POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

AdaPro: algoritmo di *clustering* semi-supervisionato con selezione automatica degli iperparametri

Relatore
Prof.ssa Elena Maria Baralis

Correlatori
Prof. Paolo Garza
Dott. Andrea Pasini

Candidato
Federico Brignone
matricola: 253119

Anno accademico 2019-2020

Ai miei nonni

Sommario

Lo scopo di questa tesi di laurea magistrale è quello di spiegare il funzionamento di AdaPro (*ADAptive clustering algorithm with PROtotypes*), algoritmo di *clustering* gerarchico semi-supervisionato con selezione automatica degli iperparametri. L'algoritmo è stato messo a punto dal Politecnico di Torino, in principio per risolvere alcuni problemi legati alle analisi di dati mineralogici. Successivamente si sono volute mettere alla prova le sue capacità anche al di fuori del contesto per il quale era stato pensato, rendendosi così conto che l'algoritmo lavorava molto bene. La caratteristica principale di AdaPro è la capacità di autoconfigurarsi, rendendo minimo lo sforzo dell'utente. L'algoritmo è inoltre spesso in grado di generare risultati di qualità migliore rispetto agli algoritmi di *clustering* standard. Per mettere in luce le sue caratteristiche, l'algoritmo è stato confrontato, nel funzionamento e nei risultati ottenuti, con altri noti algoritmi di *clustering* non supervisionati e con MPCK-Means, un altro algoritmo di *clustering* semi-supervisionato.

Indice

Capitolo 1

Introduzione

In questa tesi di laurea magistrale verrà introdotto un nuovo algoritmo di *clustering* gerarchico semi-supervisionato, denominato AdaPro (acronimo di *ADAptive clustering algorithm with PROtotypes*).

Gli algoritmi di *clustering*, che per funzionare sfruttano solamente le somiglianze che intercorrono tra i dati stessi, rientrano in una branca del *Machine Learning* che oggi sembra essere già consolidata rispetto alla corrispettiva degli algoritmi supervisionati, ovvero quella di tutti quegli algoritmi che apprendono grazie alla conoscenza delle etichette di classe relative ad ogni elemento in una collezione di dati. Ciò si può intuitivamente spiegare con il fatto che gli algoritmi supervisionati tendono, in genere, ad ottenere *performance* migliori rispetto ai corrispettivi algoritmi non supervisionati. Non sempre però si hanno a disposizione le etichette di classe, e la loro produzione manuale richiederebbe, oltre che lo sforzo di un esperto di dominio, anche un costo che risulterebbe tanto più elevato quanto più grande risultasse la mole di dati da etichettare.

Un compromesso fra queste due famiglie di algoritmi è rappresentato dagli algoritmi di *clustering* semi-supervisionati che, pur mantenendo la stessa

logica, comune a tutti gli algoritmi di *clustering*, cioè cercando il raggruppamento degli oggetti simili negli stessi *cluster*, si possono avvalere anche di una porzione dei dati che sia provvista di etichetta, sfruttandola in modo da raffinare i risultati.

L'algoritmo proposto in questa tesi, AdaPro, nasce proprio dall'obiettivo di rispondere ad un problema simile proposto da una nota azienda petrolifera italiana, il quale richiede di classificare una grande mole di dati, ma disponendo delle etichette di classe solo per una piccola parte di essi.

AdaPro possiede inoltre la caratteristica di essere in grado di selezionare internamente i propri iperparametri, in particolare le altezze migliori alle quali tagliare i dendrogrammi, strutture logiche generate dagli algoritmi di *clustering* gerarchici. Le applicazioni classiche di questi algoritmi avrebbero richiesto di effettuare vari tentativi per ogni diverso campione geologico prima di capire quali fossero i valori migliori. AdaPro quindi, con la selezione automatica di questi iperparametri, riesce a minimizzare lo sforzo dell'utente, rendendo l'analisi dei dati un processo praticamente del tutto automatizzato.

L'esposizione di questo lavoro di tesi sarà strutturata nel seguente modo: nel primo capitolo si richiameranno le basi dell'apprendimento automatico, dove verrà spiegata la differenza tra algoritmi supervisionati e non supervisionati. Inoltre, verrà spiegato in cosa consista il *pre-processing*, insieme di tecniche alle quali è necessario sottoporre i dati prima di porli in *input* a qualsivoglia algoritmo.
Nel secondo capitolo, verranno presi in esame tutti i principali algoritmi di *clustering* esistenti, come K-Means, il *clustering* gerarchico e DBSCAN. Verrà presentato subito dopo anche MPCK-Means, una rivisitazione semi-supervisionata di K-Means.

Nel terzo capitolo, verranno illustrati tutti gli strumenti matematici che potranno essere sfruttati come indici di qualità di un'assegnazione di *clustering*. Questi verranno utilizzati in seguito per valutare le *performance* di AdaPro in confronto agli altri algoritmi.

Nel quarto capitolo verrà spiegato nel dettaglio il funzionamento di AdaPro, mettendo in luce le sue qualità e i suoi meccanismi interni. A questo verrà annesso 1-Level-AdaPro, una variante semplificata di AdaPro volta a dimostrarne l'efficacia in ogni suo componente.

Nel quinto capitolo verranno infine testati vari algoritmi su diverse collezioni di dati e se ne valuterà la qualità dei risultati facendo uso degli indici presentati nel terzo capitolo.

Capitolo 2

L'apprendimento automatico

Il *Machine Learning*, termine che in italiano potremmo tradurre con "apprendimento automatico", è una branca dell'intelligenza artificiale il cui obiettivo è permettere alle macchine, cioè ai *computer*, di apprendere automaticamente qualcosa dall'esperienza, senza che ci sia bisogno che queste vengano preventivamente programmate. Per esperienza si intende una collezione di dati, che può essere fissa e immutabile, o anche espandersi nel corso del tempo. Riportiamo una delle definizioni più famose, del professor Mitchell, direttore del dipartimento di *Machine Learning* presso la Carnegie Mellon University: *"Si dice che un programma per computer impari da un'esperienza E rispetto a qualche classe di compiti T con prestazioni P, se le sue prestazioni nei compiti della classe T, misurate come P, migliorano con l'esperienza E"* [1].

2.1 Algoritmi supervisionati e non supervisionati

Tra gli algoritmi di *Machine Learning*, ne esistono principalmente due grandi famiglie: quella degli algoritmi supervisionati e quella dei non supervisionati. Vediamole ora con maggior dettaglio.

2.1.1 Apprendimento supervisionato

Gli algoritmi di *Machine Learning* supervisionati possono essere usati per quella che comunemente viene chiamata Classificazione. La peculiarità di questi algoritmi è che i dati sui quali vengono addestrati sono etichettati, cioè ogni elemento dell'insieme è abbinato alla sua classe di appartenenza. Il loro scopo sarà quindi quello di riuscire ad abbinare dei dati nuovi, non facenti parte dell'insieme di addestramento, alla propria classe di appartenenza. Capiamo meglio con un esempio.

Supponiamo di avere a disposizione un insieme di dati riguardanti delle *email*. A ciascuna di queste *email* sarà associata un'etichetta che ci dice se questa è *spam* o no. Un algoritmo di classificazione, dopo aver elaborato queste *email* ed averne studiato le caratteristiche, sarà in grado di estrapolarne un modello, grazie al quale sarà possibile dare un'etichetta "*spam*" o "non *spam*" ad una *email* nuova, non facente parte dell'insieme sul quale l'algoritmo ha costruito il suo modello. (vedi Figura 2.1)

2.1.2 Apprendimento non supervisionato

Come dicevamo in precedenza, l'altra grande famiglia di algoritmi facenti parte del *Machine Learning* è quella che fa uso di apprendimento non supervisionato: stiamo parlando degli algoritmi di *clustering*.

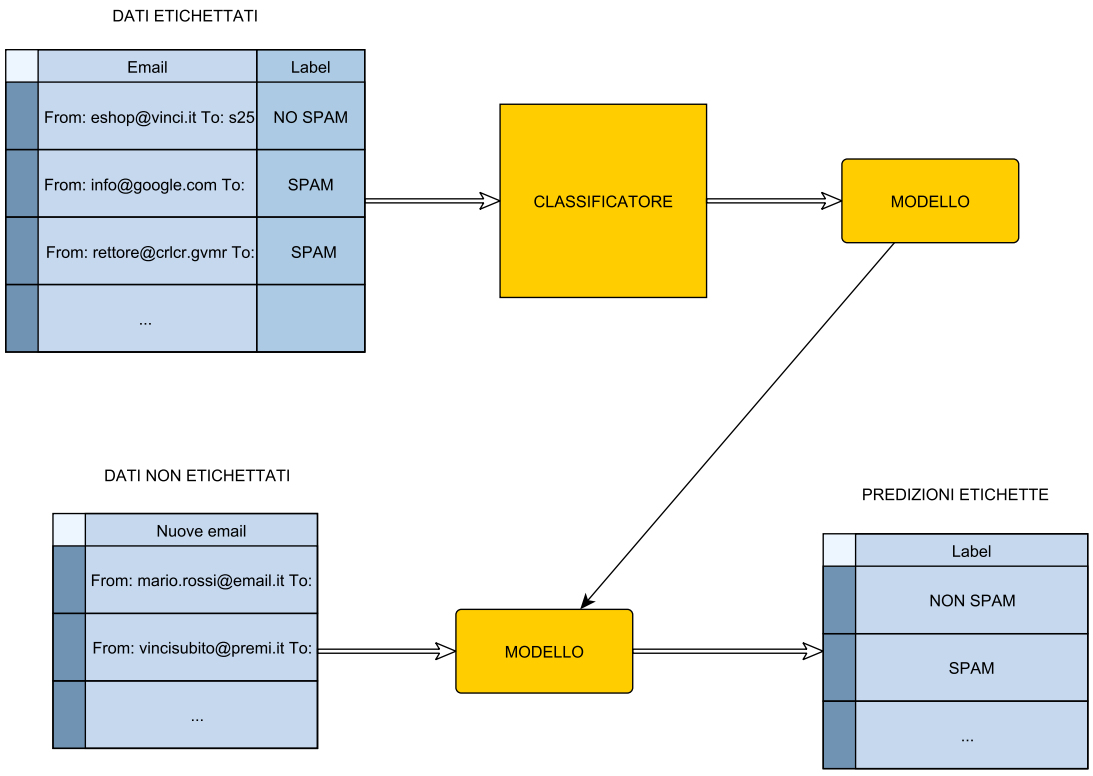


Figura 2.1: Esempio di classificazione

La differenza principale con gli algoritmi supervisionati è che questi lavorano sui dati, ma senza conoscerne la classe di appartenenza. Tutte le tecniche di *clustering* si basano su delle misurazioni di distanza tra tutti gli elementi facenti parte del *dataset* da analizzare, cioè dell'insieme di dati. Gli elementi che risulteranno essere più simili fra loro finiranno nello stesso *cluster*, cioè nello stesso gruppo, mentre gli elementi meno simili finiranno in *cluster* diversi.

Le tecniche di *clustering* possono essere usate principalmente con due diverse finalità. La prima è di comprensione: attraverso l'osservazione dei *cluster* che si formano a partire da un *dataset* è possibile risalire a eventuali relazioni e schemi fra i dati stessi. Potremmo ad esempio pensare di avere a

disposizione una collezione di dati riguardante dei geni e trovare dei *cluster* che li dividano in geni che abbiano una funzione simile; oppure potremmo dividere un elenco di clienti in possibili gruppi accomunati dagli stessi interessi per avviare campagne di *marketing ad hoc*.

L'altro fine per il quale il *clustering* può essere usato è la riduzione di un *dataset* con un numero di elementi troppo elevato ad uno più piccolo. Questo potrebbe essere necessario, per esempio, per addestrare più velocemente un algoritmo di classificazione che disponeva di un insieme di addestramento troppo grande. Applicando un algoritmo di *clustering* sarà possibile infatti trovare quali sono i suoi *cluster* principali, ovvero quei sottoinsiemi di dati simili fra loro, prendendone tra questi solo alcuni, che siano i più rappresentativi per ogni *cluster*.

2.2 *Pre-processing*

Fino ad adesso, abbiamo sempre supposto di lavorare su delle collezioni di dati che fossero "pulite", nelle quali i dati rappresentassero perfettamente gli oggetti dell'ambito in questione, e che i dati fossero completi e ben formati, senza lacune di alcun tipo. Risulta facile capire che questa situazione non si verifica quasi mai nella realtà. Prima di iniziare a procedere con una qualsiasi tecnica di *Machine Learning* è quindi necessario dedicare del tempo al cosiddetto *pre-processing*, ovvero ad un insieme di tecniche il cui scopo è eliminare o correggere le imperfezioni dai dati. Saltando questa fase, i risultati ottenuti perderebbero parecchio in termini di affidabilità e precisione; immaginiamo, ad esempio, un classificatore che non fosse in grado di predire correttamente la classe della maggior parte dei dati, oppure un *clustering* nel quale i *cluster* ottenuti non rispecchiassero bene i *cluster* reali.

I dati potrebbero avere diversi tipi di difetti. Potrebbero essere affetti da rumore, per il quale i dati reali risulterebbero sfalsati. Potrebbero essere presenti dei dati "disturbatori", molto diversi da quelli presenti in tutti gli altri elementi nella collezione dati, comunemente chiamati *outlier*. Potrebbero esserci dati mancanti e via così...

Vediamo ora quali sono i passi da compiere per rendere una collezione di dati adatta all'applicazione di tecniche supervisionate e non.

- Campionamento: lo scopo di questo passo è la riduzione della cardinalità della collezione di dati. Esistono vari modi per estrarre dei campioni. Si può estrarre senza rimpiazzo, con rimpiazzo, oppure in maniera stratificata, facendo così in modo da mantenere nell'insieme di dati campionati la stessa distribuzione dell'insieme di partenza. Si ricorre a queste tecniche quando l'elaborazione dei dati risulta troppo onerosa in termini di tempo.

- Riduzione della dimensionalità: certe collezioni di dati presentano un problema conosciuto come la "maledizione della dimensionalità" (*curse of dimensionality*); questo consiste, da un alto, nel fatto che gli elementi che costituiscono una collezione di dati possono avere troppi attributi, quindi troppe dimensioni, e rendere la computazione davvero onerosa; dall'altro lato, avendo troppe dimensioni, i valori delle distanze che intercorrono tra i dati potrebbero risultare molto simili, rendendo così difficile capire quali elementi davvero siano vicini e quali lontani. A tal problema si può porre rimedio ricorrendo a tecniche che hanno il fine di ridurre la dimensionalità dei dati, ma mantenendone sempre l'informazione. Tra queste citiamo la *Principal Component Analysis*, che vedremo

anche più avanti nel capitolo 4, la quale serve ad esprimere gli elementi in funzione di vettori facenti parte di un altro spazio geometrico, al fine di concentrare tutta l'informazione su un numero inferiore di dimensioni.

- Selezione di un sottoinsieme di *feature*: questo è un altro modo per ridurre la dimensionalità dei dati. Consiste nell'eliminare gli attributi ridondanti, la cui informazione risulta già espressa in altri attributi, o nell'eliminazione di attributi inutili ai fini delle analisi che si stanno conducendo (ad esempio, se stessimo studiando la distribuzione delle persone in fasce d'età nei vari quartieri di una città, sarebbe inutile conoscere nome e cognome di ciascuna).

- Aggregazione: consiste nell'aggregare due o più attributi (o elementi) in uno solo; questa tecnica può essere utile per ridurre il volume dei dati, quando questi fossero tanto numerosi da rendere particolarmente lunghi i tempi di analisi (un esempio potrebbe essere l'aggregazione di varie città in regioni, stati, nazioni...).

- Creazione delle *feature*: consiste nelle creazione di nuovi attributi che siano in grado di catturare le informazioni più importanti in maniera più efficiente della forma originale. Si possono mappare i dati in un nuovo spazio o si possono costruire delle *feature* che siano una combinazione di altre.

- Discretizzazione: consiste nella divisione del dominio di variazione di attributi di tipo continuo in fasce, al fine di ridurre la cardinalità dei possibili valori che può assumere un certo attributo.

- Trasformazione degli attributi: consiste nella trasformazione di tutti gli attributi. Dal momento che la maggior parte delle tecniche di analisi dei dati si basa su distanze geometriche tra i dati, ci potrebbero essere dei casi in cui degli attributi avessero dei *range* di variazione più ampio rispetto ad altri attributi. Ciò peserebbe nei risultati a favore di quegli attributi che assumono valori maggiori. Per porre rimedio a queste situazioni si può ricorrere quindi a tecniche per il riscalamento dei dati, quali normalizzazione o standardizzazione.

  La normalizzazione consiste nel riscalare tutti i dati in modo che il loro valore oscilli fra 0 e 1.

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

  La standardizzazione permette invece di ridistribuire tutti i dati secondo una distribuzione di Gauss con media $\mu = 0$ e deviazione standard $\sigma = 1$

$$X_{standardized} = \frac{X - \mu}{\sigma} \tag{2.1}$$

Capitolo 3

Clustering

In questo capitolo approfondiremo con maggior dettaglio il *clustering*. Vedremo dapprima quali tipi di *cluster* si possono ottenere e in che modo si possono classificare i diversi tipi di algoritmi di *clustering*. Poi vedremo tre algoritmi di base (K-Means, *Clustering* gerarchico e DBSCAN), ognuno dei quali è rappresentativo di una categoria di algoritmi e concluderemo con una versione più avanzata di K-Means. Infine, concluderemo con MPCK-Means, un algoritmo di *clustering* semi-supervisionato, basato su K-Means.

3.1 Considerazioni preliminari

3.1.1 Tipi di *cluster*

Come avremo modo di vedere nel corso di questo capitolo, non tutti gli algoritmi di *clustering* sono adatti a lavorare con tutti i tipi di dati. Alcuni algoritmi si comporteranno meglio in certe condizioni, altri meglio in altre. Per saper scegliere l'algoritmo migliore sarà quindi necessario conoscere quali tipi di *cluster* potremo incontrare. Vediamo ora quali sono i diversi tipi, aiutandoci a visualizzarli in un caso bidimensionale in Figura 3.1.

- *Cluster* ben separati (vedi Figura 3.1a): gli elementi facenti parte dello stesso *cluster* sono quelli più simili fra loro rispetto a tutti gli altri elementi presenti negli altri *cluster*; in un caso ideale di questa definizione, i diversi *cluster* sono ben lontani fra loro. Questo genere di *cluster* ha necessariamente una forma globulare.

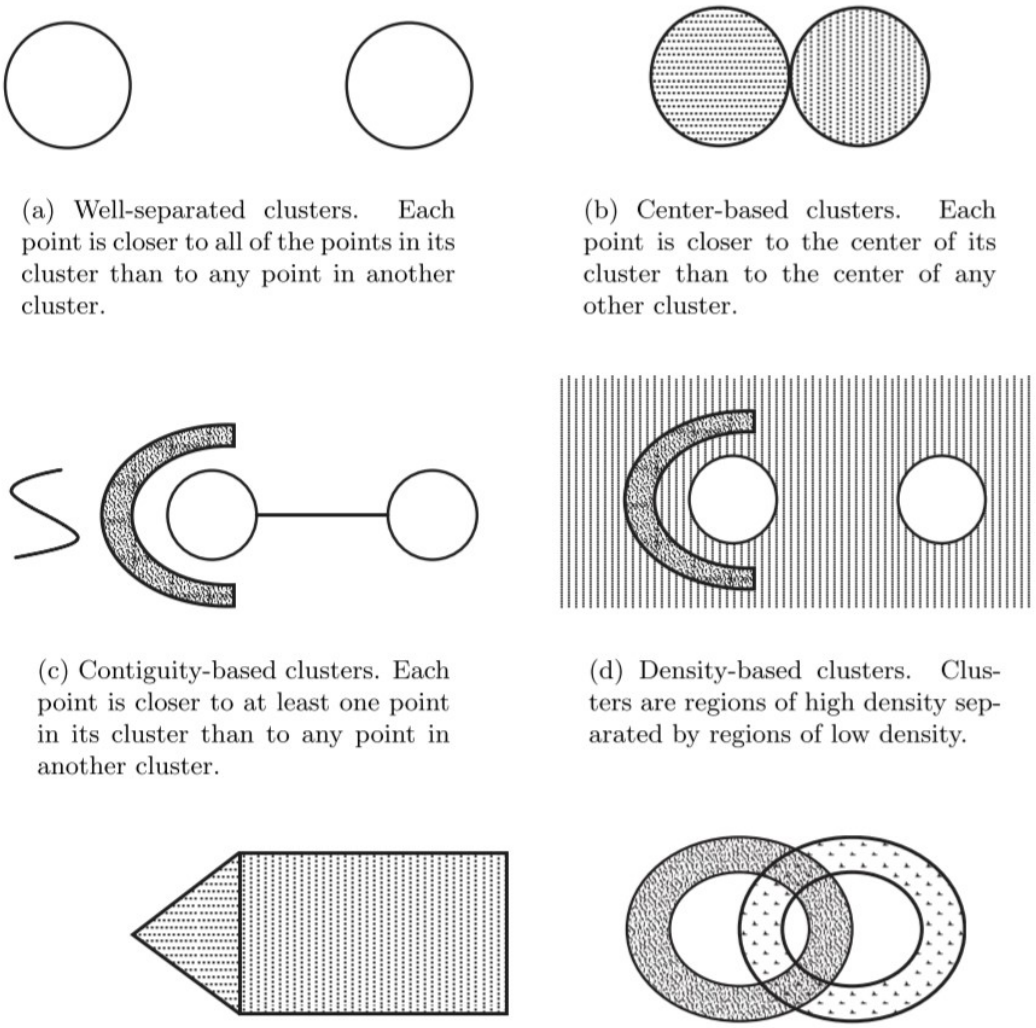
- *Cluster* basati sui prototipi (vedi Figura 3.1b): in questo caso, un *cluster* è un insieme di oggetti che sono molto vicini ad un elemento esemplare, che rappresenta al meglio una certa categoria. Questi *cluster* si chiamano anche *cluster center-based* (cioè basati sul loro elemento più rappresentativo). L'elemento più rappresentativo della collezione di dati prende il nome di medoide. Se l'elemento più rappresentativo non fa parte della collezione, si chiama invece centroide.

- *Cluster* basati su grafi: quando i dati sono rappresentati da un grafo, nel quale i nodi sono gli elementi e i rami rappresentano i collegamenti logici tra gli elementi, allora un *cluster* può essere considerato una componente connessa. Un esempio di questo genere di *cluster* sono i *cluster* basati sulla contiguità, nei quali due elementi sono connessi solo se tra i due sono separati da una certa distanza massima. In questi casi, come è possibile osservare anche in Figura 3.1c, il rumore può generare problemi, fungendo da ponte tra due *cluster* separati che vengono così visti dall'algoritmo come un unico *cluster*

- *Cluster* basati sulla densità: un *cluster* è una regione densa di elementi circondata da una regione a densità inferiore (vedi Figura 3.1d). Sfruttando questa definizione di *cluster*, laddove i *cluster* basati sui grafi non

dividano bene come in Figura 3.1c, i *cluster* risulteranno ben separati, perché il ponte che collega i *cluster* verrà considerato rumore.

- *Cluster* concettuali: la definizione di questi *cluster* potrebbe inglobare tutte le precedenti. Tutti gli elementi appartenenti a questa categoria soddisfano una o più proprietà insieme (vedi Figura 3.1e)



(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.

