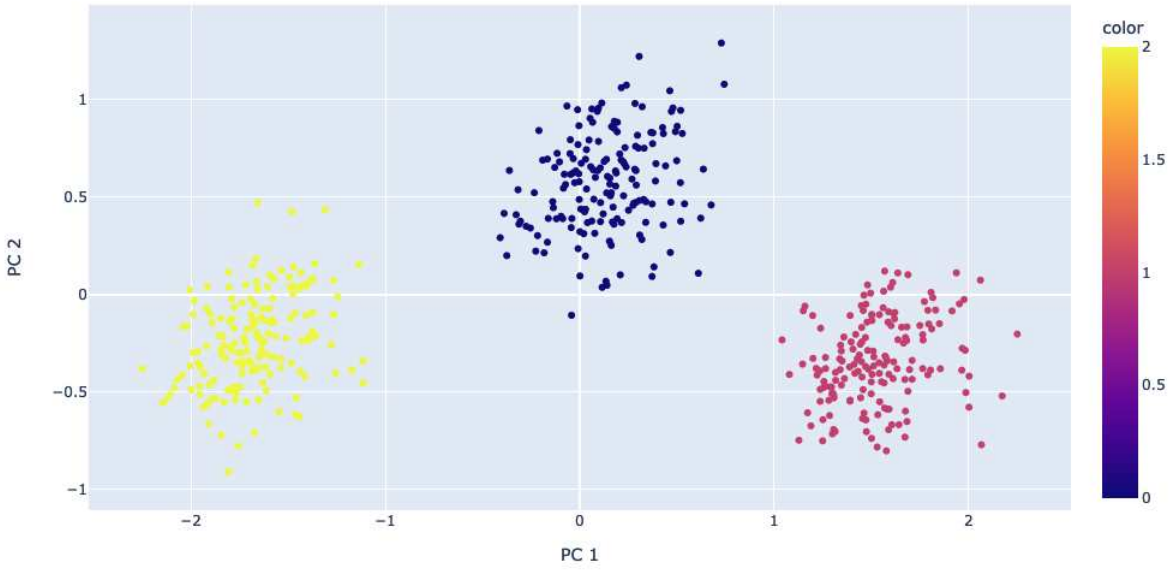
birch 92.06% delle volte

Affinity Propagation 100.0% delle volte

## 3.3 Scalabilità

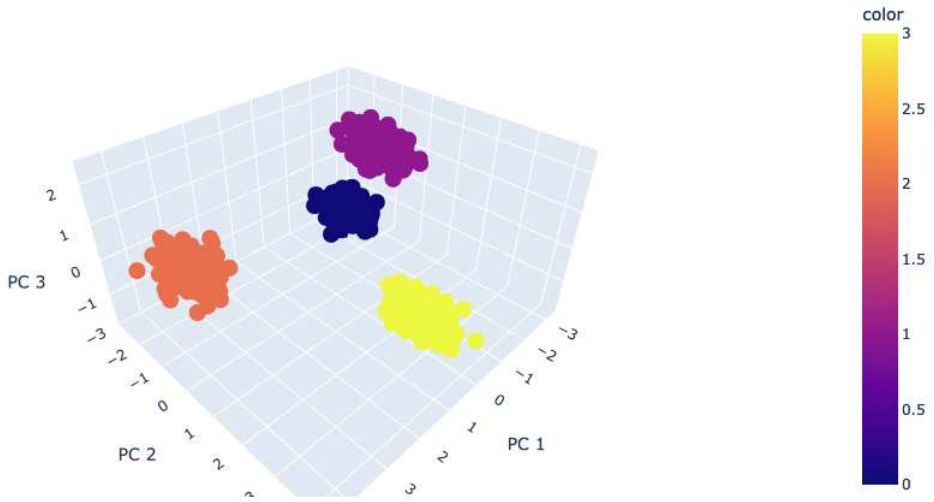
In questa sezione è analizzata una batteria di dataset artificiali di varie grandezze e dimensioni appositamente creata attraverso il framework python sklearn per esaminare i tempi di computazione dell'algoritmo CircleClustering in versione CPU e GPU. Tali valori sono poi confrontati con i tempi degli altri algoritmi di clustering. "/" indica che non è stato possibile effettuare la computazione, in quanto il tempo di esecuzione era troppo grande e/o la quantità di memoria allocata eccedeva dal limite imposto dal computer di test.