(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.

(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.

(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.

(e) Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)

Figura 3.1: Tipi di *cluster*
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

3.1.2 Classificazione degli algoritmi

Possiamo distinguere innanzitutto due famiglie di algoritmi di *clustering*:

- metodi aggregativi o *bottom-up*: secondo questo modo di fare *clustering*, inizialmente tutti gli elementi del *dataset* vengono visti come dei *cluster* a sé stanti. Si procede poi con l'unire in un solo *cluster* gli elementi più vicini fra loro, aggregando così *cluster* più piccoli in *cluster* più grandi (vedi Figura 3.2). Tutto questo fino a raggiungere una condizione prefissata, che potrebbe essere il numero di *cluster*, una distanza minima fra i *cluster* o altro, a seconda dell'algoritmo utilizzato.

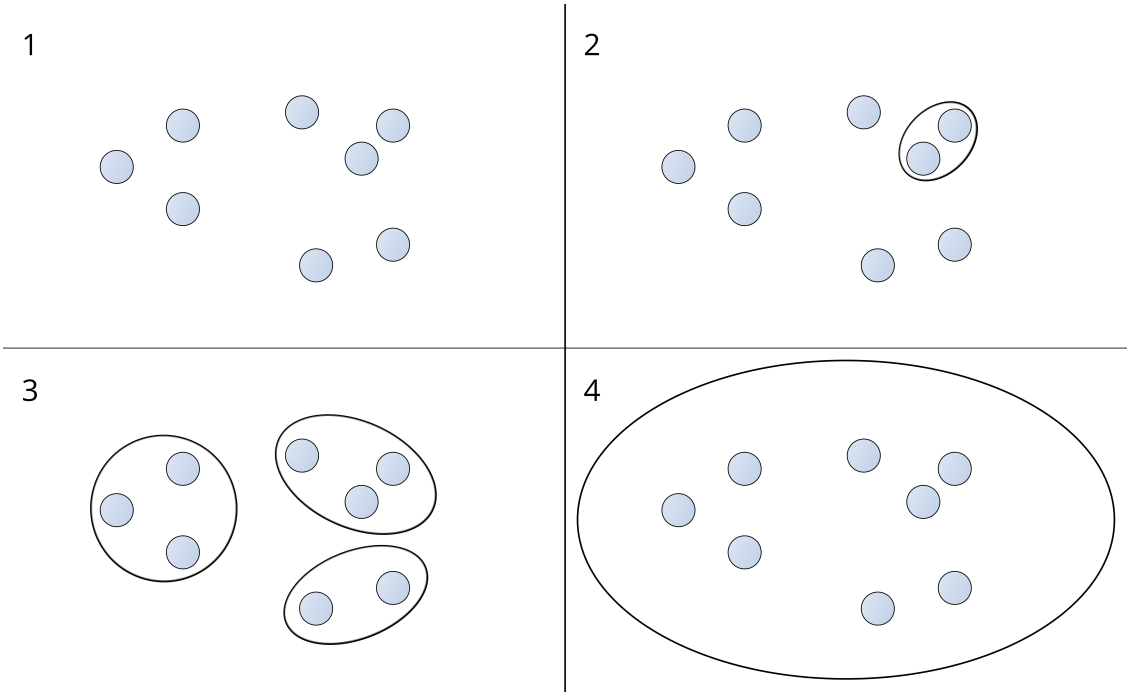


Figura 3.2: Esempio di *clustering* aggregativo

- metodi divisivi o *top-down*: secondo quest'altro modo di fare *clustering*, si parte da un unico grande *cluster* che contiene tutti gli elementi. Progredendo con l'algoritmo, il *cluster* viene diviso in *cluster* sempre più

piccoli. Si procede fino al raggiungimento di una condizione d'arresto, in genere il numero di *cluster* desiderato.

Esistono anche altre suddivisioni degli algoritmi di *clustering*. Una di queste vede contrapporsi il *clustering* esclusivo a quello non esclusivo (sebbene in questa tesi ci concentreremo solo sul *clustering* esclusivo):

- *clustering* esclusivo: in questo tipo di *clustering* un elemento può appartenere ad uno e un solo *cluster*.

- *clustering* non esclusivo (conosciuto anche come *fuzzy clustering*): qui un elemento può appartenere a più *cluster* contemporaneamente, ad ognuno secondo una probabilità p. Ovviamente, la somma delle probabilità di appartenenza di un elemento ai vari *cluster* dev'essere 1.

3.1.3 *Clustering* semi-supervisionato

Gli algoritmi di *clustering* semi-supervisionati hanno sempre lo stesso obiettivo degli altri, cioè di dividere in gruppi i dati a seconda delle loro somiglianze, ma a differenza di questi possono sfruttare, se disponibili, delle informazioni aggiuntive su una piccola porzione di dati per ottenere risultati migliori. Queste informazioni possono essere espresse sotto forma di etichetta di classe, cioè certi dati sono fin dal principio destinati ad entrare in un preciso *cluster*, oppure sotto forma di vincoli a coppie. Nello specifico si avranno due liste di vincoli: i vincoli sui dati che devono finire nello stesso *cluster*, chiamati anche *must-link*, e i vincoli che dicono quali elementi non devono finire insieme, chiamati *cannot-link*. Quando si usa un algoritmo di *clustering standard*, i *cluster* generati si basano solo sulla distribuzione dei dati nello

spazio. Grazie all'uso degli algoritmi semi-supervisionati si può raffinare il raggruppamento mediante regole semantiche, anziché topologiche.

3.1.4 Metriche e *linkage*

Per comprendere meglio la descrizione degli algoritmi dei prossimi paragrafi, introduciamo in questa sezione i concetti di metrica, che serve per misurare la distanza tra due elementi, e *linkage*, ovvero una funzione che definisce cosa sia la distanza tra due *cluster*, tenendo conto di tutte le distanze tra le coppie di elementi facenti parte di *cluster* diversi.

In generale, una metrica tra due oggetti x_i e x_j, descritti ognuno dal proprio vettore che ne contiene le componenti nelle varie dimensioni spaziali $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$ $x_j = (x_{j1}, x_{j2}, ..., x_{jp})$, deve soddisfare le condizioni di non negatività e di simmetria ($d(x_i, x_j) = d(x_j, x_i) \geq 0$, con $d(x_i, x_j) = 0$ se $x_i = x_j$) e infine deve soddisfare la disuguaglianza triangolare $d(x_i, x_j) \leq d(x_i, x_k) + d(x_j, x_k)$. Tra le metriche più conosciute possiamo citare:

- la norma L^1, conosciuta anche come distanza di *Manhattan*, definita come $d(x_i, x_j) = \|x_i - x_j\|_1 = \sum_{r=1}^{p} |x_{ir} - x_{jr}|$ (vedi Figura 3.3)
- la norma L^2, probabilmente la più famosa, conosciuta anche come distanza euclidea, definita come $d(x_i, x_j) = \|x_i - x_j\|_2 = \sqrt{\sum_{r=1}^{p} (x_{ir} - x_{jr})^2}$ (vedi Figura 3.4)
- la norma L^∞, definita come $d(x_i, x_j) = \|x_i - x_j\|_\infty = \max_{r=1,\cdots,p} \{|x_{ir} - y_{jr}|\}$; guardando l'esempio in Figura 3.3, la norma L^∞ tra il punto e l'origine degli assi sarebbe $\max\{5, 6\} = 6$

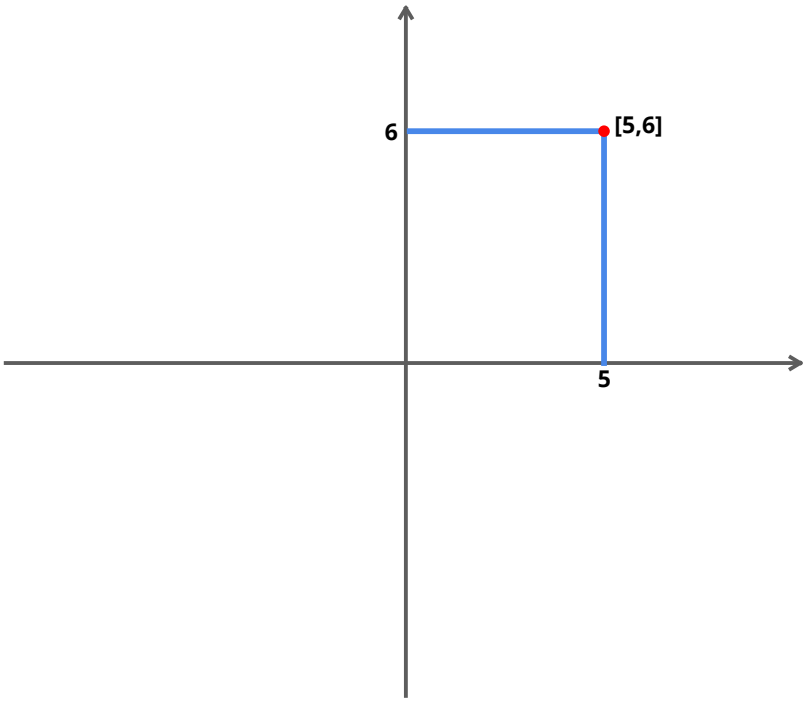


Figura 3.3: La distanza di *Manhattan* del punto dall'origine risulta essere $|5-0|+|6-0|=5+6=11$

Per citare alcuni tra i più semplici e comuni tipi di *linkage* diversi, abbiamo:

- il *complete-linkage* (vedi Figura 3.5), per il quale una misura tra due *cluster* è la massima distanza che intercorre tra un punto del primo e un punto del secondo ($d(A,B)=\max\{d(x,y):x\in A,y\in B\}$)
- il *single-linkage* (vedi Figura 3.6), per il quale la distanza tra due *cluster* è la distanza minima che intercorre tra un punto del primo e uno del secondo ($d(A,B)=\min\{d(x,y):x\in A,y\in B\}$)
- l'*average-linkage* (vedi Figura 3.7), dove la distanza tra due *cluster* è la distanza media che intercorre tra un generico punto del primo e un generico punto del secondo ($d(A,B)=\frac{1}{n_A n_B}\sum_{x\in A}\sum_{y\in B}d(x,y)$, dove n_j è il numero di elementi nel *cluster* j)

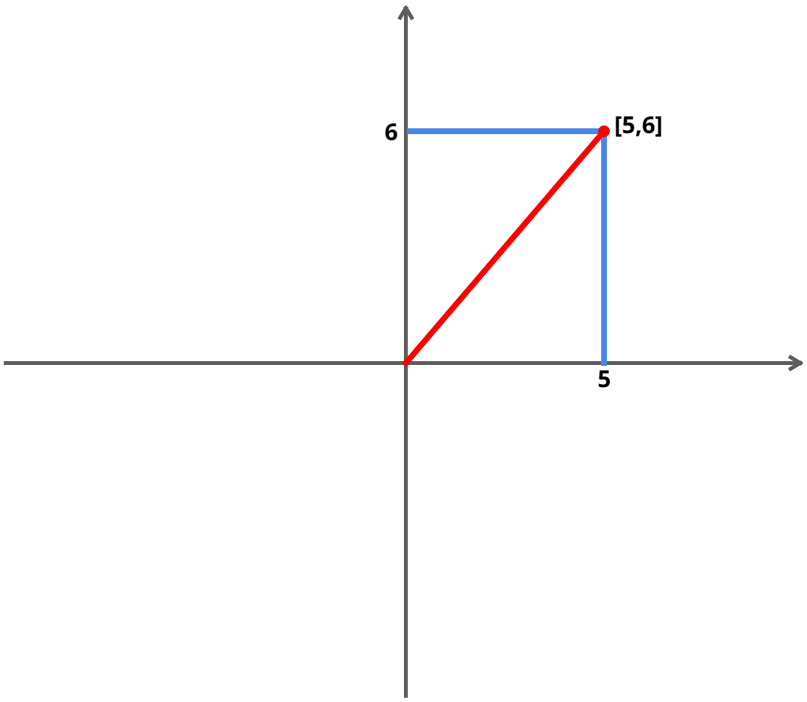


Figura 3.4: La distanza euclidea del punto dall'origine risulta essere $\sqrt{(5-0)^2+(6-0)^2} = \sqrt{25+36} = 7.8$

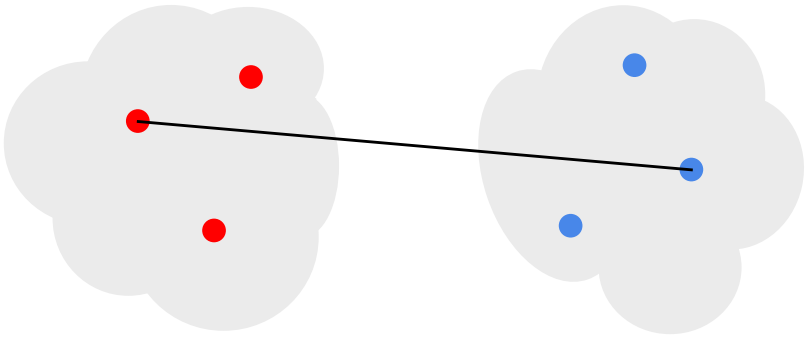


Figura 3.5: *complete-linkage*

- un altro modo di considerare la distanza tra due *cluster* è la distanza fra i loro centroidi ($d(A,B) = d((\frac{1}{n_A}\sum_{x\in A}\vec{x}),(\frac{1}{n_B}\sum_{x\in B}\vec{x}))$ dove n_j è il numero di elementi nel *cluster* j)

- infine, il metodo di Ward definisce la distanza tra due *cluster* A e B come l'incremento della somma degli errori quadratici se fondessimo insieme i

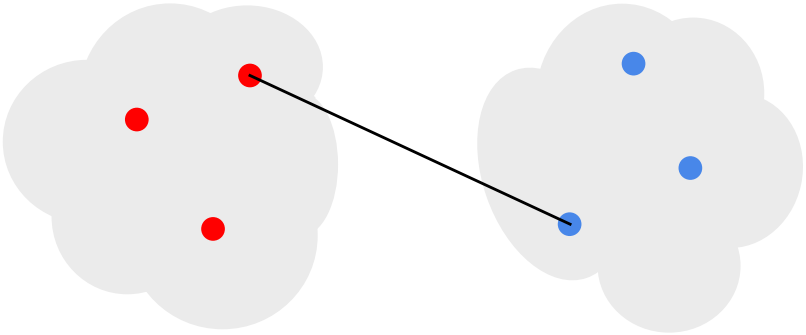


Figura 3.6: *single-linkage*

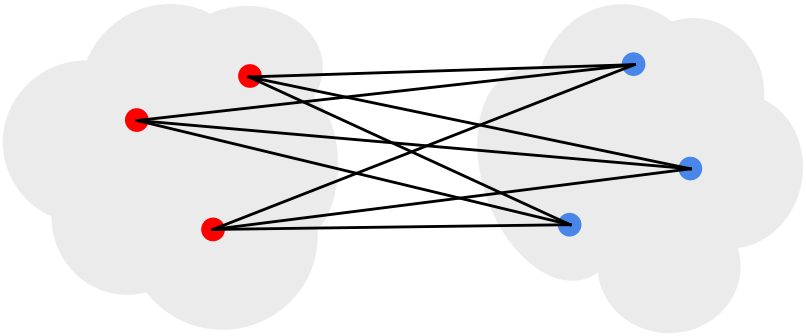


Figura 3.7: *average-linkage*

cluster A e B:

$$d(A,B) = \sum_{x \in A \cup B} \|\vec{x} - \vec{m}_{A \cup B}\|^2 - \sum_{x \in A} \|\vec{x} - \vec{m}_A\|^2 - \sum_{x \in B} \|\vec{x} - \vec{m}_B\|^2 = \\ = \frac{n_A n_B}{n_A + n_B} \|\vec{m}_A - \vec{m}_B\|^2$$

dove $\vec{m}_j$ è il centroide del *cluster* j, mentre n_j è il numero di elementi appartenenti al *cluster* j

A seconda del tipo di problema che si cercherà di risolvere, sarà più opportuno applicare un tipo di *linkage* diverso. Facciamo ora qualche considerazione prendendo in esame la Figura 3.8. Nel *single-linkage* possiamo notare come questo sia molto sensibile agli *outlier*: questi possono fare da "ponte" tra un *cluster* e l'altro, come si vede nel riquadro 1c). Dall'altro lato, il

single-linkage si comporta molto bene nei *cluster* che non hanno una forma sferica, come nel riquadro 1a) e 1b). Il *complete-linkage* è meno sensibile agli *outlier* del *single-linkage*, come possiamo vedere nel riquadro 3c). Non è però in grado di gestire al meglio *cluster* che non abbiano forma sferica, come nei riquadri 3a) e 3b). Anche quando i *cluster* sono sferici, ma sono sproporzionati fra loro, come in 3c), il comportamento tipico del *complete-linkage* è quello di spezzare i *cluster* più grossi. L'*average-linkage* costituisce un compromesso tra il *single-linkage* e il *complete-linkage*, riuscendo a gestire bene gli *outlier* come in 2c) e 2d), ma avendo anch'esso difficoltà con i *cluster* non globulari, come in 2a) e 2b). Il metodo di Ward non ottiene risultati migliori nei casi in cui i *cluster* non abbiano forma globulare, come in 4a) e 4b), ma quando la hanno riesce a gestire meglio degli altri la sproporzione fra i *cluster*, come in 4c). Anche gli *outlier* sono sopportati molto bene, come si vede in 4c) e 4d).

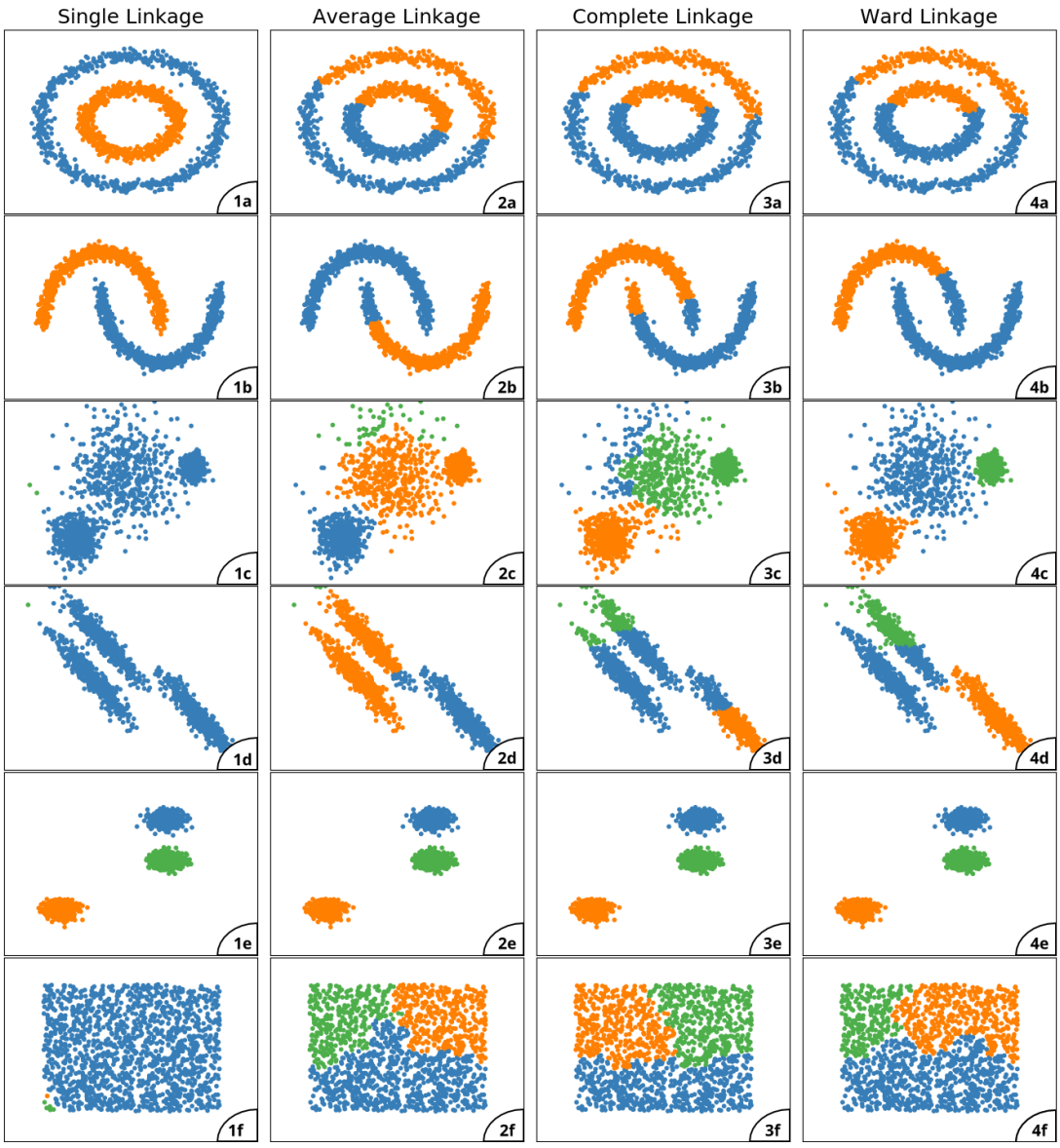


Figura 3.8: Tipi di *linkage* a confronto
[Fonte: `https://scikit-learn.org/stable/_images/sphx_glr_plot_linkage_comparison_001.png`]

3.2 *K-Means*

L'obiettivo del *K-Means* è quello di partizionare un insieme di n dati in $k \leq n$ *cluster* $S = \{S_1, S_2, S_3, \cdots, S_k\}$, minimizzando la varianza totale tra questi.