* **Samples = 500, Features = 2, Classes = 3**



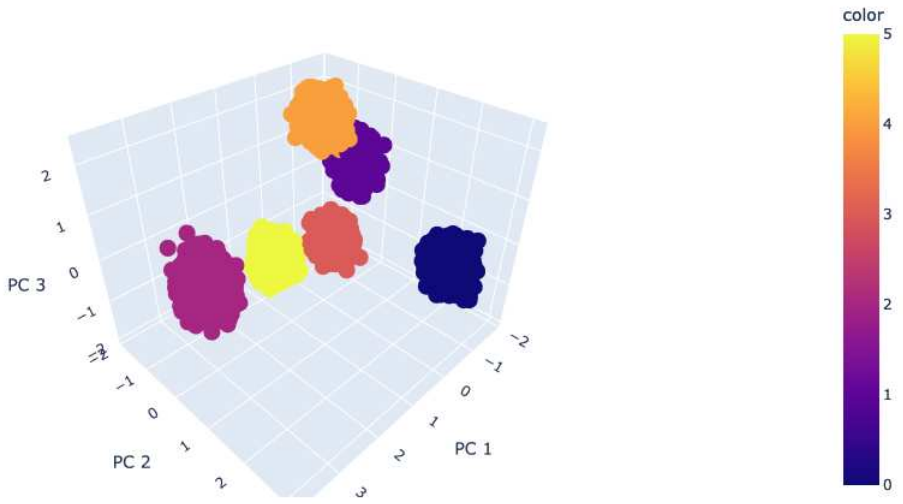
| | CPU(s) | CPU division matrix(s) | GPU division matrix(s) |
|---|---|---|---|
| CircleClustering | 2.86 | 1.28 | 1.74 (speed up x-1.35) |
| KMeans | 0.16 | | |
| Spectral C. | 0.16 | | |
| Ward | 0.007 | | |
| DBscan | 0.003 | | |
| MeanShift | 0.91 | | |
| Optics | 0.29 | | |
| Birch | 0.008 | | |
| Affinity Prop. | 0.47 | | |