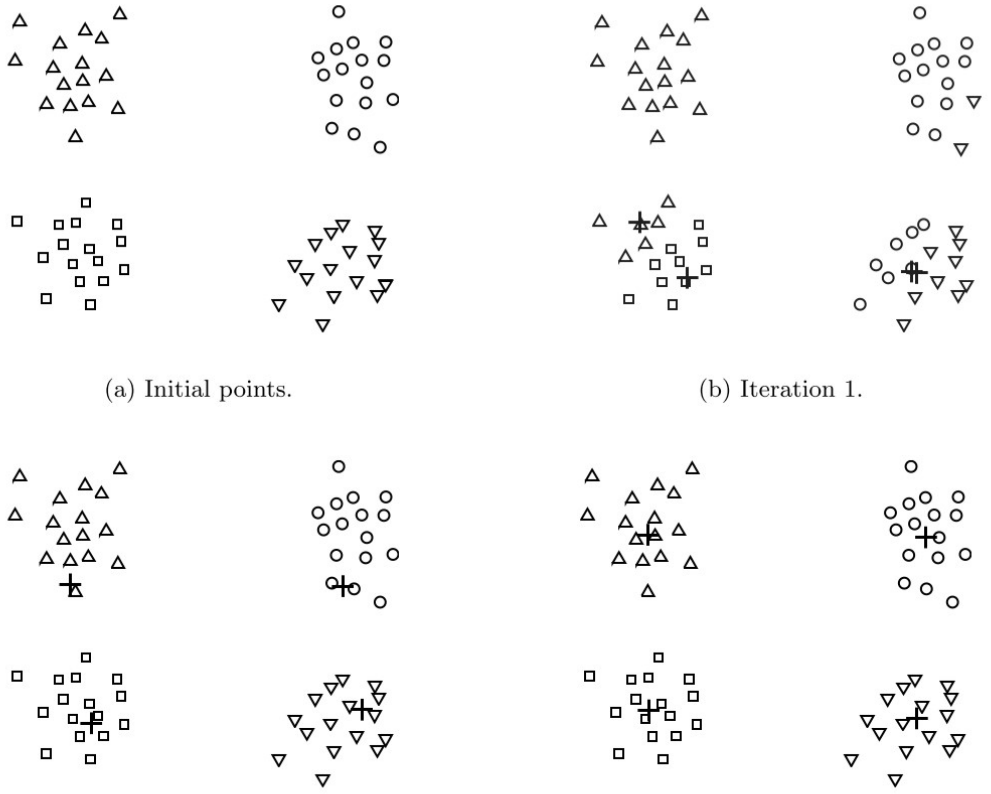
$$\arg\min_{S} \sum_{j=1}^{k} \sum_{x_i \in S_j} \|x_i - m_j\|^2$$

dove m_j rappresenta il centroide del j-esimo *cluster*, cioè un vettore di p componenti, ciascuna delle quali esprime la media delle componenti dei punti appartenenti al *cluster* j-esimo per ciascuna dimensione. La norma $\|x_i - m_j\|^2 = \sum_{r=1}^{p} (x_r - m_r)^2$ esprime la distanza euclidea tra ognuno dei p punti appartenenti al *cluster* S_j e il relativo centroide m_j. Il numero k di *cluster* che si desidera ottenere dev'essere stabilito a priori. I *cluster* raggruppano gli elementi in base a quanto sono simili.

Algoritmo 1 *K-Means*

Inizializzare K punti fissandoli come centroidi
ripetere
  Assegnare ogni punto al *cluster* col centroide più vicino
  Ricalcolare quali sono i centroidi di ogni *cluster*
finché i centroidi non cambiano più

Nella descrizione formale del *K-Means* (vedi Algoritmo 4) non viene indicato come i centroidi dei k *cluster* debbano essere inizializzati. Esistono varie strategie che si possono adottare. La più semplice consiste nello scegliere i k centroidi a caso tra i punti a disposizione. Questo comporta che, cambiando i centroidi in fase di inizializzazione, si possono ottenere potenzialmente risultati diversi, i quali sono tutti degli ottimi locali nel nostro problema di ottimizzazione (si vedano la Figura 3.9 e 3.10).

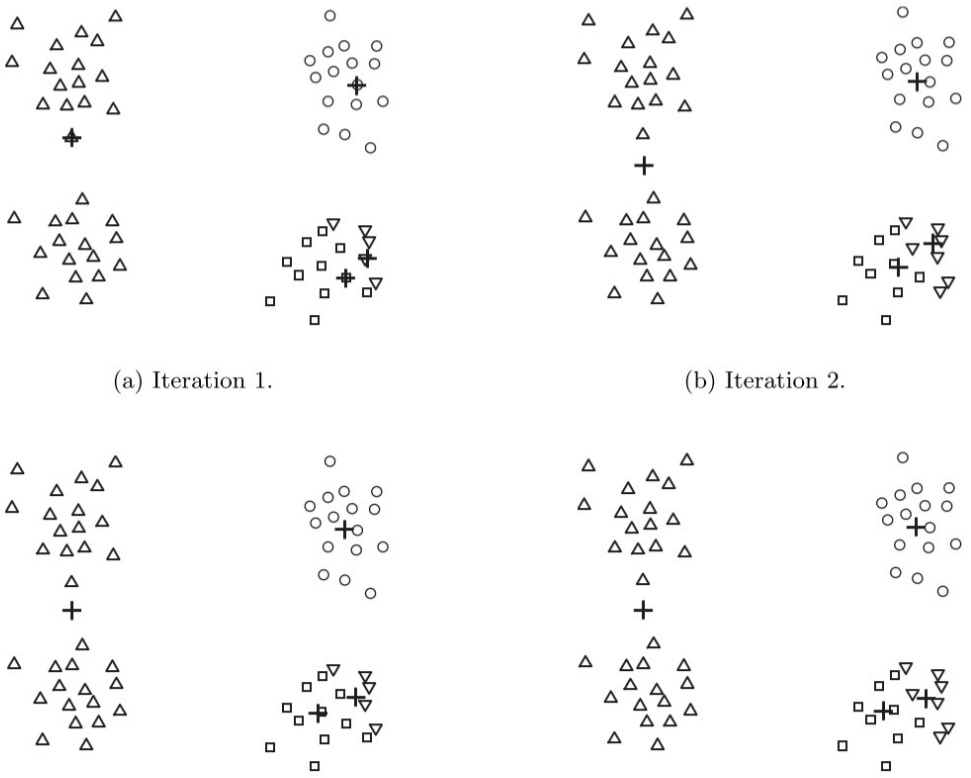


(a) Initial points. (b) Iteration 1.

(c) Iteration 2. (d) Iteration 3.

Figura 3.9: Caso in cui *K-Means* genera una soluzione ottima
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

La convergenza dell'algoritmo è assicurata dal fatto che, ad ognuno dei suoi passi, minimizza la quantità di varianza totale per ogni *cluster*; se questa non cambia più, invece, significa che si è arrivati ad un ottimo locale. Tuttavia, il problema della scelta dei centroidi iniziali non è affatto semplice. Una delle soluzioni più comuni, dal momento che l'algoritmo converge abbastanza rapidamente, è quella di eseguire più volte l'algoritmo e scegliere come soluzione quella per la quale la varianza totale dei *cluster* risulta essere



(a) Iteration 1. (b) Iteration 2.

(c) Iteration 3. (d) Iteration 4.

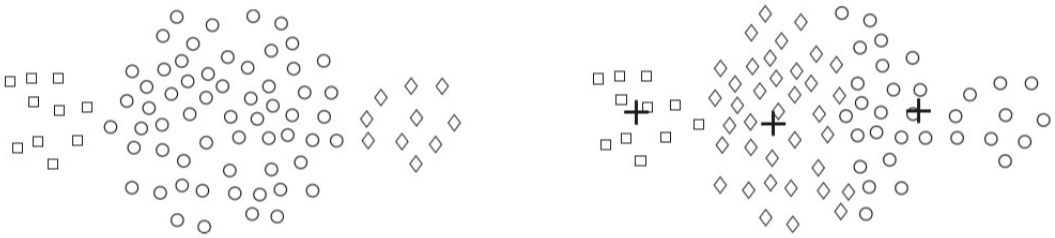
Figura 3.10: *Cluster* generati scegliendo i centroidi inziali non nel migliore dei modi
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

minore. In alternativa si può adottare una variante dell'algoritmo per la quale come primo passo viene assegnato a tutti i punti dell'insieme un *cluster* in maniera casuale. Da qui, vengono poi calcolati tutti i centroidi e si procede con le iterazioni normali dell'algoritmo. Un'altra soluzione, conosciuta come *farthest traversal*, prevede che si scelga il primo centroide c_1 casualmente; il secondo, c_2, sarà il punto più lontano da c_1; c_3 sarà il punto più lontano da $\{c_1, c_2\}$ e via di questo passo fino a quando non viene scelto anche c_k.

Purtroppo questo metodo è sensibile alla presenza di *outlier*.

In genere il *K-Means* non si comporta bene con alcuni tipi di problemi:

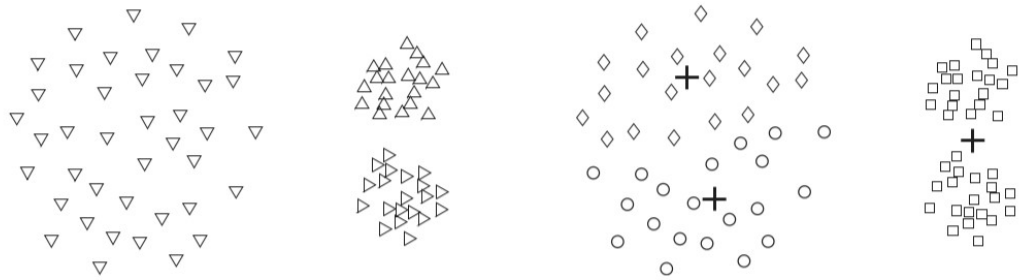
- quando i *cluster* hanno dimensioni molto diverse fra loro (vedi Figura 3.11)



(a) Original points. (b) Three K-means clusters.

Figura 3.11: *Cluster* con dimensioni diverse
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

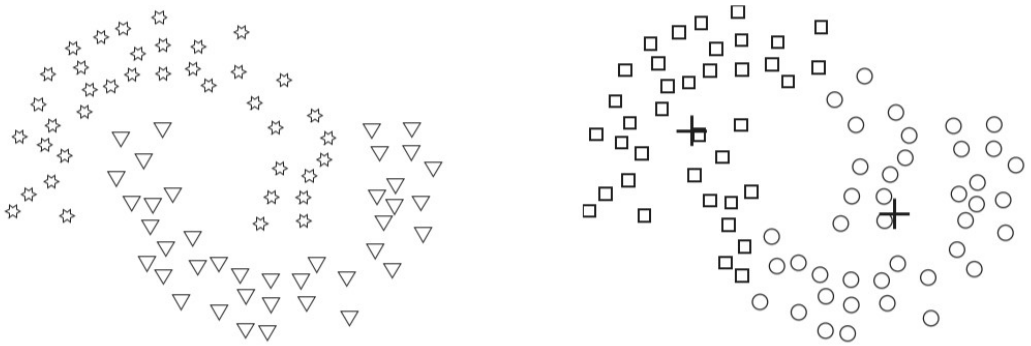
- quando i *cluster* hanno densità diverse (vedi Figura 3.12)



(a) Original points. (b) Three K-means clusters.

Figura 3.12: *Cluster* con densità diverse
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

- quando i *cluster* hanno forma non globulare (vedi Figura 3.13)

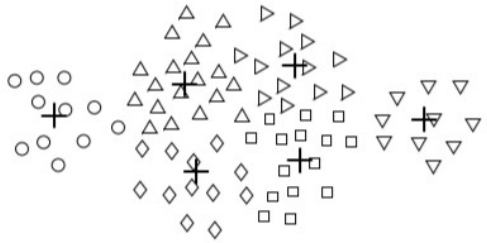


(a) Original points.

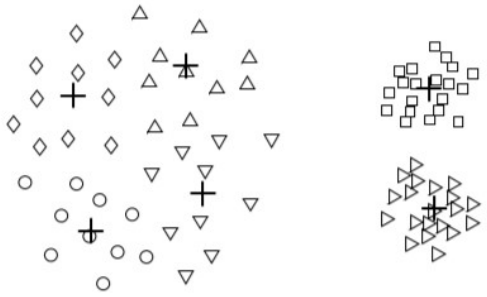
(b) Two K-means clusters.

Figura 3.13: *Cluster* con forma non globulare
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

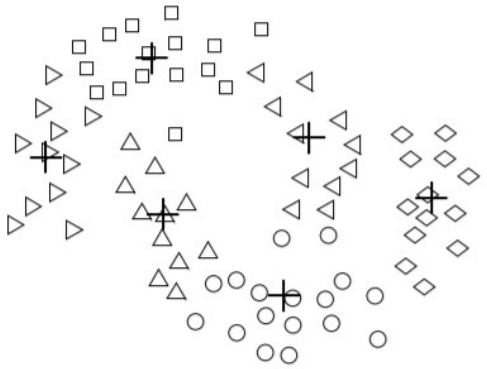
Tutti questi casi in cui *K-Means* non si comporta bene sono spesso risolvibili aumentando il numero di *cluster* k (vedi Figura 3.14). Si otterranno quindi più *cluster* di quelli che ci sarebbero stati realmente e se ne dovranno unire insieme alcuni con un'operazione di *post-processing*.



(a) Unequal sizes.



(b) Unequal densities.



(c) Non-spherical shapes.

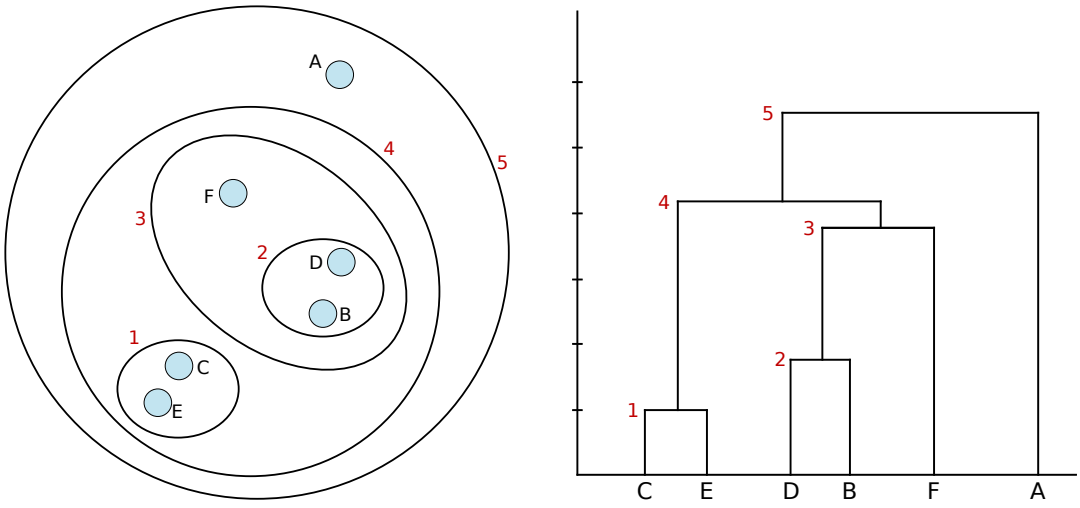
Figura 3.14: Se *K-Means* non riesce a trovare correttamente i *cluster* si può provare ad aumentare k e trovare così dei *sub-cluster* dei *cluster* reali
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

3.3 *Clustering* gerarchico

Uno dei maggiori svantaggi del *K-Means* è che bisogna specificare il numero K di *cluster* che si vogliono ottenere prima di applicare l'algoritmo. Il *clustering* gerarchico, a differenza di quest'ultimo, non richiede che il numero K di *cluster* venga deciso a priori. Un altro vantaggio è che un algoritmo di *clustering* gerarchico produce una rappresentazione ad albero, chiamata dendrogramma, che rende molto intuitiva la visualizzazione dei *cluster* formati (vedi Figura 3.15). A seconda dell'altezza alla quale si decide di tagliare il dendrogramma con una linea orizzontale, si otterranno un numero diverso di *cluster*. La scala riportata sull'asse y rappresenta la distanza che intercorre tra gli elementi o *cluster* affinché finiscano in uno stesso *cluster* più grande che li contenga tutti. Sempre osservando l'esempio in Figura 3.15, possiamo notare come i primi elementi ad essere uniti nello stesso *cluster* siano C ed E, gli elementi più vicini fra loro. Dire che C ed E sono i primi elementi a finire insieme, presuppone che si stia usando un algoritmo agglomerativo, ma è bene ricordare che un *clustering* gerarchico si può effettuare anche con un algoritmo divisivo, anche se i più usati sono i primi.

Generalmente gli algoritmi di *clustering* gerarchico fanno uso di una matrice delle distanze o di somiglianza per calcolare, ad ogni iterazione, la distanza che intercorre fra gli elementi e decidere quali elementi unire in un *cluster* o quale dividere ulteriormente (a seconda che venga usato un algoritmo agglomerativo o divisivo).

Troviamo di seguito (vedi Algoritmo 2) la descrizione formale di un generico algoritmo di *clustering* gerarchico agglomerativo.



(a) Collezione di dati (b) Dendogramma

Figura 3.15: Esempio di *clustering* gerarchico

Algoritmo 2 *Clustering* gerarchico

Calcolare la matrice di prossimità
Ogni elemento singolo diventa un *cluster*
ripetere
  Unire i due *cluster* più vicini
  Aggiornare la matrice di prossimità
finché resta un solo *cluster*

I vari passi dell'algoritmo operano su di una matrice di prossimità (o matrice delle distanze); si tratta di una matrice quadrata avente come numero di righe e di colonne il numero attuale di *cluster*. In corrispondenza dell'i-esima riga e della j-esima colonna sarà indicata la distanza fra l'elemento i e l'elemento j. È facile immaginare che questa matrice sarà simmetrica: la distanza dell'elemento i con l'elemento j sarà la stessa che c'è tra l'elemento j e l'elemento i. La diagonale invece sarà composta da zeri (la distanza di ogni elemento da se stesso). Certe varianti dell'algoritmo, anziché usare una

matrice delle distanze, fanno uso di una matrice di somiglianza, che esprime il concetto esattamente contrario: più un elemento sarà simile ad un altro, più il fattore di somiglianza che li lega si avvicinerà a 1. Sulla diagonale avremo quindi solo degli 1 (ogni elemento è uguale a se stesso).

Vediamo ora un esempio di applicazione dell'Algoritmo 2 ai dati rappresentati in Figura 3.15 con *single-linkage*. Nella Tabella 3.1 troveremo la matrice di prossimità che ad ogni *step* viene aggiornata, fondendo insieme i *cluster* aventi la coppia di punti più vicina (in accordo con il *single-linkage*).

Inizialmente, la matrice di prossimità è inizializzata considerando ogni elemento facente parte dell'insieme di dati come un *cluster* a sé stante: *{A}, {B}, {C}, {D}, {E}, {F}*. Man mano che gli elementi verranno accorpati nei *cluster*, vedremo anche le righe e le colonne della matrice diminuire. Inoltre, essendo la matrice di prossimità una matrice simmetrica, è possibile considerare solo la regione triangolare sotto (o sopra) la diagonale. Allo *step 0* abbiamo una matrice 6 x 6, riempita dalle sole distanze tra i punti, e i *cluster* che hanno la distanza minore risultano essere *{C}* ed *{E}*. La matrice di prossimità verrà così aggiornata, calcolando tutte le distanze del nuovo *cluster {C, E}* formatosi con gli altri. Dal momento che stiamo usando un *single-linkage*, calcoleremo tutte le nuove distanze così $\min(d(C,A), d(E,A)) = \min(14,15) = 14$, $\min(d(C,B), d(E,B)) = \min(6,8) = 6$, $\min(d(C,D), d(E,D)) = \min(8,9) = 8$, $\min(d(C,F), d(E,F)) = \min(7,9) = 7$. Abbiamo ottenuto così la matrice allo *step 1*. Individuata la nuova distanza minima, 3, tra i *cluster {B}* e *{D}*, potremo calcolare le nuove distanze tra il nuovo *cluster {B, D}* e gli altri a partire dalla matrice che abbiamo a disposizione dallo *step 0* $\min(d(\{B,D\},\{A\})) = \min(10,8) = 8$, $\min(d(\{B,D\},\{C,E\})) =$

step 0	**A**	**B**	**C**	**D**	**E**	**F**
A	0					
B	10	0				
C	14	6	0			
D	8	3	8	0		
E	15	8	**2**	9	0	
F	7	6	7	5	9	0

step 1	**A**	**B**	**(CE)**	**D**	**F**
A	0				
B	10	0			
(CE)	14	6	0		
D	8	**3**	8	0	
F	7	6	7	5	0

step 2	**A**	**(BD)**	**(CE)**	**F**
A	0			
(BD)	8	0		
(CE)	14	6	0	
F	7	**5**	7	0

step 3	**A**	**(BDF)**	**(CE)**
A	0		
(BDF)	7	0	
(CE)	14	**6**	0

step 4	**A**	**(BCDEF)**
A	0	
(BCDEF)	**7**	0

Tabella 3.1: Esempio di *clustering single-linkage*

$\min(6, 8) = 6$, $\min(d(\{B, D\}, \{F\})) = \min(6, 5) = 5$. Ottenuta così la matrice dello *step 2*, e trovata come minima distanza 5 tra i *cluster {B, D}* ed *{F}*, possiamo calcolare le distanze tra il nuovo *cluster {B, D, F}* e gli altri, osservando l'ultima matrice calcolata allo *step 2*. Avremo che: $\min(\{B, D, F\}, \{A\}) = \min(8, 7) = 7$, $\min(\{B, D, F\}, \{C, E\}) =$

$\min(6, 7) = 6$. I *cluster* più vicini ora risultano essere *{B, D, F}* e *{C, E}* a distanza 6. Calcoliamo le nuove distanze $\min(\{B, D, F, C, E\}, \{A\}) = \min(7, 14) = 7$. Ottenuta la matrice dello *step 4*, l'ultimo *step* consisterà nell'unione tra i *cluster {A}* e *{B, C, D, E, F}* che distano 7.

3.4 DBSCAN

DBSCAN è un algoritmo basato sulla densità dei punti. Per poterci addentrare nel suo funzionamento, occorre prima definire alcune categorie di punti di cui l'algoritmo fa uso. Per semplificare il discorso, immaginiamo che i punti di cui dover fare un *clustering* si trovino su un piano bidimensionale (vedi Figura 3.16).

- i punti *core* (punti centrali) sono tali se, tracciando un raggio di lunghezza *Eps* attorno a questi, il cerchio creatosi racchiude almeno altri *MinPts* punti. I punti all'interno del cerchio fanno parte del vicinato del punto *core*

- i punti *border* (punti di confine) non sono punti *core*, perché non hanno abbastanza punti nel loro vicinato, ma fanno parte del vicinato di altri punti *core*

- i punti *noise* (punti di rumore) sono quei punti che non sono né punti *core* né punti *border*

Ora che abbiamo definito cosa siano i punti di *core*, *border* e *noise*, possiamo descrivere il funzionamento dell'algoritmo. In maniera non formale, potremmo dire che ogni coppia di punti *core* che siano abbastanza vicini (a distanza minore o uguale di *Eps*) finiranno nello stesso *cluster*. Ogni punto *border* che sia abbastanza vicino ad un punto *core*, finirà nello stesso *cluster* del punto *core* (se sono vicini più punti *core* di diversi *cluster*, ne verrà scelto uno solo). Infine, tutti i punti *border* verranno scartati. Troviamo una descrizione formale, anche se semplificata, nell'Algoritmo 3.

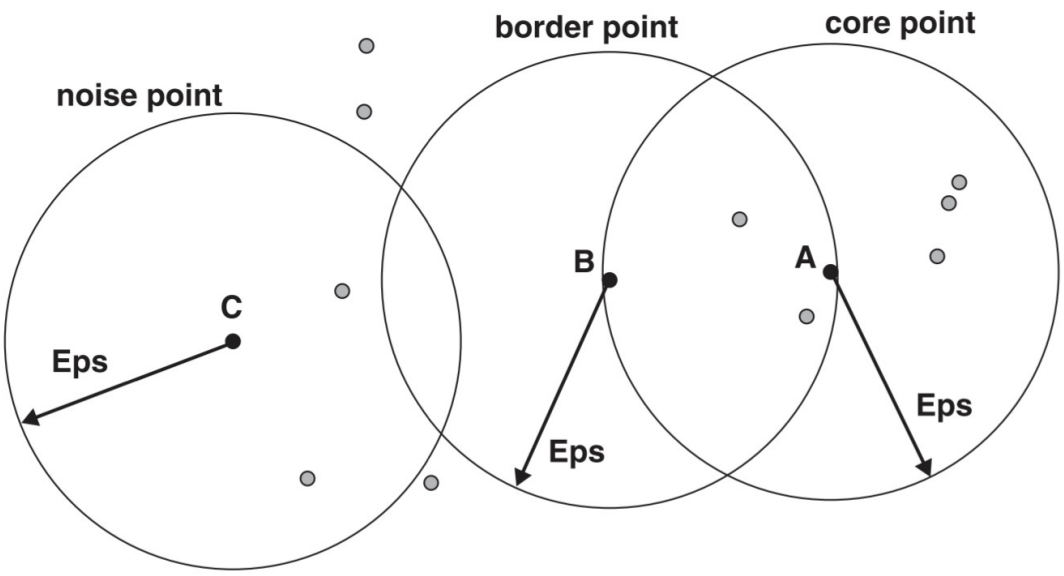


Figura 3.16: Classificazione dei punti nel DBSCAN ($MinPts = 4$)
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

Algoritmo 3 DBSCAN

Assegnare a tutti i punti un'etichetta tra *core*, *border* e *noise*
Eliminare i punti *noise*
Mettere un bordo tra tutti i punti *core* che si trovano all'interno di *Eps* l'uno dell'altro
Mettere ogni gruppo di punti *core* collegati in un *cluster* separato
Assegnare ogni punto *border* a uno tra i *cluster* dei punti *core* associati

Uno dei modi migliori per scegliere i parametri *MinPts* e *Eps* consiste nel misurare tutte le distanze tra ogni punto e il suo k-esimo vicino in ordine di distanza (per $k = MinPts$). Dopodiché si ordinano tali distanze. Otterremo un grafico come in Figura 3.17. Nella zona in cui il grafico presenterà un "gomito", ovvero dove ci sarà una brusca variazione della distanza del k-esimo vicino, è possibile scegliere un valore ragionevole per *Eps*. Tutti i punti

che avranno distanza minore del valore selezionato *Eps* saranno punti *core* e quelli che avranno distanza maggiore saranno punti *noise*. Se *MinPts* assume un valore troppo piccolo, una porzione indesiderata di *outlier* dei punti *noise* che sono abbastanza vicini tra loro verranno inseriti in un *cluster*. Se il valore di *MinPts* viene preso troppo grande, allora i *cluster* meno densi verranno etichettati come punti *noise*. In Figura 3.17 si è usato il valore 4 per *MinPts*. Se dovessero comparire più gomiti nel grafico, probabilmente significa che ci sono *cluster* a densità diverse: per trovare tutti i *cluster* allora sarà necessario applicare più volte DBSCAN con diversi valori dei suoi parametri.

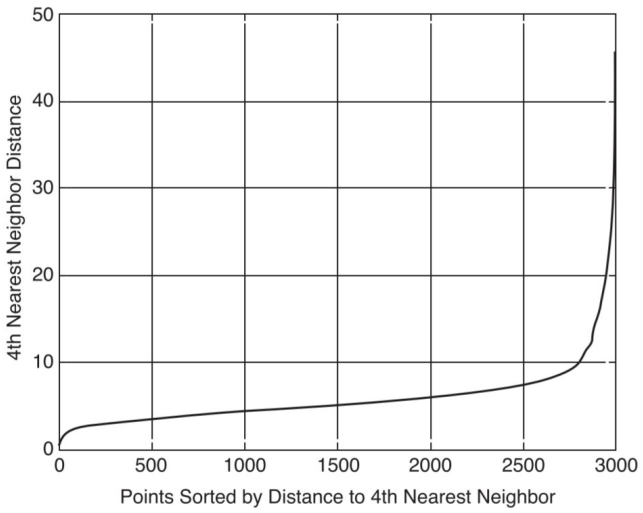


Figura 3.17: Grafico con distanze dal k-vicino ordinate ($k = 4$)
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

In Figura 3.18 possiamo osservare come si comporti DBSCAN nel trattare *cluster* di diversa forma e cardinalità. Anche gli *outliers*, che sono punti di rumore (punti *noise*), sono gestiti correttamente.

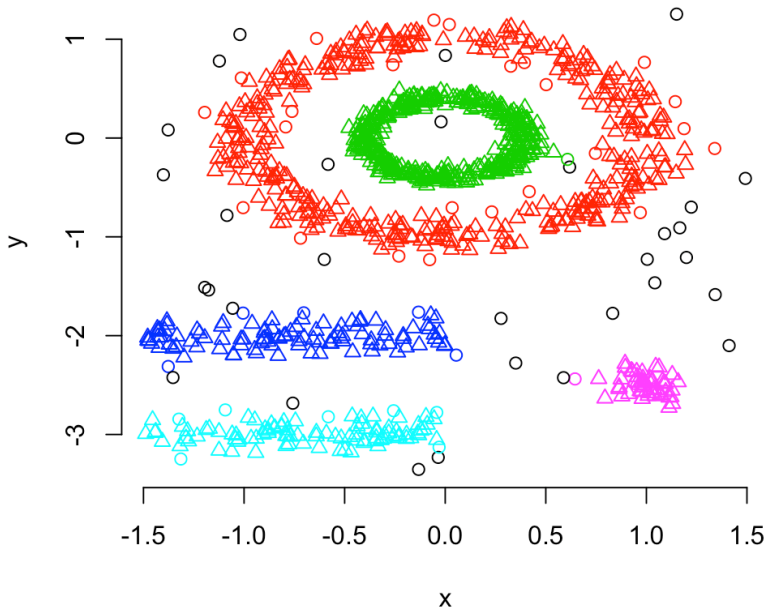


Figura 3.18: *Clustering* effettuato da DBSCAN
[Fonte: `http://www.sthda.com/sthda/RDoc/figure/clustering/dbscan-density-based-clustering-density-based-clustering-1.png`]

Nel caso in cui DBSCAN debba analizzare una collezione di dati nella quale sono presenti *cluster* a densità diversa, come nell'esempio in Figura 3.19, non riuscirà a trovare contemporaneamente tutti i *cluster*. Quando nell'esempio viene usato un *Eps* di 9.75 si può notare come i *cluster* piccoli siano tutti inglobati nel grande *cluster* giallo/verde. Diminuendo *Eps* a 9.62 i tre *cluster* piccoli e molto densi vengono invece correttamente individuati, ma i tre grandi *cluster* vengono considerati rumore. Si noti inoltre, come per una piccolissima variazione di *Eps* i risultati cambino in modo significativo.

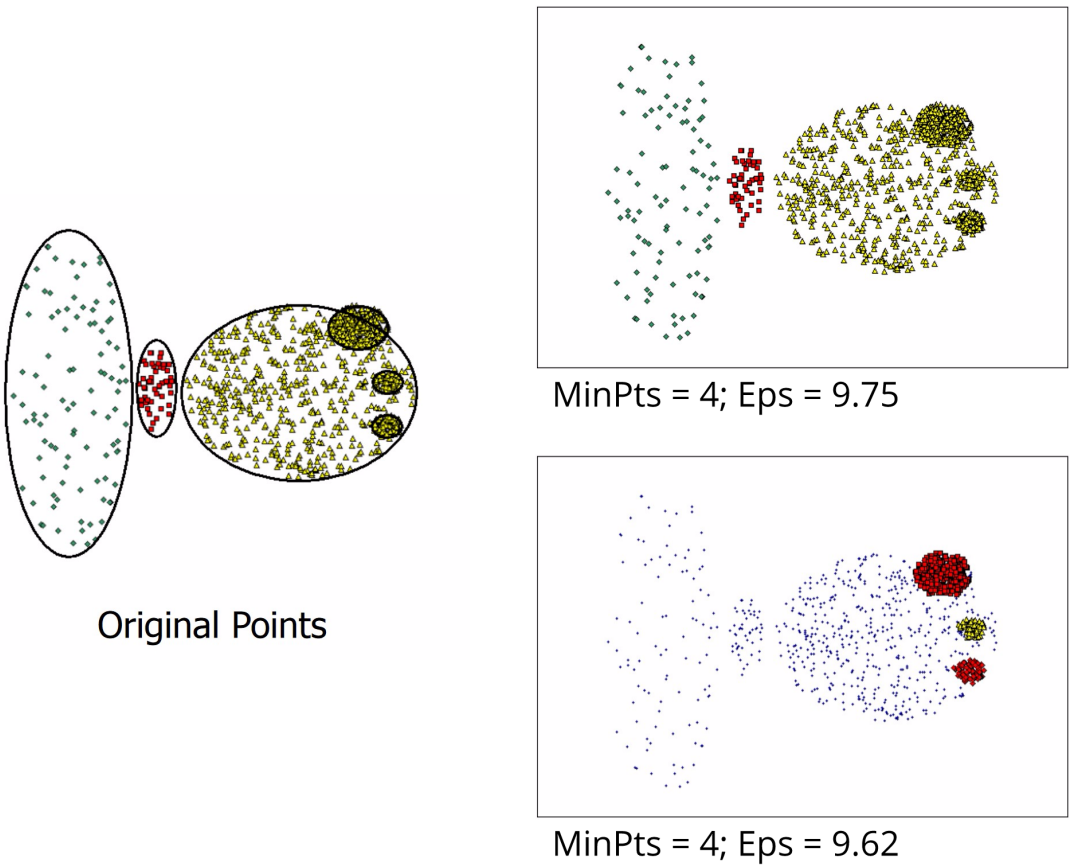


Figura 3.19: Per trovare *cluster* a diversa densità è necessario eseguire più volte il DBSCAN
[Tan, Steinbach, Kumar, *Introduction to Data Mining*, Pearson Education, 2006]

3.5 MPCK-Means

L'algoritmo di cui parleremo in questa sezione è una variante del K-Means, da cui si differenzia principalmente per un motivo, ovvero che non è non supervisionato come gli algoritmi che abbiamo visto finora, ma è semi-supervisionato. Capiamo subito cosa significa questo.

3.5.1 Integrazione di vincoli e metrica

Il funzionamento di MPCK-Means, come dicevamo prima, si basa su quello di K-Means: l'obiettivo è quello di raggruppare i dati in modo che la varianza totale all'interno di ogni *cluster* sia resa minima. L'informazione ausiliaria, che rende l'algoritmo semi-supervisionato, è espressa mediante vincoli a coppie. Per supportare i vincoli a coppie è necessario modificare la funzione obiettivo del K-Means, associando ad ogni vincolo non rispettato una penalità. Supponiamo, quindi, che $\mathcal{M}$ sia la lista dei vincoli *must-link*, cioè di tutte quelle coppie di punti che dovrebbero finire all'interno dello stesso *cluster*, due a due, e che $\mathcal{C}$ sia la lista dei *cannot-link*, cioè di tutte quelle coppie di punti che non dovrebbero essere nello stesso *cluster*. Le coppie $(x_i, x_j) \in \mathcal{M}$ e $(x_i, x_k) \in \mathcal{C}$ implicano quindi che x_i e x_j debbano finire nello stesso *cluster*, ma anche che x_i non debba essere nello stesso *cluster* di x_k. Siano anche $W = \{w_{ij}\}$ e $\overline{W} = \{\overline{w}_{ij}\}$ le penalità per aver infranto rispettivamente i vincoli in $\mathcal{M}$ e in $\mathcal{C}$. La funzione obiettivo da minimizzare sarebbe:

$$\mathcal{J}_{\text{pckmeans}} = \sum_{x_i \in \mathcal{X}} \|x_i - \mu_{l_i}\|^2 + \sum_{(x_i,x_j)\in\mathcal{M}} w_{ij} \mathbb{1}\left[l_i \neq l_j\right] + \sum_{(x_i,x_j)\in\mathcal{C}} \bar{w}_{ij} \mathbb{1}\left[l_i = l_j\right] \tag{3.1}$$

dove $\mathcal{X}$ è l'insieme di tutti i punti della collezione di dati, i punti x_i, x_j sono assegnati alle partizioni l_i, l_j con centroidi rispettivamente μ_{l_i}, μ_{l_j} e dove

$\mathbb{1}$ è una funzione indicatrice che vale 1 se il suo argomento è vero e 0 se è falso.

MPCK-Means è in grado anche di imparare una metrica per ogni *cluster*, cosicché ognuno possa avere una forma diversa. Per permettere questo è necessario adottare una matrice di pesi A_h per ogni *cluster* h diverso. La matrice è una parametrizzazione della distanza euclidea come segue: $|x_i - x_j|_{A_h} = \sqrt{(x_i - \mu_{l_i})^T A_h (x_i - \mu_{l_i})}$. Si può dimostrare che quanto detto equivale a minimizzare la seguente funzione obiettivo:

$$\mathcal{J}_{\text{mkmeans}} = \sum_{x_i \in \mathcal{X}} \left(\|x_i - \mu_{l_i}\|^2_{A_{l_i}} - \log\left(\det\left(A_{l_i}\right)\right) \right) \tag{3.2}$$

Combinando le equazioni (3.1) e (3.2) otteniamo la seguente funzione obiettivo:

$$\begin{aligned} \mathcal{J}_{\text{combined}} = & \sum_{x_i \in \mathcal{X}} \left(\|x_i - \mu_{l_i}\|^2_{A_{l_i}} - \log\left(\det\left(A_{l_i}\right)\right) \right) \\ & + \sum_{(x_i, x_j) \in \mathcal{M}} w_{ij} \mathbb{1}\left[l_i \neq l_j\right] \\ & + \sum_{(x_i, x_j) \in \mathcal{C}} \bar{w}_{ij} \mathbb{1}\left[l_i = l_j\right] \end{aligned} \tag{3.3}$$

Se assumiamo che i costi w_{ij} e $\overline{w}_{ij}$ siano uniformi, tutte le violazioni dei vincoli saranno trattate ugualmente. Ma la penalità per aver violato un *must-link* tra due punti lontani dovrebbe essere più alta che per la violazione di un *must-link* tra punti vicini. Questo intuitivamente perché, se due punti che dovrebbero essere insieme vengono considerati lontani per la metrica attuale, significa che la metrica è inadeguata e deve ancora subire cambiamenti importanti. Si può ottenere questo comportamento moltiplicando il secondo addendo dell'equazione (3.3) per la seguente penalità:

$$f_M\left(x_i, x_j\right) = \frac{1}{2} \|x_i - x_j\|^2_{A_{l_i}} + \frac{1}{2} \|x_i - x_j\|^2_{A_{l_j}} \tag{3.4}$$

Allo stesso modo, la penalità per la violazione di un *cannot-link* tra due punti vicini dovrebbe essere maggiore che per due punti lontani: significherebbe, altrimenti, che la metrica attuale non è in grado di considerare come distanti due punti che dovrebbero essere considerati tali. Si può ottenere questo comportamento moltiplicando il terzo addendo nell'equazione (3.3) per la seguente penalità:

$$f_C\left(x_i, x_j\right) = \left\|x'_{l_i} - x''_{l_i}\right\|^2_{A_{l_i}} - \left\|x_i - x_j\right\|^2_{A_{l_i}} \tag{3.5}$$

dove $\left(x'_{l_i}, x''_{l_i}\right)$ risulta essere la coppia di punti maggiormente separati secondo la l_i-esima metrica. Inoltre, questa forma di f_C assicura che la violazione di un vincolo *cannot-link* non sia negativa, dal momento che il secondo termine non è mai più grande del primo. Combinando le equazioni (3.3), (3.4) e (3.5) otteniamo la funzione obiettivo del MPCK-Means:

$$\begin{aligned}\mathcal{J}_{\text{mpckmeans}} = &\sum_{x_i \in \mathcal{X}} \left(\left\|x_i - \mu_{l_i}\right\|^2_{A_{l_i}} - \log\left(\det\left(A_{l_i}\right)\right)\right) \\ &+ \sum_{(x_i, x_j) \in \mathcal{M}} w_{ij} f_M\left(x_i, x_j\right) \mathbb{1}\left[l_i \neq l_j\right] \\ &+ \sum_{(x_i, x_j) \in \mathcal{C}} \overline{w}_{ij} f_C\left(x_i, x_j\right) \mathbb{1}\left[l_i = l_j\right]\end{aligned} \tag{3.6}$$

3.5.2 Funzionamento dell'algoritmo

Dato un insieme di punti $\mathcal{X}$, un insieme di vincoli *must-link* $\mathcal{M}$, un insieme di vincoli *cannot-link* $\mathcal{C}$, degli insiemi dei costi corrispondenti W e $\overline{W}$, e il numero di *cluster* desiderato K, l'algoritmo MPCK-Means (vedi Algoritmo 4) trova K partizioni disgiunte $\{\mathcal{X}_h\}_{h=1}^K$ di $\mathcal{X}$ tali che la funzione obiettivo (3.6) sia localmente minimizzata.

Algoritmo 4 *MPCK-Means*

Creare λ vicinati $\{N_p\}_{p=1}^{K}$ a partire da $\mathcal{M}$ e $\mathcal{C}$

se $\lambda \geq K$ **allora**

inizializzare i $\{\mu_h^{(0)}\}_{h=1}^{K}$ centroidi usando la tecnica *farthest-first traversal* iniziando dal vicinato N_p più grande

altrimenti se $\lambda < K$ **allora**

inizializzare i $\{\mu_h^{(0)}\}_{h=1}^{\lambda}$ con i centroidi di $\{N_p\}_{p=1}^{\lambda}$

inzializzare i restanti *cluster* a caso

fine se

ripetere

assegnare un punto x_i al *cluster* h^* (cioè all'insieme $\mathcal{X}_{h^*}^{(t+1)}$), per

$$\begin{aligned} h^* = \arg\min_h \left(\left\| x_i - \mu_h^{(t)} \right\|_{A_h}^2 - \log\left(\det\left(A_h\right)\right) \right) \\ + \sum_{(x_i,x_j)\in\mathcal{M}} w_{ij} f_M\left(x_i, x_j\right) \mathbb{1}\left[h \neq l_j\right] \\ + \sum_{(x_i,x_j)\in\mathcal{C}} \overline{w}_{ij} f_C\left(x_i, x_j\right) \mathbb{1}\left[h = l_j\right] \end{aligned}$$

calcolare tutti i nuovi centroidi: $\{\mu_h^{(t+1)}\}_{h=1}^{K} = \left\{ \frac{1}{\left|\mathcal{X}_h^{(t+1)}\right|} \sum_{x\in\mathcal{X}_h^{(t+1)}} x \right\}_{h=1}^{K}$

aggiornare le metriche:

$$\begin{aligned} A_h = \left|\mathcal{X}_h\right| \Big(\sum_{x_i\in\mathcal{X}_h} \left(x_i - \mu_h\right)\left(x_i - \mu_h\right)^T \\ + \sum_{(x_i,x_j)\in\mathcal{M}_h} \frac{1}{2} w_{ij} \left(x_i - x_j\right)\left(x_i - x_j\right)^T \mathbb{1}\left[l_i \neq l_j\right] \\ + \sum_{(x_i,x_j)\in\mathcal{M}_h} \frac{1}{2} w_{ij} \left(\left(x_h' - x_h''\right)\left(x_h' - x_h''\right)^T - \left(x_i - x_j\right)\left(x_i - x_j\right)^T \right) \mathbb{1}\left[l_i = l_j\right] \Big)^{-1} \end{aligned}$$

$t \leftarrow (t+1)$

finché c'è convergenza

Come per il K-Means, risulta molto importante una scelta corretta dei centroidi di inizializzazione. Nel caso dell'MPCK-Means, per la scelta dei centroidi iniziali si usano le informazioni racchiuse nei vincoli a coppie forniti. Si parte con l'analisi dei *must-link* e se ne cercano tutte le componenti connesse λ. Queste componenti connesse sono usate per creare λ vicinati $\{N_p\}_{p=1}^{\lambda}$, i quali sono formati da tutti i punti connessi insieme attraverso i *must-link*. Per ogni coppia di vicinati N_p e $N_{p'}$ per i quali esista almeno un vincolo *cannot-link* tra i due, andremo ad aggiungere il vincolo ad entrambi i vicinati. Se il numero di vicinati λ che avremo ottenuto dopo questa fase è maggiore o uguale a K, inizializzeremo i centroidi dell'algoritmo con i centroidi di K vicinati. Applicheremo la tecnica del *first farthest traversal* pesata: sceglieremo quindi K tra i λ centroidi dei vicinati in modo da massimizzare la distanza totale che intercorre tra questi; il peso di ciascun centroide sarà il numero di punti che appartengono ad ogni vicinato. Se invece $\lambda < K$, inizializzeremo i rimanenti $K - \lambda$ centroidi con delle perturbazioni del centroide di $\mathcal{X}$.

Dopo la fase di inizializzazione appena presa in esame, l'algoritmo MPCK-Means vede il susseguirsi di due diverse azioni, fino a quando non si raggiunge una condizione di arresto. Nella prima, ogni punto verrà assegnato ad uno dei vari *cluster* in modo da minimizzare la distanza dai centroidi, in accordo con la metrica di ciascun *cluster*. Subito dopo l'assegnazione, si verifica che non sia stato violato alcun vincolo. In questa fase un punto sarà spostato da un *cluster* ad un altro solo se il suo spostamento causa una diminuzione del valore della funzione obiettivo (vedi Equazione (4)). Nella seconda fase si ricalcoleranno i centroidi di tutti i *cluster*, ottenendo così una riduzione del valore della funzione obiettivo. Successivamente si ricalcoleranno le metriche

A_h di ogni *cluster*.

3.5.3 Comportamento dell'algoritmo

MPCK-Means, grazie al fatto che parametrizza una metrica diversa per ogni *cluster*, è in grado di fare il *clustering* anche di una collezione di dati nella quale i diversi *cluster* abbiano una diversa forma tra di loro, a patto che si abbiano a disposizione i vincoli a coppie nei quali è racchiusa questa informazione. Questo rappresenta un vantaggio rispetto alla versione originale di K-Means.

Nell'articolo in cui viene presentato MPCK-Means [5], si evidenzia anche un altro comportamento dell'algoritmo: al crescere del numero di vincoli che si hanno a disposizione, la qualità del *clustering* aumenta sempre più.

Nelle analisi sperimentali che saranno condotte nel Capitolo 6, però, solo una piccola porzione della collezione dati sarà usata per generare dei vincoli a coppie. Si procede in tal modo per mettersi il più possibile nelle condizioni reali, nelle quali non si hanno a disposizione delle etichette e bisogna far produrre queste manualmente a degli esperti di dominio. Sarà proprio in queste condizioni che AdaPro dimostrerà di ottenere risultati di qualità superiore, pur avendo a disposizione una piccolissima parte di prototipi per classe.

Capitolo 4

Strumenti di valutazione delle analisi *clustering*

Nel capitolo precedente abbiamo visto come funzionano alcuni tra i principali algoritmi di *clustering*, ma non ci siamo mai occupati degli strumenti matematici di cui disponiamo per valutarne le *performance*.

Possiamo distinguere principalmente due famiglie di misure di qualità dei *cluster* ottenuti:

- le misure interne: queste misurazioni avvengono senza conoscere le classi di appartenenza di ogni singolo dato. A loro volta si dividono in due ulteriori famiglie: quelle che misurano la coesione degli oggetti all'interno di uno stesso *cluster* e quelle che misurano la separazione tra i diversi *cluster*

- le misure esterne: queste misure hanno bisogno di informazioni esterne, come le etichette di classe di ogni singolo oggetto appartenente alla collezione di dati

4.1 Misure interne

Innanzitutto, definiamo meglio due concetti ai quali abbiamo già accennato all'inizio di questo capitolo e nel capitolo precedente. Si tratta della coesione dei dati all'interno di un *cluster* e della separazione che intercorre tra i *cluster* ottenuti.