* **Samples = 1000, Features = 10, Classes = 4**



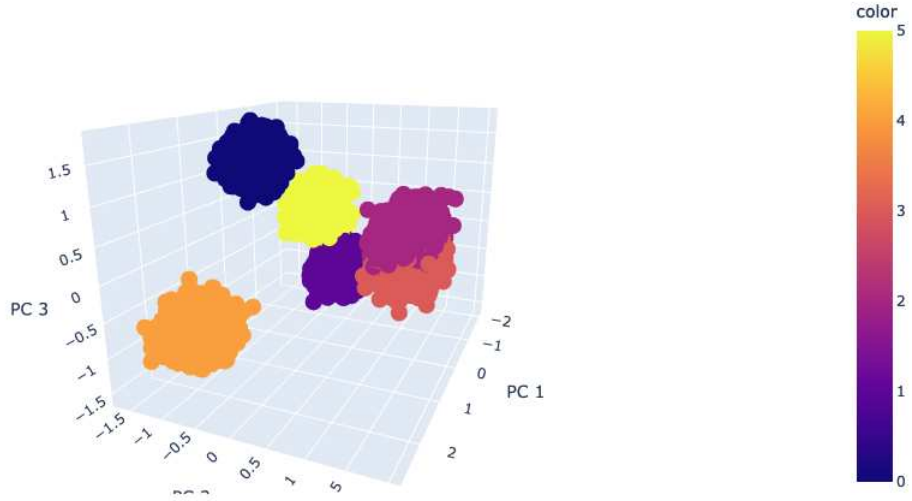

| | CPU(s) | CPU division matrix(s) | GPU division matrix(s) |
|---|---|---|---|
| CircleClustering | 0.24 | 2.47 | 3.45 (speed up x-1.36) |
| KMeans | 0.37 | | |
| Spectral C. | 0.28 | | |
| Ward | 0.03 | | |
| DBscan | 0.02 | | |
| MeanShift | 2.20 | | |
| Optics | 0.63 | | |
| Birch | 0.06 | | |
| Affinity Prop. | 1.04 | | |

* **Samples = 10000, Features = 6, Classes = 6**



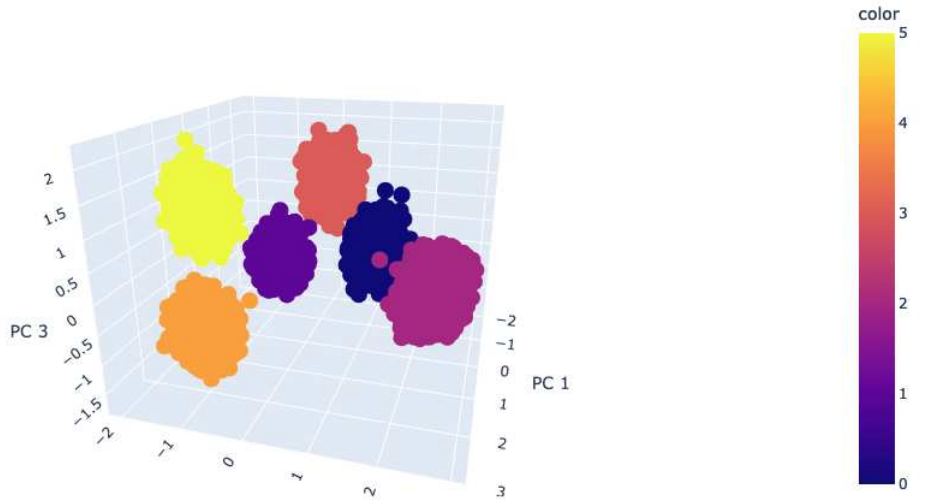

| | CPU(s) | CPU division matrix(s) | GPU division matrix(s) |
|---|---|---|---|
| CircleClustering | 7.96 | 423.8 | 86.9 (speed up x4,87) |
| KMeans | 0.75 | | |
| Spectral C. | 26.5 | | |
| Ward | 3.29 | | |
| DBscan | 0.62 | | |
| MeanShift | 140.1 | | |
| Optics | 9.14 | | |
| Birch | 0.16 | | |
| Affinity Prop. | 286.6 | | |