La coesione indica quanto gli elementi all'interno di uno stesso *cluster* siano vicini. A seconda del tipo di *cluster* che stiamo considerando, è possibile calcolare la coesione in modi diversi. Nei *cluster center-based*, come quelli ottenuti da K-Means, possiamo misurarla con il SSE (*sum of squared error*) o WSS (*within cluster sum of squares*), cioè la somma delle distanze euclidee di ogni punto del *cluster* dal relativo centroide (vedi Figura 4.1(a)).

$$WSS = \sum_{i} \sum_{x \in \mathcal{C}_i} (x - m_i)^2$$

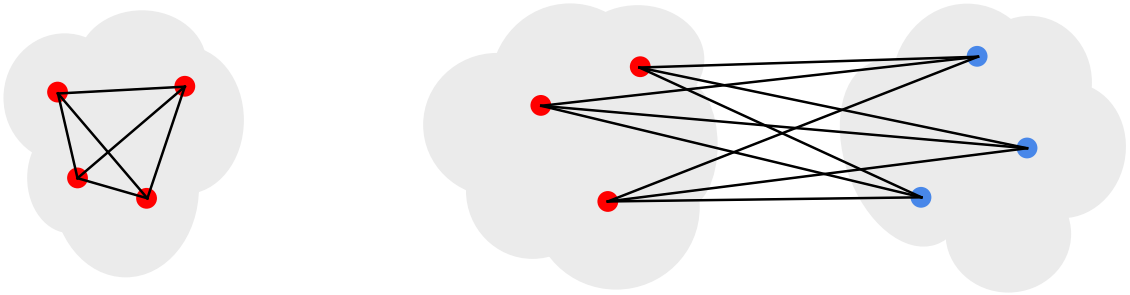
dove x rappresenta i punti appartenenti alla collezione di dati, $\mathcal{C}_i$ è il *cluster* di x e m_i è il centroide di $\mathcal{C}_i$.

La separazione indica quanto un *cluster* sia ben separato dagli altri, cioè quanta distanza intercorra mediamente tra un *cluster* e tutti gli altri (vedi Figura 4.1(b)). La separazione di un *cluster* $\mathcal{C}_i$ dagli altri è misurata tramite il BSS (*between cluster sum of squares*):

$$BSS_{\mathcal{C}_i} = \sum_{j} |\mathcal{C}_j| \, (m_i - m_j)^2$$

dove m_i, m_j sono i centroidi di $\mathcal{C}_i$ e $\mathcal{C}_j$, $|\mathcal{C}_i|$ è il numero di punti di $\mathcal{C}_i$.

Un *cluster* che abbia un livello di coesione alto può essere considerato migliore di un altro che abbia un valore inferiore. Si può utilizzare questa informazione per migliorare la qualità del *clustering*: se un *cluster* non risulta molto coeso, potremmo decidere di dividerlo in più *sub-cluster*. Se invece



a) coesione b) separazione

Figura 4.1

avessimo due *cluster* molto coesi, ma poco separati, potremmo optare per fonderli in un solo *cluster*.

4.1.1 *Silhouette*

La *silhouette* è una misura che combina insieme i concetti di coesione e separazione. Per ogni oggetto i, $a(i)$ è la distanza media di i con tutti gli altri oggetti del proprio *cluster*, $b(i)$ è la più bassa tra le distanze medie tra i e tutti gli oggetti appartenenti agli altri *cluster* (tranne il *cluster* di cui i è membro). Il coefficiente *silhouette* viene quindi calcolato così:

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Il valore di tale coefficiente può oscillare tra -1 e 1. Valori negativi non sono buoni, perché significano che $a(i)$ è più grande di $b(i)$, cioè la distanza media dell'oggetto i con gli elementi del suo *cluster* è più grande della minima tra le distanze medie con gli oggetti degli altri *cluster*.

Per calcolare il valore della *silhouette* di un *cluster*, basta fare la media dei valori di *silhouette* di tutti i membri appartenenti allo stesso *cluster*. Infine, per calcolare il valore di *silhouette* di un intero *clustering*, basta fare la media dei valori di *silhouette* di ogni punto.

La *silhouette* viene comunemente usata per scegliere un buon valore di K nell'algoritmo K-Means. Si procede con il calcolare il valore della *silhouette* per l'intero *clustering* fatto con diversi K: il K che restituirà il valore più alto sarà quello selezionato. Possiamo vederne un esempio in Figura 4.2: dopo aver provato il K-Means con vari valori di K, il valore per il quale la *silhouette* è più alta è $K = 2$.

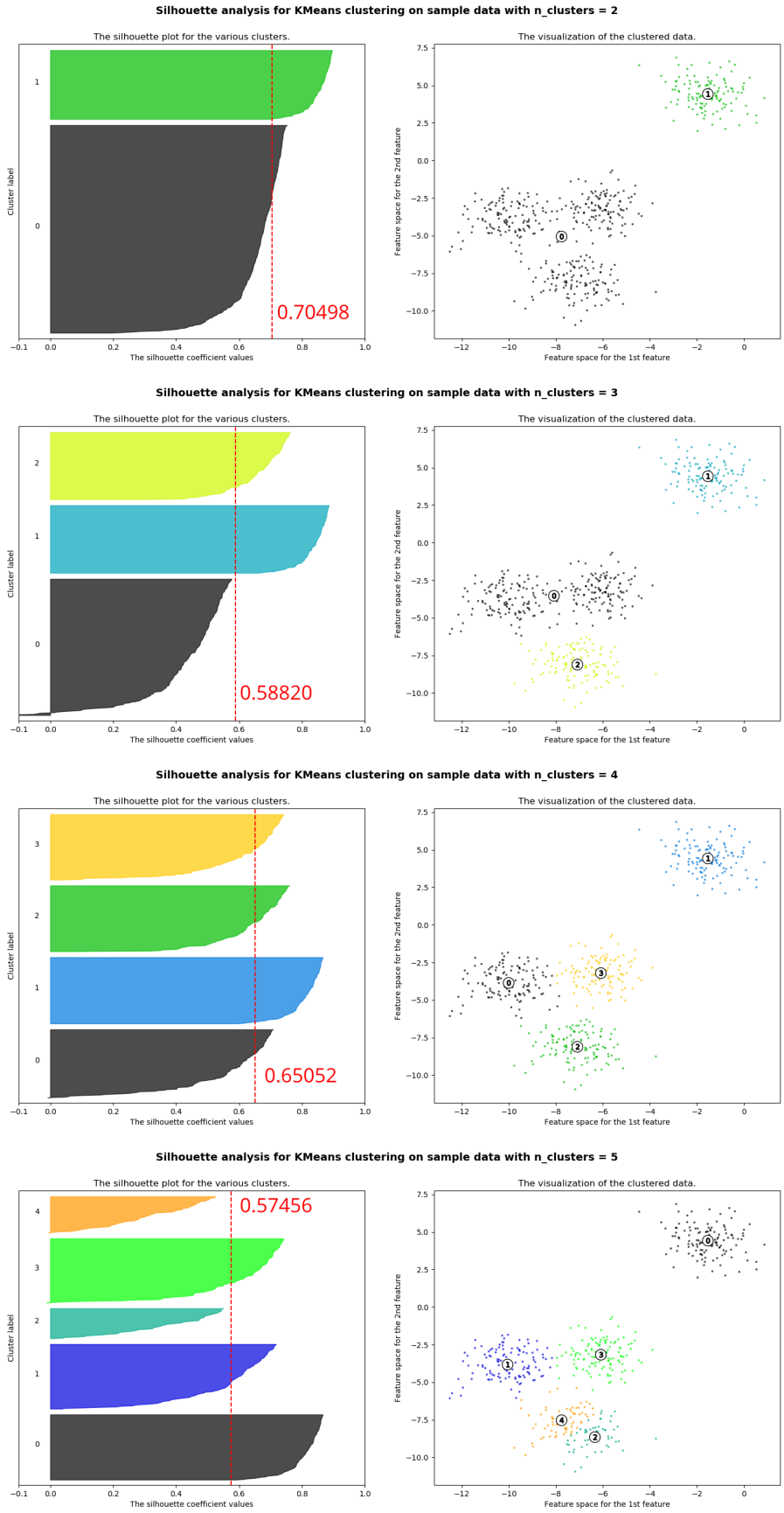


Figura 4.2: Esempio di analisi del valore *silhouette* per diversi K dopo l'applicazione dell'algoritmo K-Means
[Fonte: `https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html`]

4.2 Misure esterne

Come abbiamo detto prima, le misure esterne sono quelle misure sul *clustering* che si possono fare solo avendo a disposizione delle informazioni oltre alla collezione di dati, e queste informazioni sono le etichette di classe di ciascun dato. L'insieme delle etichette di classe di ciascun dato è chiamato *ground truth*.

4.2.1 *Homogeneity*, *Completeness* e *V-Measure*

Innanzitutto, si definisce $H(C|K)$ come l'entropia condizionale della suddivisione dei dati nelle classi *ground-truth* C rispetto alla suddivisione dei dati nei *cluster* K:

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

dove n è il numero totale di campioni, n_k è il numero di campioni che appartiene al *cluster* K e $n_{c,k}$ è il numero di campioni della classe C che sono stati assegnati al *cluster* K. Maggiore è l'entropia $H(C|K)$, maggiore è la suddivisione in modo disordinato delle classi C nei *cluster* K.

Si definisce, poi, l'entropia di una classe $H(C)$:

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

dove n è il numero totale di campioni e n_c è il numero di campioni che appartengono alla classe c. $H(C)$ assume valori elevati quando le varie classi C hanno un numero simile di elementi n_c. Si definiscono inoltre $H(K)$ e $H(K|C)$ in modo speculare rispetto a $H(C)$ e $H(C|K)$.

La *Homogeneity* è una misura che serve per verificare che un *cluster* contenga elementi appartenenti solo ad una classe, quindi per misurarne la

purezza.

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

La *completeness* invece serve per misurare il grado per il quale tutti gli elementi appartenenti ad una stessa classe siano finiti nello stesso *cluster*.

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

Sia l'*homogeneity* che la *completeness* assumono valori tra 0 e 1: 1 è il caso in cui tutti gli elementi appartengono alla stessa classe per la *homogeneity* oppure è il caso per cui tutti gli elementi appartenenti ad una stessa classe siano finiti nello stesso *cluster* per la *completeness*. La media armonica tra *homogeneity* e *completeness* è la *V-Measure*.

$$v = \frac{(1+\beta) * homogeneity * completeness}{(\beta * homogeneity) + completeness}$$

Solitamente il valore di β viene impostato a 1. Se $\beta < 1$ significa che verrà dato più peso alla *homogeneity* che alla *completeness*, invece se $\beta > 1$ viceversa. Anche la V-Measure può assumere valori compresi fra 0 e 1.

Il problema di queste metriche è che non sono normalizzate contro l'etichettatura casuale: a seconda del numero di campioni, di *cluster* e di *ground truth*, un *clustering* casuale non produrrà sempre gli stessi risultati in termini di *homogeneity*, *completeness* e *v-measure*. Il valore ottenuto, in particolare, non sarà 0, soprattutto quando si ha un numero elevato di *cluster*. Questo problema può essere ignorato, soprattutto quando il numero di campioni è maggiore di un migliaio e il numero di *cluster* è minore di 10. In tutti gli altri casi è conveniente usare altri tipi di misure, come l'*Adjusted Rand Index* che vedremo nel paragrafo successivo.

4.2.2 *Adjusted Rand Index*

L'*Adjusted Rand Index* è una misura della similarità tra due diverse assegnazioni di etichette. Per misurare un *clustering* confronteremo un'assegnazione di etichette fatta dal *clustering* con il *ground truth*. In particolare, l'*Adjusted Rand Index* è in grado di ignorare le permutazioni. Supponiamo di avere un *ground truth* come assegnato in Tabella 4.1: se calcoliamo l'ARI dell'assegnazione a) rispetto al *ground truth* otteniamo un certo punteggio; se decidessimo di sostituire le etichette dell'assegnazione a), in particolare gli 0 con gli 1, gli 1 con i 3 e i 2 con i 4, ottenendo così l'assegnazione b), l'ARI non cambierebbe affatto.

ground truth	0	0	0	1	1	1
assegnazione a	0	0	1	1	2	2
assegnazione b	1	1	3	3	4	4

Tabella 4.1: Esempio di diverse assegnazioni di *clustering* paragonate alle etichette di classe reali (*ground truth*)

In caso di un *clustering* perfetto, ovvero un *clustering* nel quale tutte le etichette assegnate ai punti coincidessero con le etichette di classe reali, il valore massimo raggiunto è di 1. Un'assegnazione casuale nei *cluster* produce un *adjusted rand index* di 0, mentre il punteggio peggiore è -1.

Prima di definire l'*adjusted rand index*, definiamo il *rand index*.

$$RI = \frac{a+b}{C_2^{n_{samples}}}$$

dove:

- C è il *ground truth*, cioè l'insieme delle etichette di classe reali
- K è l'insieme delle etichette assegnate dal *clustering*

- a è il numero di coppie di elementi che sono nello stesso insieme sia in C sia in K
- b è il numero di elementi che sono stati assegnati in *cluster* diversi in C e in K
- $C_2^{n_{samples}}$ è il numero di tutte le possibili coppie nella collezione dati.

Il *Rand Index* ha il difetto che non restituisce 0 per un'assegnazione di *cluster* casuale. Si può porre rimedio a questo effetto con il *Rand Index*, stimato $E[I]$ come segue:

$$ARI = \frac{RI - E[I]}{\max(RI) - E[RI]}$$

4.2.3 *Fowlkes-Mallows*

Questa misura è definita come la media geometrica tra precisione e richiamo.

$$FMI = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}}$$

dove:

- TP è il numero di coppie di punti che si trovano nello stesso *cluster* sia per l'insieme delle etichette assegnate dal *clustering* K, sia per l'insieme delle etichette reali C
- FP è il numero di coppie di elementi che appartengono allo stesso *cluster* nel *ground truth* C, ma non nell'insieme delle etichette assegnate dal *clustering* K
- FN è il numero di coppie che nell'insieme delle etichette di *clustering* K, ma non nell'insieme di etichette reali C, appartengono allo stesso *cluster*.

Nel caso di un'assegnazione perfetta, anche questa misura restituisce 1. Un'assegnazione dei *cluster* casuale produce un valore di FM intorno allo 0.

Capitolo 5

AdaPro

5.1 Origini dell'algoritmo

L'algoritmo AdaPro nasce da una collaborazione del Politecnico di Torino con una nota azienda petrolifera italiana. Prende il suo nome dall'inglese (*Adaptive clustering algorithm with prototypes*). In origine, questo algoritmo è stato sviluppato per essere eseguito su delle collezioni di dati molto specifiche, ovvero dei dati estratti a partire da un'immagine ottenuta al microscopio elettronico di una sottilissima sezione di roccia. Ogni dato estratto da questa immagine rappresenta un poro nel campione di roccia. Nel campo della geologia lo studio dei pori permette di stimare la presenza di un combustibile fossile, come il petrolio. Concentrandosi maggiormente nell'ottica dell'analisi dati, il problema principale era che gli algoritmi di *clustering* tradizionali, applicati ai pori geologici, non portavano a dei risultati molto soddisfacenti, ma al contempo non si poteva usare un approccio supervisionato a causa dell'elevato numero di dati senza etichetta di classe. Generare, a partire da questi, una collezione di dati etichettata, avrebbe comportato un lavoro manuale troppo oneroso da parte dei geologi, gli esperti di settore in questo caso.

5.2 Panoramica

AdaPro, grazie al suo funzionamento semi-supervisionato, per funzionare ha bisogno solo di una piccolissima parte di dati che siano già classificati. In particolare, qualora non si disponesse già di una porzione minima di dati etichettati, la loro etichettatura manuale risulterebbe essere l'unico passaggio da effettuarsi manualmente. I dati etichettati vengono utilizzati come **prototipi**, ovvero i dati che maggiormente sono rappresentativi del loro *cluster* di appartenenza.

Un'altra delle qualità di questo algoritmo è infatti dovuta alla selezione automatica degli iperparametri: ciò significa che non dovrà esserci un intervento umano prima dell'applicazione dell'algoritmo al fine di capire quali siano i parametri migliori per ottenere un buon *clustering*, ma sarà l'algoritmo stesso a studiare internamente i dati e capire quali siano i settaggi migliori, rendendo così l'applicazione dell'algoritmo del tutto automatica.

Come anticipato, AdaPro è un algoritmo semi-supervisionato che, nello specifico, si basa sul *clustering* gerarchico. L'algoritmo è strutturato in due diversi stadi. Il primo stadio, nel quale non si fa uso delle etichette dei prototipi e che quindi è non supervisionato, consiste nell'applicazione di un *clustering* agglomerativo. Da questo primo *clustering* otteniamo così quelli che abbiamo battezzato come *super-cluster*. Questo primo *clustering* è fatto su tutti i dati, ovvero sull'unione del *reference dataset*, cioè la collezione di dati che hanno l'etichetta di classe (i prototipi), e sul *target dataset*, ovvero la collezione di tutti gli altri dati, privi dell'etichetta di classe.
Nel secondo stadio i *super-cluster* vengono esaminati uno ad uno: studiando

la presenza degli elementi del *reference dataset*, si può capire se il *super-cluster* sia puro o no, e procedere in quest'ultimo caso con un'ulteriore divisione, ottenendo così i cosiddetti *sub-cluster*. Sarà proprio questa ulteriore divisione dei *super-cluster* a conferire maggiore purezza ai *cluster* finali, raffinando così i risultati grazie ad una piccola porzione di dati etichettati.
Vediamo un esempio in Figura 5.1 che può farci capire meglio ciò di cui stiamo parlando.

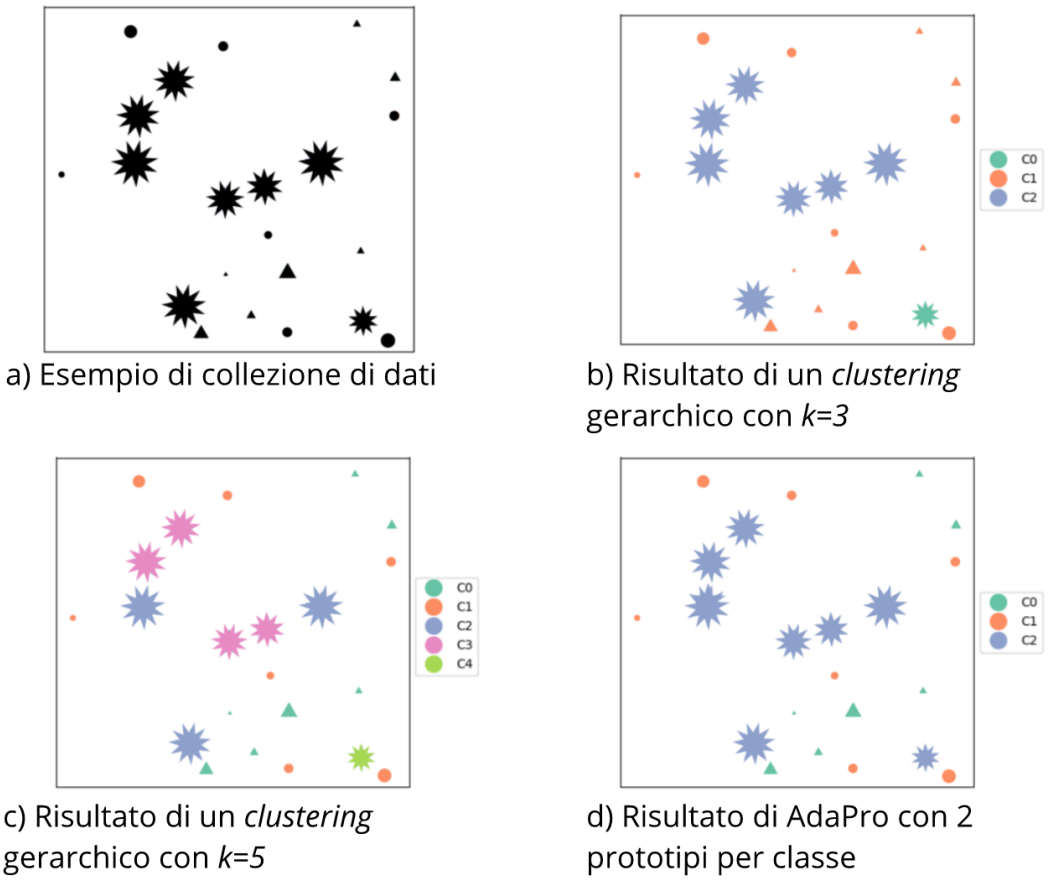


Figura 5.1: Esempio di *clustering* effettuato da un algoritmo gerarchico e da AdaPro

La collezione di dati che vediamo in Figura 5.1a) è generata sinteticamente e rappresenta un insieme di dati con diverse caratteristiche. Possiamo comunque pensare che gli elementi grandi a forma di stella appartengano ad una classe, quelli a forma di triangolo ad un'altra e quelli a forma sferica ad un'altra ancora, per un totale di tre classi. Ogni classe contiene 10 elementi. Inoltre, gli elementi appartenenti ad una stessa classe non sono tutti uguali, differendo tra di loro per due attributi, quali dimensione e numero di spigoli:

- Elementi a stella: hanno diverse dimensioni; il numero di spigoli è 22 per 6 elementi, 20 per altri 2 e 24 per gli ultimi 2. L'attributo della dimensione di questi elementi è caratterizzato da una distribuzione gaussiana di media 60 e deviazione standard uguale a 20
- Elementi a triangolo: questi elementi hanno tutti 3 spigoli, mentre l'attributo dimensione ha una distribuzione gaussiana con media uguale a 0.5 e una deviazione standard di 1
- Elementi sferici: hanno un solo spigolo e hanno l'attributo dimensione caratterizzato da una distribuzione gaussiana con media 0.5 e deviazione standard di 1

Effettuando un *clustering* gerarchico con $k = 3$, come vediamo in Figura 5.1b, otteniamo tre *cluster*. Il primo *cluster*, C0, contiene solo gli elementi a stella più piccoli, C2 contiene tutti gli altri elementi a forma di stella e C1 contiene gli elementi sferici e a triangolo insieme. La qualità del *clustering* ottenuto è bassa, perché gli elementi a forma di triangolo e di cerchio sono nello stesso *cluster* e gli elementi a stella sono separati in due *cluster* diversi. Proviamo così ad aumentare la purezza dei *cluster* aumentando il valore di k a 5. Possiamo subito notare in Figura 5.1c che gli elementi a triangolo e gli

elementi sferici sono stati separati in due *cluster* diversi, rispettivamente C0 e C1. Gli elementi a stella sono invece divisi in ben tre *cluster* diversi: C2, C3 e C4. In un caso reale il valore di *cluster* K non è noto a priori. Facendo un'analisi della *silhouette* per i diversi *cluster*, come quella fatta nel capitolo precedente (vedi Figura 4.2), si nota che il valore di K che massimizza i risultati è 2. Questa soluzione vede due *cluster*: il primo, che contiene solo elementi a stella e il secondo, che contiene sia elementi sferici sia elementi a triangolo. La coesione complessiva dei *cluster* è massimizzata.

La classe degli elementi a stella è caratterizzata dall'avere una varianza maggiore rispetto alle altre classi per quanto riguarda l'attributo dimensione. Gli elementi a triangolo e sferici hanno invece una dimensione piuttosto simile, ma un numero diverso di spigoli (3 per i primi, 1 per gli ultimi). Dal momento che il *clustering* gerarchico tende a separare per primi gli elementi a stella data la loro varianza maggiore, la divisione degli elementi a triangolo da quelli sferici richiede l'aumento del numero di *cluster* K. L'avere a disposizione delle informazioni aggiuntive, come le etichette di classe almeno per qualche elemento, può facilitare la suddivisione in *cluster*.

AdaPro riesce a raggiungere una suddivisione corretta grazie al suo approccio di *clustering* a 2 stadi. In Figura 5.1d) possiamo vedere come si comporta AdaPro ricevendo in *input* la stessa collezione di dati, più 2 prototipi per ogni classe. Servendosi dell'analisi sui valori della *silhouette*, il primo passo di AdaPro genererà due *cluster*: il primo con gli elementi a stella e il secondo con gli elementi sferici insieme a quelli a triangolo. Il secondo passo, invece, studiando la distribuzione dei prototipi nei diversi *cluster*, rileverà la presenza di un *cluster* non puro. L'informazione sarà così sfruttata per dividere tale *cluster* in due *sub-cluster*, uno contenente solo elementi triangolari,

l'altro solo elementi sferici.

5.3 Funzionamento dell'algoritmo

AdaPro è un algoritmo semi-supervisionato. Per poter funzionare ha bisogno quindi che una parte dei dati sia etichettata. Avremo quindi un *reference dataset* R_l, nel quale i dati sono provvisti dell'etichetta di classe, e un *target dataset* T, contenente i dati sui quali dover fare propriamente il *clustering*. Chiameremo W l'insieme del *target dataset* e del *reference dataset* sprovvisto delle etichette di classe (R).

$$W = R \cup T$$

Il primo passo consiste nella standardizzazione dei dati, come abbiamo discusso nel Capitolo 2 (vedi Equazione 2.1).
Dopodiché si procede con i passi descritti in Algoritmo 5. Si esegue *clustering* agglomerativo (linea 1), il quale genererà un dendrogramma (vedi Immagine 5.2). Per l'applicazione del *clustering* gerarchico, AdaPro usa la metrica delle distanze euclidee e il *linkage* col metodo di Ward. Se l'utente lo desiderasse, è possibile cambiare questi parametri in base alle caratteristiche dei dati analizzati.

Una volta che si dispone dei risultati del *clustering* agglomerativo, è possibile calcolare la *silhouette* al variare dell'altezza del taglio sul dendrogramma, quindi al variare del numero di *cluster* k. Ogni diversa altezza di taglio produrrà quindi un insieme di *cluster* C_k, definito come:

$$C_k = \left\{c_0^k, \ldots, c_i^k, \ldots, c_{k-1}^k\right\}, \quad 2 \leq k \leq max_k$$

Il valore max_k per le nostre analisi è fissato arbitrariamente in base al numero di *cluster* che si vogliono ottenere. Qualora si stesse analizzando una collezione di dati con un numero elevato di possibili classi, è possibile aumentare tale valore.

Algoritmo 5 AdaPro

1: Si fa *clustering* gerarchico su W

2: Prima di tagliare il dendrogramma, si calcola quale sia il miglior numero di *cluster* $\hat{k}$:

$$\hat{k} = \arg\max_k \left(silh(k) - silh(k+1)\right), \quad 2 \leq k \leq max_k$$

3: Effettuato il taglio, si ottengono i *super-cluster*:

$$C_{\hat{k}} = \left\{c_0^{\hat{k}}, \ldots, c_i^{\hat{k}}, \ldots, c_{\hat{k}-1}^{\hat{k}}\right\}$$

4: **per ogni** *super-cluster* $c_i^{\hat{k}}$ **:**

5: Si calcola l'insieme K' dei possibili numeri di *cluster* per i quali tagliare il sottodendrogramma avente radice in $c_i^{\hat{k}}$:

$$K'(c_i^{\hat{k}}) = \left\{k' | FM(subclusters_{k'}(c_i^{\hat{k}})) - FM(subclusters_{k'-1}(c_i^{\hat{k}})) > 0\right\}$$

6: Si sceglie $\hat{k'}$ tale che:

$$\hat{k'} = \begin{cases} \arg\max_{k'} \left(FM(subclusters_{k'}(c_i^{\hat{k}})), k' \in K'\right) & , |K'| > 0 \\ 1 & , |K'| = 0 \end{cases}$$

7: Se $|K'| > 0$, dal *super-cluster* $c_i^{\hat{k}}$ vengono generati i *sub-cluster*:

$$C_{\hat{k'}} = \left\{c_0^{i,\hat{k},\hat{k'}}, \ldots, c_i^{i,\hat{k},\hat{k'}}, \ldots, c_{\hat{k'}-1}^{i,\hat{k},\hat{k'}}\right\}, \quad 2 \leq k' \leq max_{k'}$$

8: **fine iterazione**

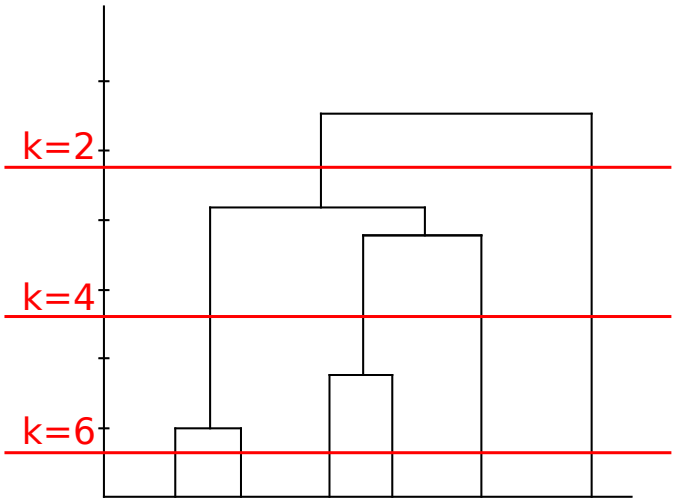


Figura 5.2: Esempio di un dendrogramma generato da un *clustering* gerarchico con tagli effettuati a diverse altezze

Per scegliere il numero di *cluster* $\hat{k}$ desiderato, quindi l'altezza alla quale effettuare il taglio del dendrogramma, possiamo guardare il grafico della *silhouette* e scegliere il valore seguito dalla decrescita maggiore (vedi Immagine 5.3). Viene usato il punto prima della massima decrescita perché, nelle collezioni di dati mineralogici che originariamente avrebbe dovuto analizzare AdaPro, all'aumentare del valore k il valore della *silhouette* decresce. La massima decrescita della *silhouette* è allora considerata, euristicamente, il momento giusto al quale dividere ulteriormente i *super-cluster*: la decrescita del valore della *silhouette* indica proprio un peggioramento significativo della qualità del *clustering*. Più formalmente, scegliamo l'altezza del taglio che faccia ottenere il numero di *cluster* $\hat{k}$, come indicato alla linea 2 dell'algoritmo:

$$\hat{k} = \arg\max_{k} \left(silh(k) - silh(k+1) \right), \quad 2 \leq k \leq max_k$$

Il valore di k è considerato a partire da 2, perché altrimenti, se non ci si aspettasse di dividere i dati almeno in due *cluster*, non avrebbe senso applicare un

algoritmo di *clustering*.

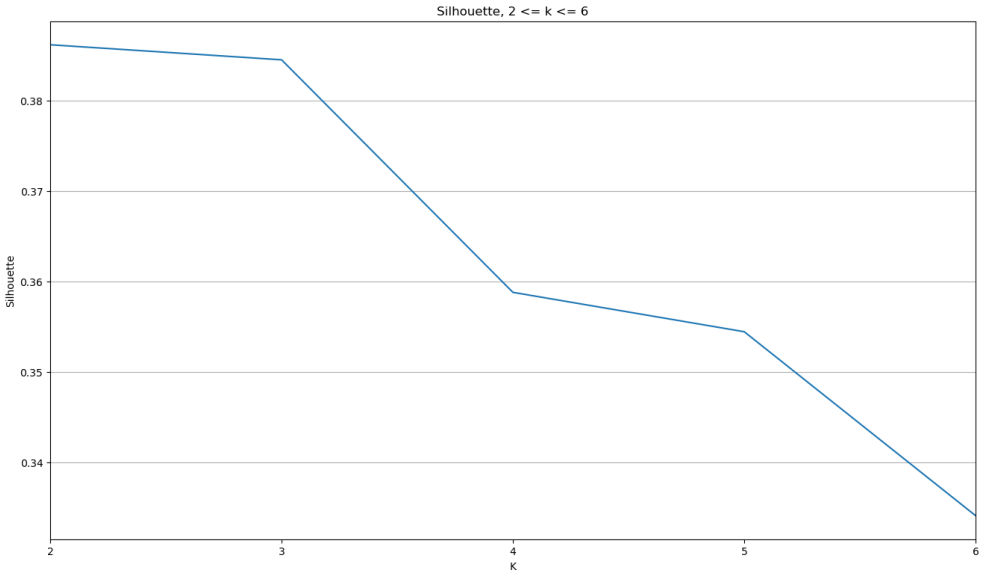


Figura 5.3: Esempio grafico della *silhouette*, usato per scegliere il valore $\hat{k}$. Nell'esempio mostrato verrà scelto $k = 3$

Tagliando il dendrogramma (linea 3), otteniamo così l'insieme dei *super-cluster* $C_{\hat{k}}$.

$$C_{\hat{k}} = \left\{ c_0^{\hat{k}}, \ldots, c_i^{\hat{k}}, \ldots, c_{\hat{k}-1}^{\hat{k}} \right\}$$

Fino a qui le informazioni sulle etichette di classe del *reference dataset* R_l non sono state utilizzate. Esse verranno utilizzate nel secondo stadio del nostro algoritmo, per dividere ulteriormente quei *super-cluster* che non fossero abbastanza puri e che contenessero elementi appartenenti a più classi contemporaneamente.

Nel secondo stadio di AdaPro, ognuno dei *super-cluster* generati viene esaminato singolarmente, per poter capire se è conveniente o meno procedere con un'ulteriore divisione di questi in *cluster* ancora più piccoli, chiamati *sub-cluster*. Viene quindi considerato il sottodendrogramma che pone la sua radice nel *super-cluster* attualmente considerato.

Per decidere se effettuare un'ulteriore divisione del *super-cluster* considerato, vengono calcolati i valori del *Fowlkes-Mallows* (vedi Paragrafo 4.2.3), basandosi sul calcolo di una matrice di contingenza per ognuno dei possibili *clustering* ottenuti tagliando il sottodendrogramma corrispondente. Per il calcolo della matrice di contingenza e del *Fowlkes-Mallows* verranno considerati solo quegli elementi appartenenti al *super-cluster* che appartenevano in origine al *reference dataset* R_l e per i quali è quindi possibile conoscere la classe di appartenenza. Stavolta, il grafico di *Fowlkes-Mallows* non viene generato, come per la *silhouette*, per valori di k a partire da 2, ma a partire da 1: non è detto che un *super-cluster* debba per forza essere diviso in *sub-cluster*. L'altezza alla quale effettuare un eventuale taglio del sottodendrogramma è quindi scelta nel punto nel quale segue il massimo incremento di *Fowlkes-Mallows* (vedi Figura 5.4). Anche questa è una scelta euristica, dovuta al fatto che non sempre esiste un massimo assoluto nei grafici di *Fowlkes-Mallows*. In particolare, i grafici potrebbero essere monotoni decrescenti o potrebbero subire un incremento per $k' > 1$, fino al raggiungimento di un massimo relativo. Un aumento del valore di *Fowlkes-Mallows* all'aumentare di k rappresenta, perciò, una suddivisione del *super-cluster* che porta ad una maggiore qualità del *clustering*.

In maniera più formale (vedi linea 5), viene prima calcolato un insieme dei possibili valori k': vengono selezionati solo i tagli che hanno comportato un aumento del *Fowlkes-Mallows*.

$$K'(c_i^{\hat{k}}) = \left\{ k' | fowlkes(subcluster(c_i^{\hat{k}})) - fowlkes(subcluster(c_i^{\hat{k}})) > 0 \right\}$$

Successivamente (linea 6), se l'insieme non è vuoto, viene scelto il valore di

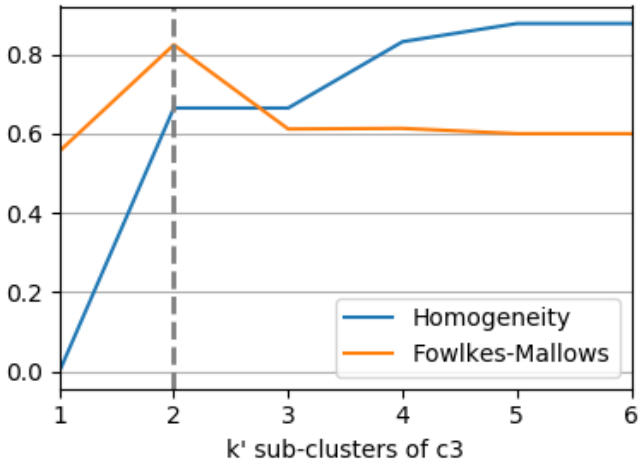


Figura 5.4: Esempio di grafico di *Fowlkes-Mallows* usato per la scelta del $\hat{k'}$

k' che comporta il massimo aumento di *Fowlkes-Mallows*.

$$\hat{k'} = \arg\max_{k'} \left(fowlkes(subcluster_{k'}(c_i^{\hat{k}})), k' \in K' \right)$$

Se l'insieme $K'(c_i^{\hat{k}})$ invece risulta vuoto, sceglieremo $\hat{k'} = 1$, ovvero il *sub-cluster* sarà identico al *super-cluster*, senza dividerlo ulteriormente.
Tagliato il sottodendrogramma (linea 7), si ripete l'analisi del *Fowlkes-Mallows* per ogni *super-cluster*, finché non saranno stati esaminati tutti.

5.4 1-Level-AdaPro

A partire dalla versione completa di AdaPro, abbiamo sviluppato una sua variante semplificata denominata 1-Level-AdaPro (vedi Algoritmo 6). L'idea alla base di questa variante è quella di eliminare il passaggio di generazione dei *super-cluster*, mantenendo invece la generazione dei *sub-cluster*. Ciò consentirà di sperimentare l'efficacia dell'utilizzo di un doppio livello di *clustering*, come nella versione completa di AdaPro. 1-Level-AdaPro consiste quindi nel considerare tutti i dati come facenti parte di un unico grande

super-cluster e decidere se e dove tagliare il dendrogramma generato a partire da questo grande *super-cluster* con la stessa tecnica euristica usata in AdaPro, ovvero studiando i grafici del Fowlkes-Mallows (linee 2 e 3 dell'Algoritmo 6). In tutto si avrà quindi un solo taglio del dendrogramma, contro i due di AdaPro. Oltre che verificare l'utilità del doppio livello di *clustering* di AdaPro per la maggior parte delle collezioni di dati, questa variante di AdaPro risulta adatta per certe particolari collezioni di dati e può portare in alcuni casi a dei risultati simili o addirittura, solo in certi casi, migliori di AdaPro. Dal momento che 1-Level-AdaPro opera sempre facendo uso delle informazioni sulle etichette di classe di una piccola porzione di dati, è anch'esso un algoritmo semi-supervisionato.

Algoritmo 6 1-Level-AdaPro

1: Si fa *clustering* gerarchico su W

2: Si calcola l'insieme K' dei possibili tagli del dendrogramma:

$$K'(W) = \{k' | FM(subclusters_{k'}(W)) - FM(subclusters_{k'-1}(W)) > 0\}$$

3: Si sceglie $\hat{k'}$ tale che:

$$\hat{k'} = \begin{cases} \arg\max_{k'} \left(FM(subclusters_{k'}(W)), k' \in K'\right) & , |k'| > 0 \\ 1 & , |k'| = 0 \end{cases}$$

4: Se $|k'| > 0$ dal *super-cluster* W vengono generati i *sub-cluster*:

$$C_{\hat{k'}} = \left\{ c_0^{\hat{k'}}, \ldots, c_i^{\hat{k'}}, \ldots, c_{\hat{k'}-1}^{\hat{k'}} \right\}, \quad 2 \leq k' \leq max_{k'}$$

Capitolo 6

Analisi sperimentale

In questo capitolo ci occuperemo di confrontare le *performance* di AdaPro e la sua variante ad un solo livello, 1-level-AdaPro, con algoritmi tradizionali come *clustering* gerarchico, KMeans e anche la sua versione semi-supervisionata MPCK-Means. Per testare gli algoritmi useremo 11 diverse collezioni di dati:

- *frogs*, reperibile su `https://archive.ics.uci.edu/ml/datasets/Anuran+Calls+%28MFCCs%29`
- *iris*, reperibile su `https://ifcs.boku.ac.at/repository/data/iris/index.html`
- *wine*, reperibile su `https://archive.ics.uci.edu/ml/datasets/wine`
- *transfusion*, reperibile su `https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center`
- *electrical grid stability*, reperibile su `https://archive.ics.uci.edu/ml/datasets/Electrical+Grid+Stability+Simulated+Data+`
- *seeds*, reperibile su `https://archive.ics.uci.edu/ml/datasets/seeds`

- *pop failures*, reperibile su `https://archive.ics.uci.edu/ml/datasets/Climate+Model+Simulation+Crashes`
- *wdbc*, reperibile su `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`
- *seismic bumps*, reperibile su `https://archive.ics.uci.edu/ml/datasets/seismic-bumps`
- *avila*, reperibile su `https://archive.ics.uci.edu/ml/datasets/Avila`
- *data banknote authentication*, reperibile su `https://archive.ics.uci.edu/ml/datasets/banknote+authentication`

La Tabella 6.1 fornisce una visione generale di tutte le collezioni di dati, della loro cardinalità, del numero di classi diverse di ciascuna e del numero di dimensioni di ogni elemento.

A partire dalle collezioni di dati sopra citate, per poter eseguire gli algoritmi semi-supervisionati su queste, è stato necessario selezionare, estraendoli da ogni insieme, i cosiddetti prototipi, ovvero dei dati etichettati, rappresentativi per ogni classe. L'informazione sulla classe di appartenenza di tutti gli altri dati è stata utilizzata solo al fine di valutare la qualità del *clustering* effettuato, tramite le misure esterne.

Per ogni collezione di dati è stata estratta una piccola percentuale di prototipi p, con un processo analogo a quella illustrato in Figura 6.1, in modo da simulare ciò che potrebbe avvenire nella realtà, dove un esperto di dominio dovrebbe attribuire manualmente l'etichetta ad una piccola parte dei dati; inoltre, come abbiamo visto nel precedente capitolo, AdaPro non necessita

	cardinalità	dimensioni	cardinalità classi
frogs (Family)	7 195	22	4
frogs (Genus)	7 195	22	8
frogs (Species)	7 195	22	10
iris	150	4	3
wine	178	13	3
transfusion	748	4	2
electrical grid stability	10 000	11	2
seeds	210	7	3
pop failures	540	18	2
wdbc	569	30	2
seismic bumps	2 584	10	2
avila	20 867	10	12
data banknote authentication	1 372	4	2

Tabella 6.1: Caratteristiche principali di ogni collezione di dati analizzata

di troppi dati etichettati.
Qualora in alcune collezioni la percentuale di dati etichettati da estrarre risultasse troppo piccola (minore di 0), si è deciso di estrarre un numero di prototipi uguale a 4 (o minore, se ne fossero disponibili meno), per permettere anche a MPCK-Means di poter funzionare correttamente e avere a disposizione dei vincoli *must-link*, oltre che quelli *cannot-link*.

Alcune collezioni di dati, come *frogs*, dispongono di più attributi che possono essere usati come etichetta di classe. In questi casi, le analisi sono state condotte più volte, utilizzando ad ogni esperimento una tra le possibili

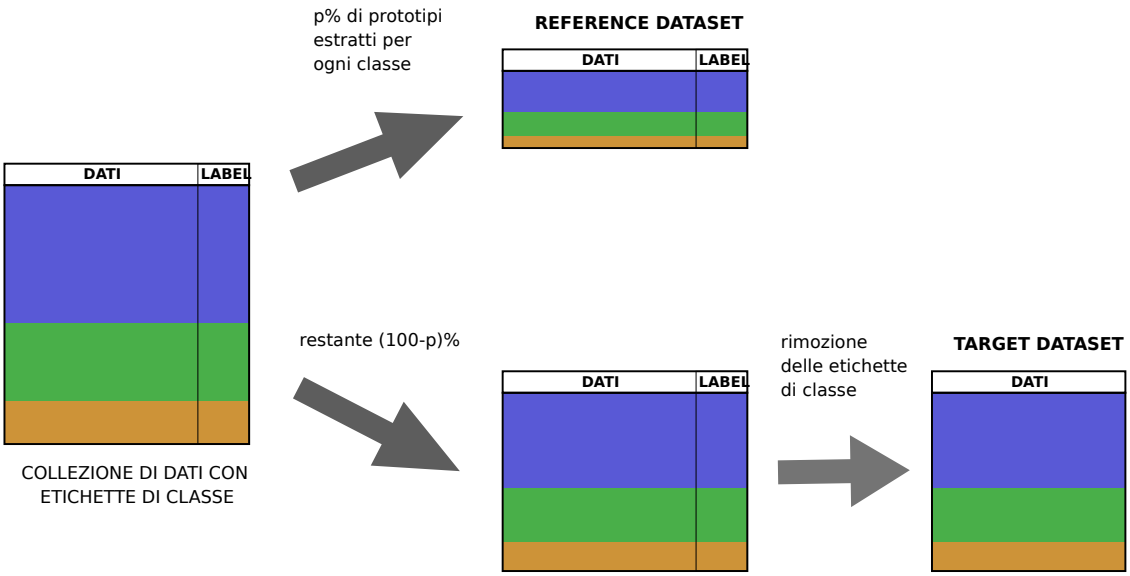


Figura 6.1: Esempio di estrazione di *target* e *reference dataset* da una collezione di dati provvista di etichette di classe

etichette di classe ed eliminando le altre.

Le analisi sono state condotte su una macchina Windows 10 con 16GB di RAM e processore i7-8565U. Tutti gli algoritmi sono stati implementati in Python 2.7.

6.1 Risultati

6.1.1 *frogs (Family)* con 1% di prototipi

In Figura 6.2 possiamo osservare i risultati delle analisi condotte sulla collezione di dati *frogs*, usando come etichetta di classe l'attributo *Family*. Troviamo riportati i risultati ottenuti dagli indici *ARI*, *V-Measure* e *Fowlkes-Mallows* per ognuno degli algoritmi eseguiti. Tra gli algoritmi troviamo: AdaPro, 1-Level-AdaPro, MPCK-Means con $|C| \leq k \leq |C| + 6$, dove $|C|$ è la cardinalità delle classi, infine i risultati del *clustering* gerarchico e di K-Means con $2 \leq k \leq 16$. Notiamo che, dal momento che AdaPro e 1-Level-AdaPro sono in grado di selezionare automaticamente quali siano gli iperparametri migliori, non è necessario eseguirli più volte con diversi valori, come invece siamo costretti a fare con gli altri algoritmi.