* **Samples = 15000, Features = 3, Classes = 6**



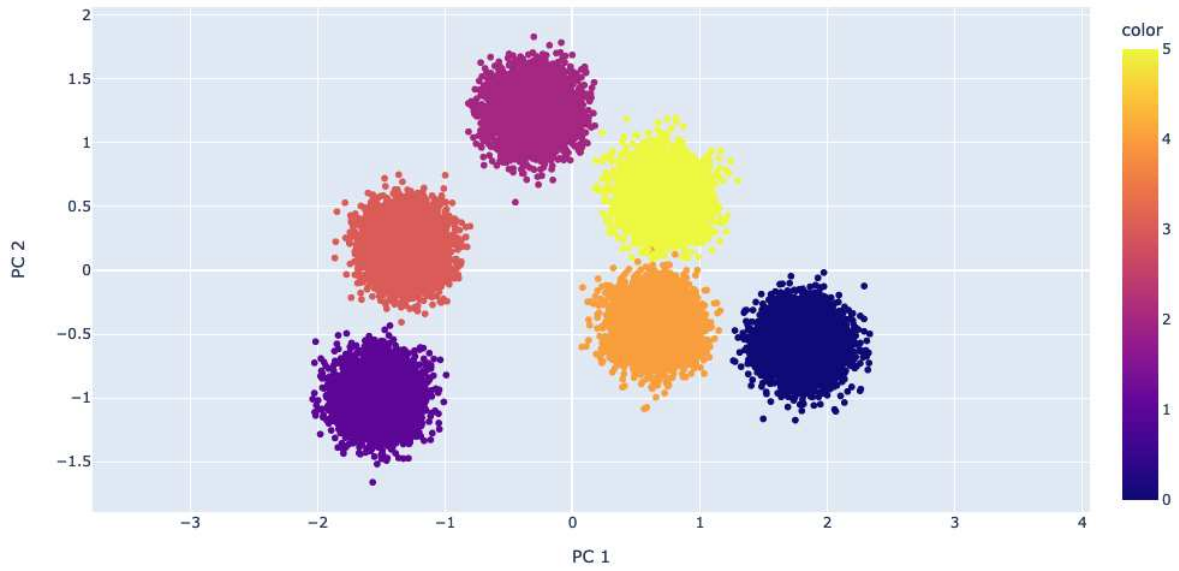

| | CPU(s) | CPU division matrix(s) | GPU division matrix(s) |
|---|---|---|---|
| CircleClustering | / | 414.1 | 70.1 (speed up x5.9) |
| KMeans | 1.01 | | |
| Spectral C. | 200.49 | | |
| Ward | 9.49 | | |
| DBscan | 11.58 | | |
| MeanShift | 419.02 | | |
| Optics | 16.6 | | |
| Birch | 0.27 | | |
| Affinity Prop. | 910.8 | | |

* **Samples = 25000, Features = 5, Classes = 6**



| | CPU(s) | CPU division matrix(s) | GPU division matrix(s) |
|---|---|---|---|
| CircleClustering | / | 1072.04 | 174.5 (speed up x6) |
| KMeans | 0.612 | | |
| Spectral C. | / | | |
| Ward | 32.19 | | |
| DBscan | 2.62 | | |
| MeanShift | 128.6 | | |
| Optics | 37.78 | | |
| Birch | 0.40 | | |
| Affinity Prop. | / | | |

* **Samples = 35000, Features = 2, Classes = 6**




| | CPU(s) | CPU division matrix(s) | GPU division matrix(s) |
|---|---|---|---|
| CircleClustering | / | 2669.69 | 71.5 (speed up x37) |
| KMeans | 2.47 | | |
| Spectral C. | / | | |
| Ward | 146.0 | | |
| DBscan | 8.44 | | |
| MeanShift | 499.5 | | |
| Optics | 63.34 | | |
| Birch | 0.65 | | |

*Continua nella prossima pagina*

Tabella 3.29: *(Continued)*

| | CPU | CPU division matrix | GPU division matrix |
|---|---|---|---|
| Affinity Prop. | / | | |

## 3.4 Numero di cluster corretti

Analizziamo infine la corretta predizione del numero di classi presenti in un dataset da parte del CircleClustering e delle altre metodologie esaminate. Gli algoritmi presi in considerazione, oltre ovviamente al metodo CircleClustering da noi sviluppato sono:

- Dbscan

- Optics

- Meanshift

- Affinity propagation

Di tali tecniche, solo Meanshift risulta essere un vero e proprio algoritmo non supervisionato. Esso infatti non accetta in ingresso nessun parametro e tenta di identificare in modo completamente autonomo il numero corretto di cluster e di effettuare un buon partizionamento. Le altre tecniche, ovvero DBscan, Optics e Affinity propagation non ricevono in ingresso il numero di cluster, ma fanno utilizzo di valori di threshold, che se non dichiarati, sono impostati ad un valore di default. In particolare, Dbscan richiede un parametro *eps* (default = 0.5) che rappresenta la distanza massima tra due campioni perchè questi siano considerati come appartenenti allo stesso "vicinato" e richiede un parametro aggiuntivo *min_samples* (default = 5), numero di campioni o pesi massimi per decretare un *core point*. Optics richiede anch'esso il parametro *min_samples* e ulteriori parametri. Affinity Propagation vuole un fattore di Damping che se non impostato manualmente è posizionato di default a 0.5. Tali algoritmi di clustering non sono quindi in grado di fare completa inferenza sui dati. Tuttavia, anche se tali algoritmi richiedono i seguenti parametri, tra questi non rientra il numero esatto di cluster, k. Perciò tali tecniche tentano di fare una previsione sul numero esatto di cluster ed è corretto quindi confrontarli, valutandoli nella loro versione default, ovvero non fornendogli nessuna comunicazione riguardo le caratteristiche del dataset analizzato. Sono ora esposti i risultati delle varie procedure.

* **Batteria: fcps**

| Algoritmo | previsioni corrette | totale |
|---|---|---|
| **CircleClustering** | **3** | **9** |
| MeanShift | 2 | 9 |
| DBscan | 2 | 9 |
| Optics | 0 | 9 |
| Affinity Prop. | 0 | 9 |

* **Batteria: sipu**

| Algoritmo | previsioni corrette | totale |
|---|---|---|
| **CircleClustering** | **7** | **16** |
| MeanShift | 1 | 16 |
| DBscan | 1 | 16 |
| Optics | 1 | 16 |
| Affinity Prop. | 0 | 16 |

* **Batteria: wut**

| Algoritmo | previsioni corrette | totale |
|---|---|---|
| **CircleClustering** | **5** | **22** |
| MeanShift | 4 | 22 |
| DBscan | 3 | 22 |
| Optics | 0 | 22 |
| Affinity Prop. | 0 | 22 |

* **Batteria: other**

| Algoritmo | previsioni corrette | totale |
|---|---|---|
| **CircleClustering** | **1** | **8** |
| MeanShift | 0 | 8 |
| DBscan | 0 | 8 |
| Optics | 0 | 8 |
| Affinity Prop. | 0 | 8 |

* **Batteria: graves**

| Algoritmo | previsioni corrette | totale |
|---|---|---|
| CircleClustering | 2 | 10 |
| MeanShift | 1 | 10 |
| **DBscan** | **5** | **10** |
| Optics | 0 | 10 |
| Affinity Prop. | 0 | 10 |

* **Batteria: uci**

| Algoritmo | previsioni corrette | totale |
|---|---|---|
| **CircleClustering** | **4** | **8** |
| MeanShift | 1 | 8 |
| DBscan | 0 | 8 |
| Optics | 1 | 8 |
| Affinity Prop. | 0 | 8 |

* **Batteria: g2mg**

| Algoritmo | previsioni corrette | totale |
|---|---|---|
| **CircleClustering** | **33** | **62** |
| MeanShift | 13 | 62 |
| DBscan | 5 | 62 |
| Optics | 4 | 62 |
| Affinity Prop. | 0 | 62 |

* **Batteria: h2mg**

| Algoritmo | previsioni corrette | totale |
|---|---|---|
| CircleClustering | 41 | 62 |
| MeanShift | 3 | 62 |
| **DBscan** | **45** | **62** |
| Optics | 45 | 62 |
| Affinity Prop. | 0 | 62 |