In Tabella 6.2 sono riportati i risultati migliori ottenuti dai vari algoritmi. Possiamo innanzitutto notare che i risultati migliori sono stati raggiunti da AdaPro, che ha generato ben 10 *cluster* separati. Degli ottimi risultati di *Fowlkes-Mallows* (maggiori di 0.7) sono stati raggiunti anche dalla variante di AdaPro 1-Level, dal *clustering* gerarchico con $k = 5$ e K-Means con $k = 5$. Osservando la colonna con i risultati inerenti alla *Homogeneity*, notiamo subito che AdaPro si distingue nettamente dagli altri. Per quanto riguarda invece la *Completeness*, tutti gli algoritmi ottengono valori simili: AdaPro riesce ad ottenere valori più alti di *Homogeneity* senza però penalizzare la *Completeness*.

In Figura 6.3a possiamo vedere che AdaPro genera in prima battuta 5 *super-cluster*. Per ognuno di essi sono riportate graficamente le diverse quantità di prototipi e la loro reale classe di appartenenza, in modo da poter capire

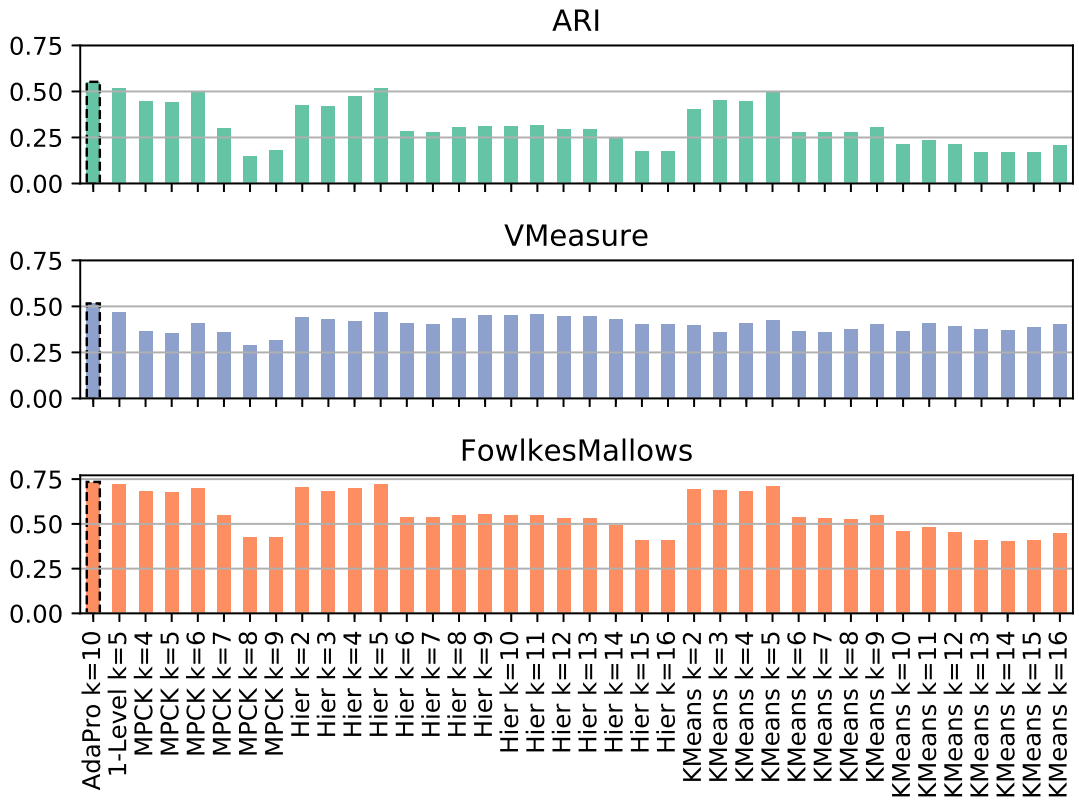


Figura 6.2: Andamento risultati su *frogs (Family)* con 1% di prototipi per classe

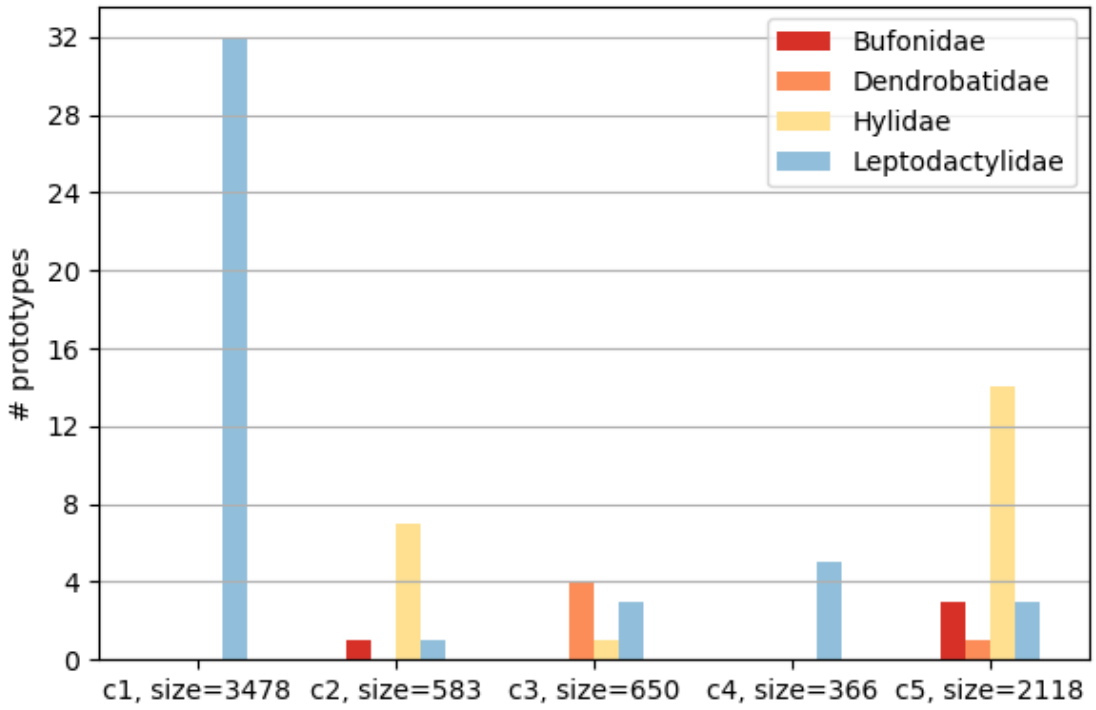
	ARI	Complet.	FMI	Homog.	V-Measure
AdaPro k=10	**0.552**	0.392	**0.735**	**0.754**	**0.516**
1-Level k=5	0.521	0.397	0.720	0.567	0.467
MPCK k=6	0.496	0.332	0.701	0.527	0.408
Hier k=5	0.521	0.397	0.720	0.567	0.467
KMeans k=5	0.503	0.359	0.709	0.516	0.423

Tabella 6.2: Risultati migliori su *frogs (Family)* con 1% di prototipi per classe

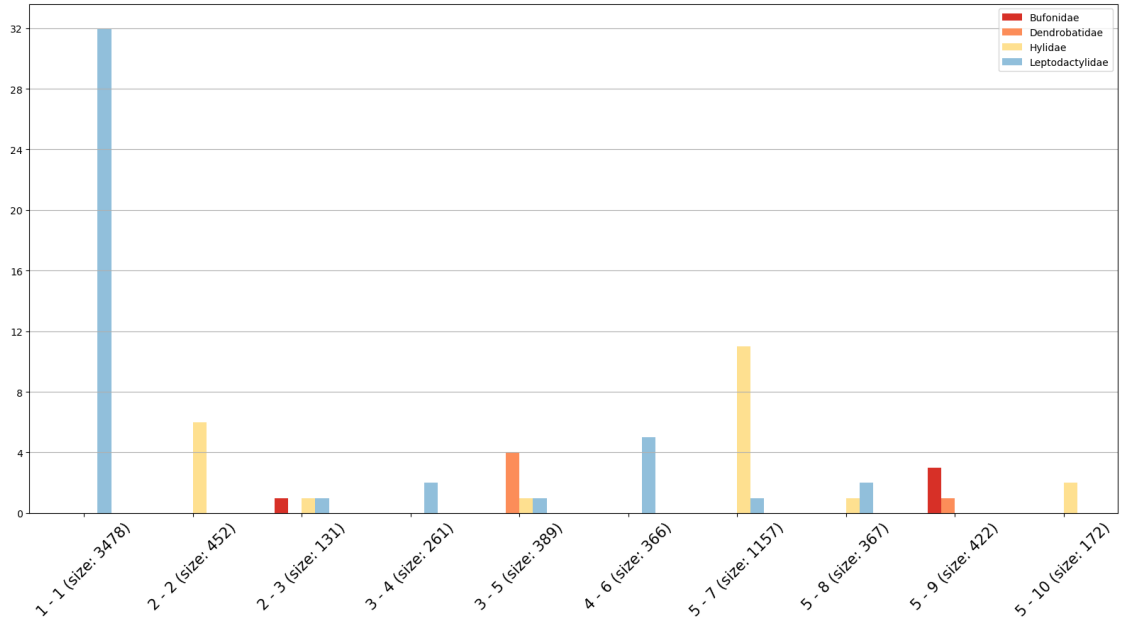
quanto puri risultino i *super-cluster*. In Figura 6.3b, possiamo invece osservare quali *sub-cluster* abbia generato la seconda fase di AdaPro. Ad essere divisi ulteriormente sono i *super-cluster* 2, 3 e 5, rispettivamente in 2, 2 e 4 *sub-cluster* per arrivare ad un totale di 10 *sub-cluster* generati.

1-Level-AdaPro invece genera subito 5 *sub-cluster*, raggiungendo il secondo miglior risultato con 0.720 di *Fowlkes-Mallows*. Il fatto che i risultati ottenuti siano inferiori rispetto a quelli ottenuti da AdaPro, conferma il fatto che la logica usata da AdaPro con due tagli del dendrogramma ottiene risultati migliori.

Infine, in Figura 6.4 possiamo confrontare i *cluster* generati da AdaPro con le classi del *ground truth* della collezione nello spazio delle tre componenti principali (PCA3D).

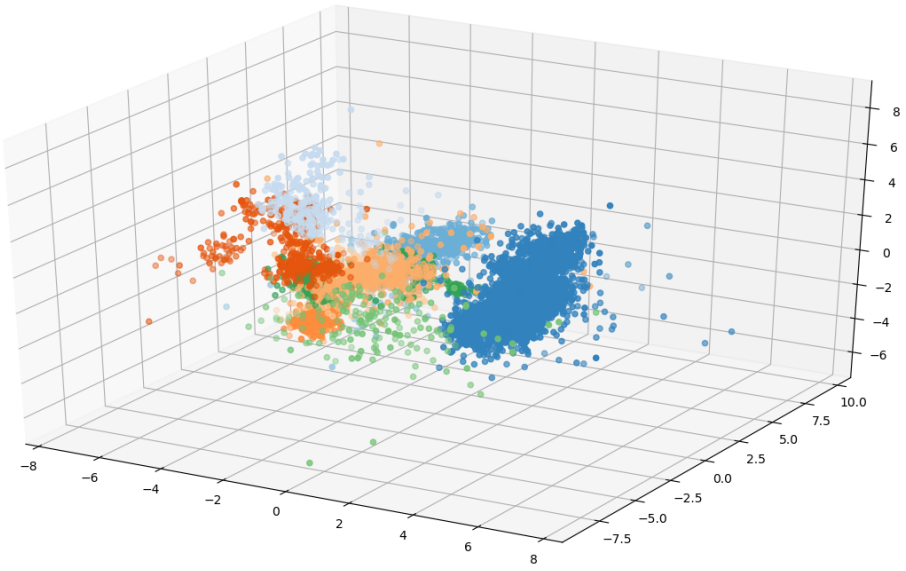


(a) *super-cluster*

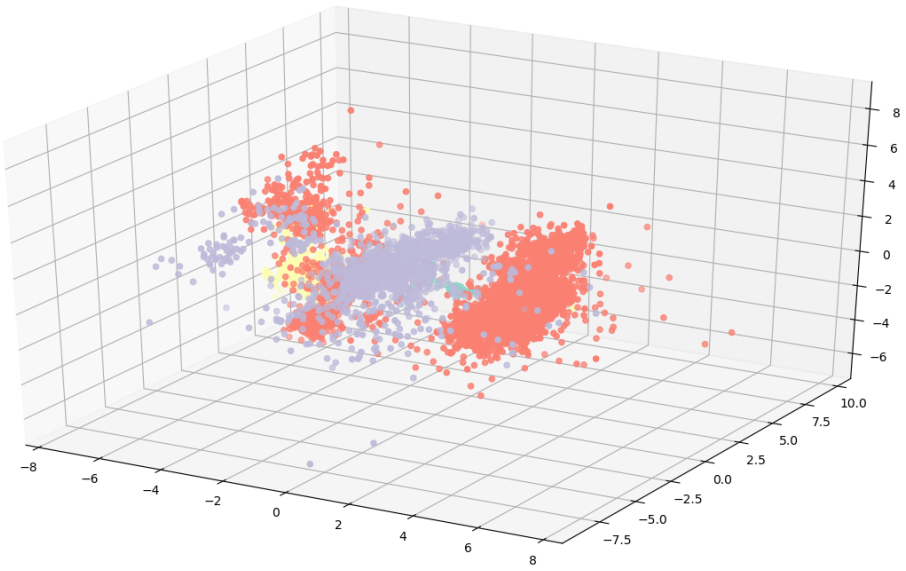


(b) *sub-cluster*

Figura 6.3: Risultati di AdaPro su *frogs (Family)* con 1% di prototipi per classe



(a) *cluster* generati da AdaPro



(b) Classi del *ground truth*

Figura 6.4: PCA3D di *frogs (Family)*

6.1.2 *frogs (Genus)* con 1% di prototipi

Continuiamo ad analizzare i risultati delle analisi condotte sulla collezione di dati *frogs*, ma stavolta usando come attributo di classe *Genus*.

Anche in questo caso, come possiamo osservare rapidamente dal grafico in Figura 6.5, è AdaPro ad ottenere le *performance* migliori con un *Fowlkes-Mallows* di 0.798. In seconda e terza posizione, in termini di punteggi ottenuti abbiamo K-Means e MPCK-Means (vedi Tabella 6.3). Osservando gli andamenti di *Fowlkes-Mallows* in Figura 6.5 vediamo che MPCK-Means, pur essendo un algoritmo semi-supervisionato, comincia a perdere qualità per gli esperimenti con $10 \leq k \leq 13$.

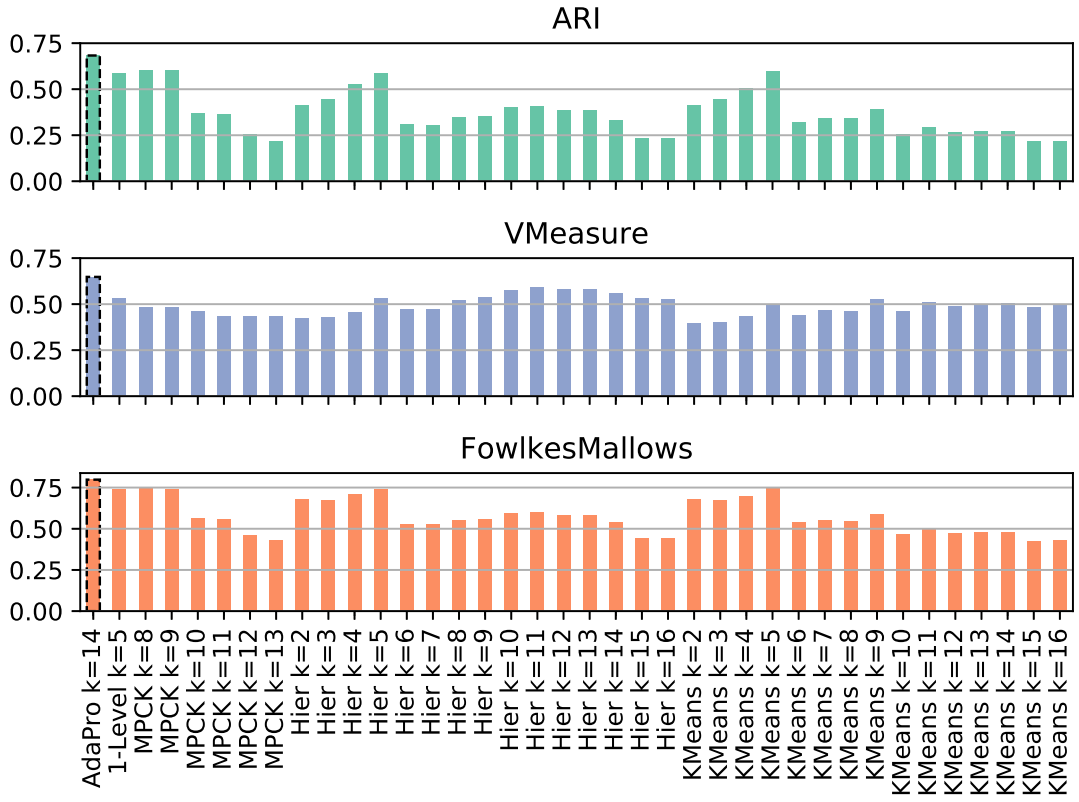
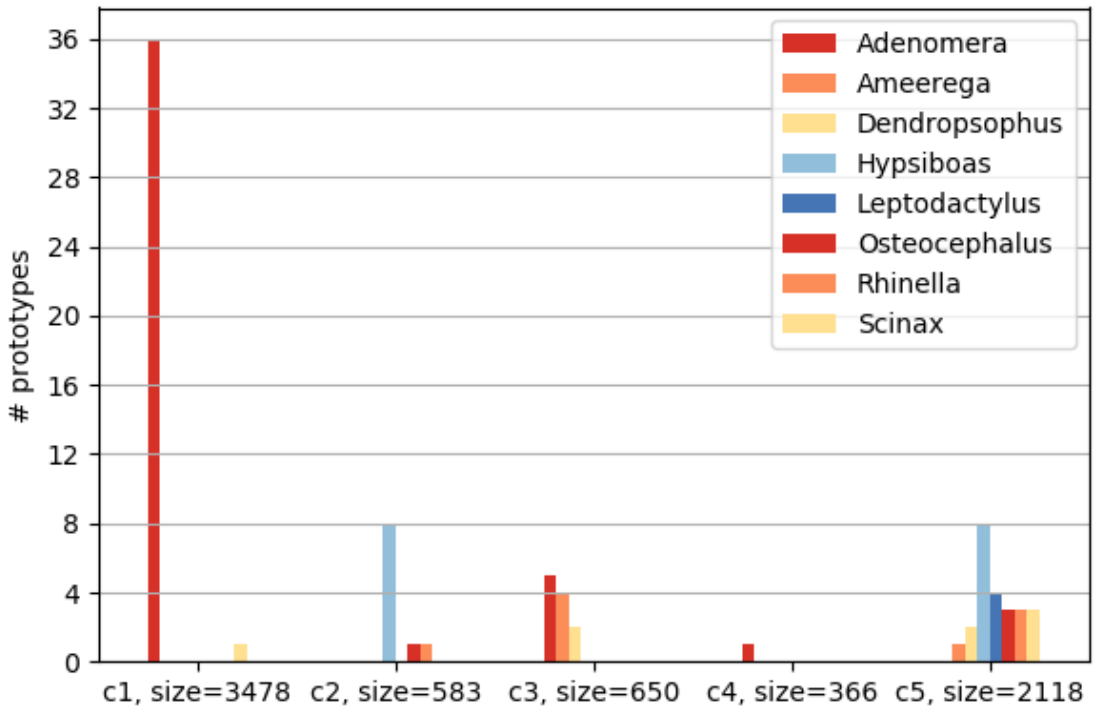


Figura 6.5: Andamento risultati su *frogs (Genus)* con 1% di prototipi per classe

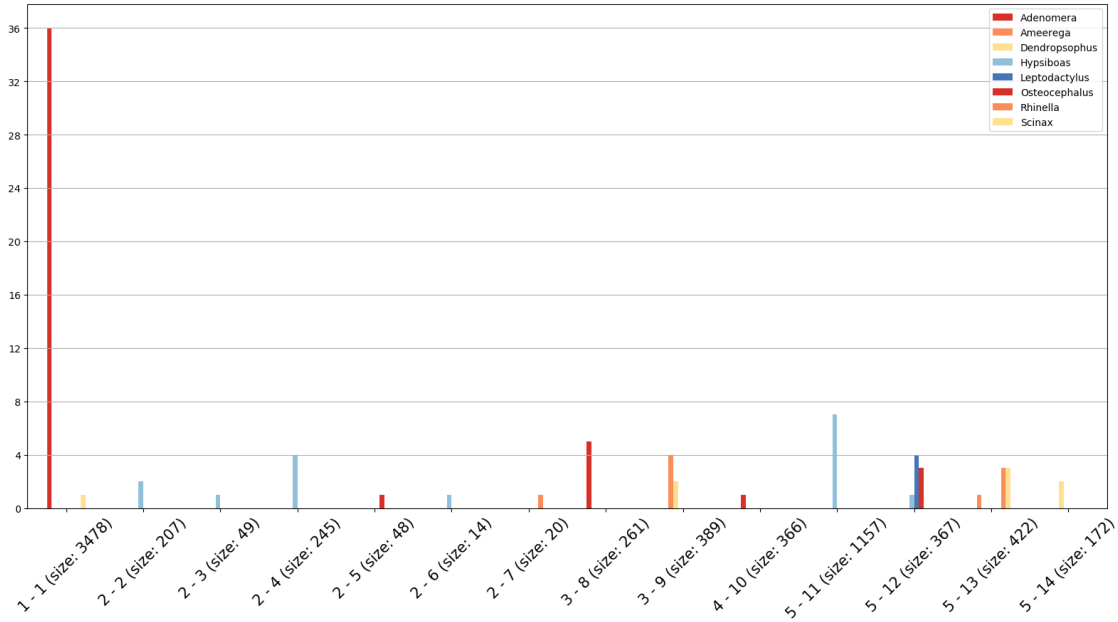
	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=14	**0.683**	**0.557**	**0.798**	**0.774**	**0.648**
1-Level k=5	0.585	0.534	0.739	0.531	0.533
MPCK k=8	0.603	0.434	0.743	0.547	0.484
Hier k=5	0.586	0.534	0.739	0.532	0.533
KMeans k=5	0.600	0.487	0.746	0.502	0.495

Tabella 6.3: Risultati migliori su *frogs (Genus)* con 1% di prototipi per classe

Concentriamo ancora le nostre attenzioni su AdaPro. In Tabella 6.3 osserviamo che, oltre che ottenere i punteggi più alti per quanto riguarda gli indici *ARI*, *V-Measure* e *Fowlkes-Mallows*, AdaPro riesce anche a distinguersi dagli altri algoritmi per quanto riguarda la *Completeness* e l'*Homogeneity*: ciò significa che i *cluster* ottenuti sono puri, senza però perdere in *Completeness*, ovvero senza dividere troppo gli elementi appartenenti ad una stessa classe di dati in *cluster* diversi. In Figura 6.6a possiamo vedere i *super-cluster* generati. Sono 5 di cui il secondo, il terzo e il quinto si vede che non sono puri, dal momento che contengono prototipi di tipo diverso. In Figura 6.6b vediamo che il *super-cluster* 2 è stato suddiviso in ben 6 *sub-cluster*, il *super-cluster* 3 in 2 *sub-cluster* e per finire il *super-cluster* 5 in 4 *sub-cluster*, arrivando così ad un totale di 14 *sub-cluster*. Di questi *sub-cluster* la maggior parte sono puri, cioè la suddivisione in *sub-cluster* ha aumentato la purezza totale dei *cluster* ottenuti, ma restano non puri i *sub-cluster* 5-12 e 5-13, che contengono ancora prototipi appartenenti a 3 famiglie diverse.
Anche in questo caso, AdaPro ottiene risultati migliori della versione semplificata 1-Level-AdaPro.
Infine, in Figura 6.7 possiamo confrontare i *cluster* generati da AdaPro con le classi del *ground truth* della collezione nello spazio delle tre componenti principali (PCA3D).