# Conclusioni e lavoro futuro

In questo lavoro di tesi è stato proposto un nuovo algoritmo di clustering basato sul cerchio unitario e sugli alberi gerarchici. Tale algoritmo è stato confrontato con altre tecniche di clustering quali KMeans, Spectral Clustering, Agglomerative Clustering (Ward), Birch, Affinity Propagation, DBscan, Optics e Meanshift, le quali sono state divise secondo la richiesta di k, parametro in input rappresentate il numero di classi presenti in un determinato dataset. E' stata quindi generata una suite benchmark composta da 8 batterie in grado di testare diverse caratteristiche degli algoritmi in considerazione, come i risultati del labeling e la corretta predizione del numero di classi. Inoltre è stata realizzata una versione parallela dell'algoritmo, utilizzante CUDA, per cercare di ovviare ai tempi computazionali molto pesanti generati dalla matrice di (dis)similarità. Nonostante i risultati siano stati incoraggianti, rimangono ancora aperte molte sfide nel processo esecutivo dell'algoritmo. In particolare tre sviluppi futuri possono essere i seguenti:

- Il primo riguarda il preprocessing della seconda fase dell'algoritmo CircleClustering. Sarebbe necessario uno studio approfondito riguardo l'eliminazione del rumore all'interno di un istogramma di densità. Tale pulizia iniziale influisce enormemente sul parametro k e quindi sul risultato della tecnica.

- Un secondo sviluppo, più dispendioso, potrebbe essere quello di generare la PDF (probability density function) dall'istogramma di densità e utilizzare essa per ricavare il parametro k, calcolando le aree sulle partizioni di funzione.

- Uno studio approfondito riguardo il confronto del numero k generato dal CircleClustering e dagli indici presenti in letteratura.

# Bibliografia


[1] Serhat Emre Akhanli and Christian Henning. Comparing clusterings and numbers of clusters by aggregation of calibrated clustering validity indexes. *Department of Statistical Science, London, UK*, 2020.

[2] P.Campadelli A. Bertoni and G.Grossi. An approximation algorithm for the maximum cut problem and its experimental analysis. *Dipartimento di Scienze dell'informazione, Università degli studi di Milano*.

[3] Marek Gagolewski. A framework for benchmarking clustering algorithms. *Deakin University*, 2022.

[4] Kamalpreet Bindra and Anuranjan Mishra. A detailed study of clustering algorithms. In *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pages 371–376, 2017. doi: 10.1109/ICRITO.2017.8342454.

[5] Dongkuan Xu Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2015.

[6] Ying Lai, Ratko Orlandic, Wai Gen Yee, and Sachin Kulkarni. Scalable clustering for large high-dimensional data based on data summarization. In *2007 IEEE Symposium on Computational Intelligence and Data Mining*, pages 456–461, 2007. doi: 10.1109/CIDM.2007.368910.

[7] Gonzalo E. Paredes and Luis S. Vargas. Circle-clustering: A new heuristic partitioning method for the clustering problem. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012. doi: 10.1109/IJCNN.2012.6252570.

[8] Xiaowei Xu, M. Ester, H.-P. Kriegel, and J. Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *Proceedings 14th International Conference on Data Engineering*, pages 324–331, 1998. doi: 10.1109/ICDE.1998.655795.

[9] Rogério Rodrigues de de Vargas and Benjamín Rene Callejas Bedregal. A way to obtain the quality of a partition by adjusted rand index. In *2013 2nd Workshop-School on Theoretical Computer Science*, pages 67–71, 2013. doi: 10.1109/WEIT.2013.33.

[10] Nisha and Puneet Jai Kaur. Cluster quality based performance evaluation of hierarchical clustering method. In *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, pages 649–653, 2015. doi: 10.1109/NGCT.2015.7375201.

[11] Armağan Karabina and Erdal Kiliç. An automated clustering algorithm based on agglomerative clustering. In *2016 24th Signal Processing and Communication Application Conference (SIU)*, pages 1801–1804, 2016. doi: 10.1109/SIU.2016.7496111.

[12] Fatma Najar, Sami Bourouis, Nizar Bouguila, and Safiya Belghith. A comparison between different gaussian-based mixture models. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 704–708, 2017. doi: 10.1109/AICCSA.2017.108.

[13] vahid Mirjalili Sebastian Raschka, Yuxi (Hayden) Liu. *Machine Learning with Pytorch and Scikit-Learn*.