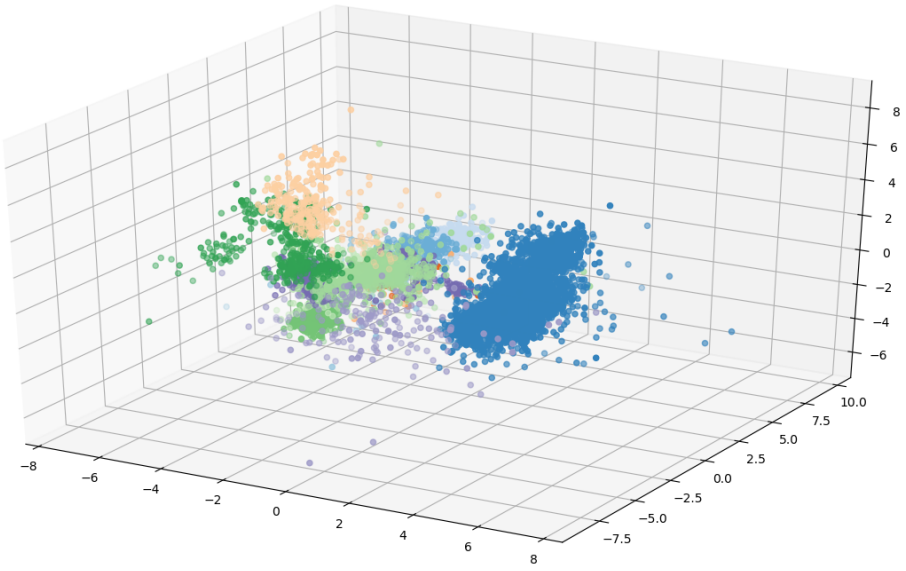


(a) *super-cluster*

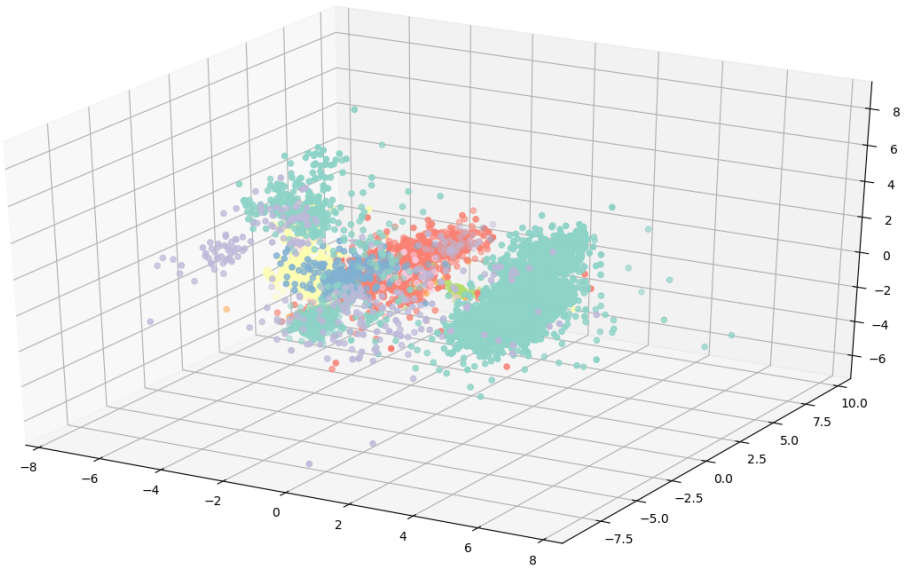


(b) *sub-cluster*

Figura 6.6: Risultati di AdaPro su *frogs (Genus)* con 1% di prototipi per classe



(a) *cluster* generati da AdaPro



(b) Classi del *ground truth*

Figura 6.7: PCA3D di *frogs (Genus)*

6.1.3 *frogs (Species)* con 1% di prototipi

Usando come attributo di classe *Species* (vedi Figura 6.8), stavolta sono il *clustering* gerarchico con $k = 5$ e 1-Level-AdaPro ad ottenere i migliori risultati, che sono davvero molto simili (possiamo considerarli uguali). Questa collezione di dati è quindi una di quelle collezioni per le quali non è necessario eseguire un doppio livello di *clustering*. Il fatto che sia il *clustering* gerarchico sia 1-Level-AdaPro abbiano generato 5 *cluster* ($k = 5$), è un punto a favore della logica semi-supervisionata: 1-Level-AdaPro grazie ai prototipi ha scelto in maniera completamente automatica il valore di k; inoltre, in un caso reale, nel quale non si conoscono le etichette reali dei dati, non si sarebbe potuto calcolare il *Fowlkes-Mallows* e quindi non si sarebbe potuto sapere che il *clustering* gerarchico con $k = 5$ era tra gli algoritmi che riuscivano ad ottenere le *performance* migliori.

In Tabella 6.4 notiamo che è AdaPro l'algoritmo a generare i *cluster* (ben 19) con la più alta *Homogeneity*, a scapito però della *Completeness*. Questo potrebbe essere considerato comunque un buon risultato, se l'obiettivo delle analisi fosse quello di ottenere dei *cluster* molto puri. Il *Fowlkes-Mallows* basso è dovuto proprio all'alto numero di *cluster* generato.

In questo scenario, MPCK-Means con i suoi 10 *cluster*, rappresenta un compromesso tra i 5 *cluster* ottenuti dal *clustering* gerarchico e i 19 di AdaPro, anche per quanto riguarda i valori di *Homogeneity* e *Completeness*.

In Figura 6.4 possiamo infine osservare i *cluster* generati da 1-Level-AdaPro con le classi del *ground truth* della collezione nello spazio delle tre componenti principali (PCA3D).

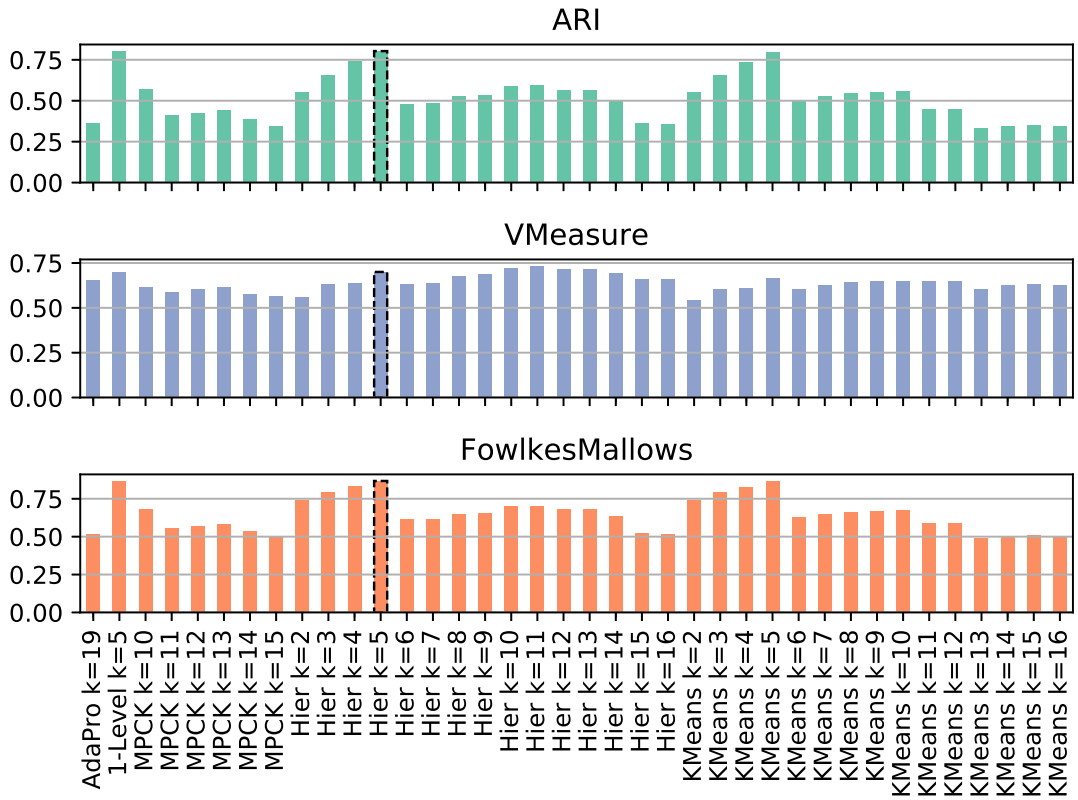
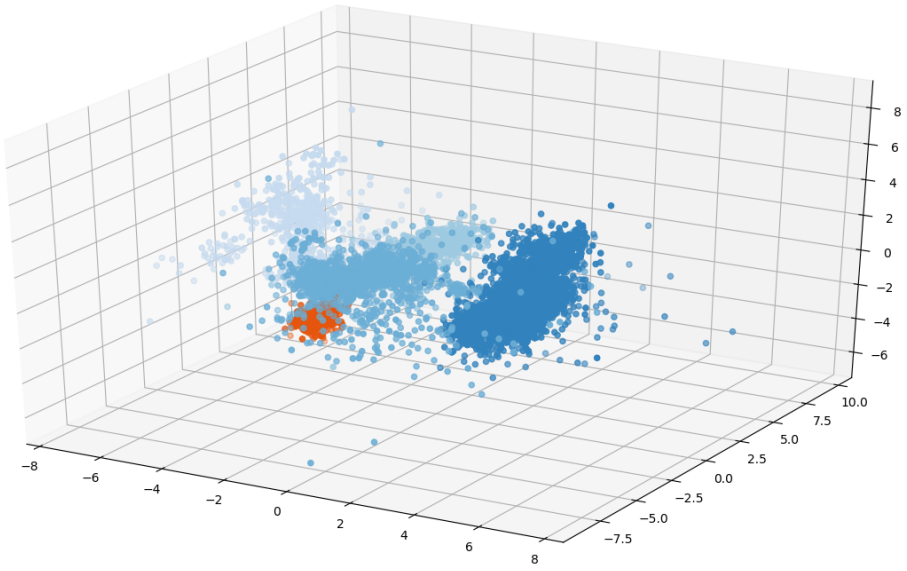


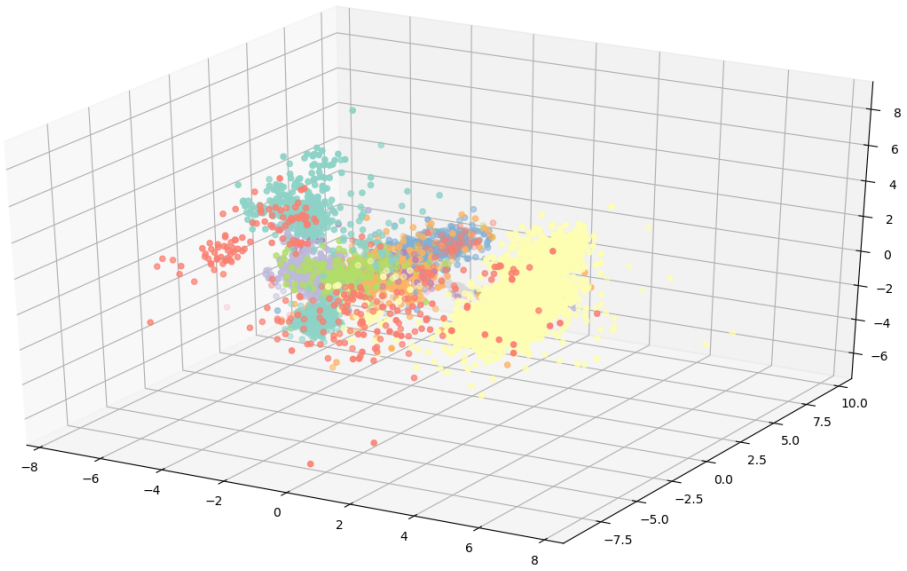
Figura 6.8: Andamento risultati su *frogs (Species)* con 1% di prototipi per classe

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=19	**0.361**	0.546	0.520	**0.825**	0.657
1-Level k=5	**0.802**	**0.808**	**0.866**	0.617	**0.700**
MPCK k=10	0.568	0.575	0.680	0.700	0.619
Hier k=5	**0.803**	**0.808**	**0.867**	**0.618**	**0.700**
KMeans k=5	0.798	0.768	0.864	0.590	0.667

Tabella 6.4: Risultati migliori su *frogs (Species)* con 1% di prototipi per classe



(a) *cluster* generati da 1-Level-AdaPro



(b) Classi del *ground truth*

Figura 6.9: PCA3D di *frogs (Species)*

6.1.4 *iris* con 1% di prototipi

In Figura 6.10 possiamo osservare gli andamenti dei vari algoritmi sulla collezione di dati *iris*. Consultando la Tabella 6.5 notiamo che i migliori risultati sono ottenuti da K-Means con $k = 2$: in particolare, notiamo l'altissima *Completeness*. Probabilmente questa collezione di dati ha una distribuzione per la quale risulta essere maggiormente adatta ad un algoritmo come K-Means anziché ad uno come il *clustering* gerarchico, che non ottiene risultati altrettanto buoni anche con $k = 2$.
Tra gli altri algoritmi, anche AdaPro, 1-Level-AdaPro e il *clustering* gerarchico con $k = 2$ ottengono risultati paragonabili. Si nota però come AdaPro e 1-Level-AdaPro tendano ad avere un'*Homogeneity* dei *cluster* più elevata, penalizzando, anche se non troppo, la *Completeness*.
Infine, MPCK-Means ottiene i risultati meno buoni, nonostante la sua logica semi-supervisionata.

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=3	0.604	0.687	0.743	**0.648**	0.667
1-Level k=3	0.604	0.687	0.743	**0.648**	0.667
MPCK k=3	0.528	0.516	0.685	0.510	0.513
Hier k=2	0.541	0.945	0.757	0.543	0.690
KMeans k=2	**0.568**	**1.0**	**0.771**	**0.580**	**0.734**

Tabella 6.5: Risultati migliori su *iris* con 1% di prototipi per classe

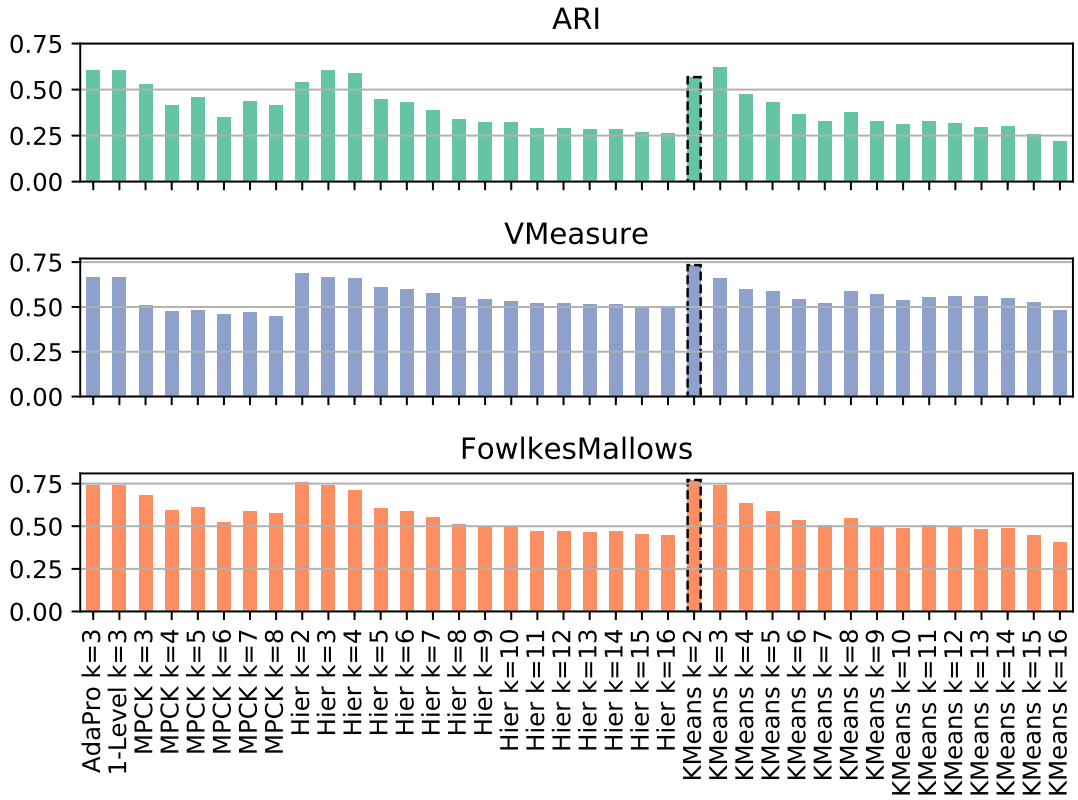


Figura 6.10: Andamento risultati su *iris* con 1% di prototipi per classe

6.1.5 *wine* con 1% di prototipi

Per quanto riguarda la collezione di dati *wine*, possiamo notare osservando il grafico in Figura 6.11 che i risultati migliori si hanno sempre in corrispondenza di $k = 3$, che corrisponde al numero di classi nella collezione. Nello specifico (vedi Tabella 6.6), le *performance* migliori sono ottenute da K-Means con $k = 3$. Anche in questo caso i risultati di AdaPro non si discostano molto dal risultato migliore, risultando ad ogni modo superiori a quelli ottenuti da MPCK-Means. Notiamo anche che i risultati di AdaPro e 1-Level-AdaPro risultano in questo caso gli stessi: AdaPro produce tre *super-cluster*, che non vengono divisi ulteriormente in *sub-cluster*; 1-Level-AdaPro, invece, divide subito l'intera collezione in tre *sub-cluster*.

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=3	0.453	0.398	0.690	0.647	0.493
1-Level k=3	0.453	0.398	0.690	0.647	0.493
MPCK k=3	0.365	0.290	0.636	0.471	0.359
Hier k=3	0.453	0.398	0.690	0.647	0.493
KMeans k=3	**0.553**	**0.491**	**0.752**	**0.795**	**0.607**

Tabella 6.6: Risultati migliori su *wine* con 1% di prototipi per classe

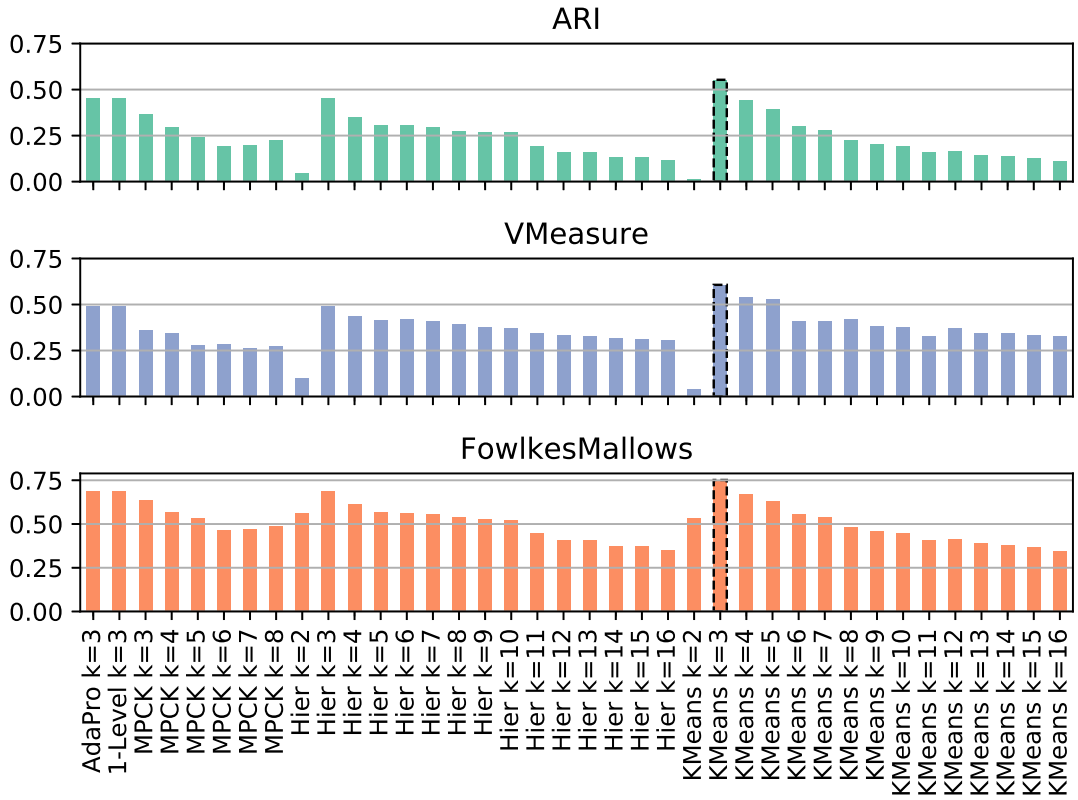


Figura 6.11: Andamento risultati su *wine* con 1% di prototipi per classe

6.1.6 *tranfusion* con 1% di prototipi

Nel caso della collezione di dati *transfusion*, notiamo subito dal grafico in Figura 6.12 che i risultati relativi a *ARI* e *V-Measure* sono molto bassi. In Tabella 6.7 notiamo che a questi bassi valori di *ARI* e *V-Measure*, corrispondono bassi valori di *Completeness* e *Homogeneity* (sappiamo infatti che i due indici precedenti sono calcolati proprio a partire da questi ultimi due). Tali valori sono probabilmente dovuti ad una difficoltà da parte dell'algoritmo ad ottenere una buona suddivisione della collezione di dati. A conferma di questo, in Figura 6.13 troviamo i dati rappresentati nelle prime 3 componenti principali (PCA3D): si vede che i dati appartenenti alle diverse classi sembrano essere tutti confusi insieme e non ben distinti. Il fenomeno non intacca invece il *Fowlkes-Mallows*.

In ogni caso, i risultati di *Fowlkes-Mallows* migliori sono raggiunti da K-Means con $k = 2$ (che corrisponde alla cardinalità dell'attributo di classe).

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=3	0.036	0.013	0.606	0.016	0.014
1-Level k=5	0.043	0.030	0.456	0.073	0.042
MPCK k=2	0.016	0.063	0.579	0.079	0.070
Hier k=2	0.030	0.006	0.608	0.007	0.006
KMeans k=2	**0.057**	0.011	**0.665**	0.011	**0.011**

Tabella 6.7: Risultati migliori su *transfusion* con 1% di prototipi per classe

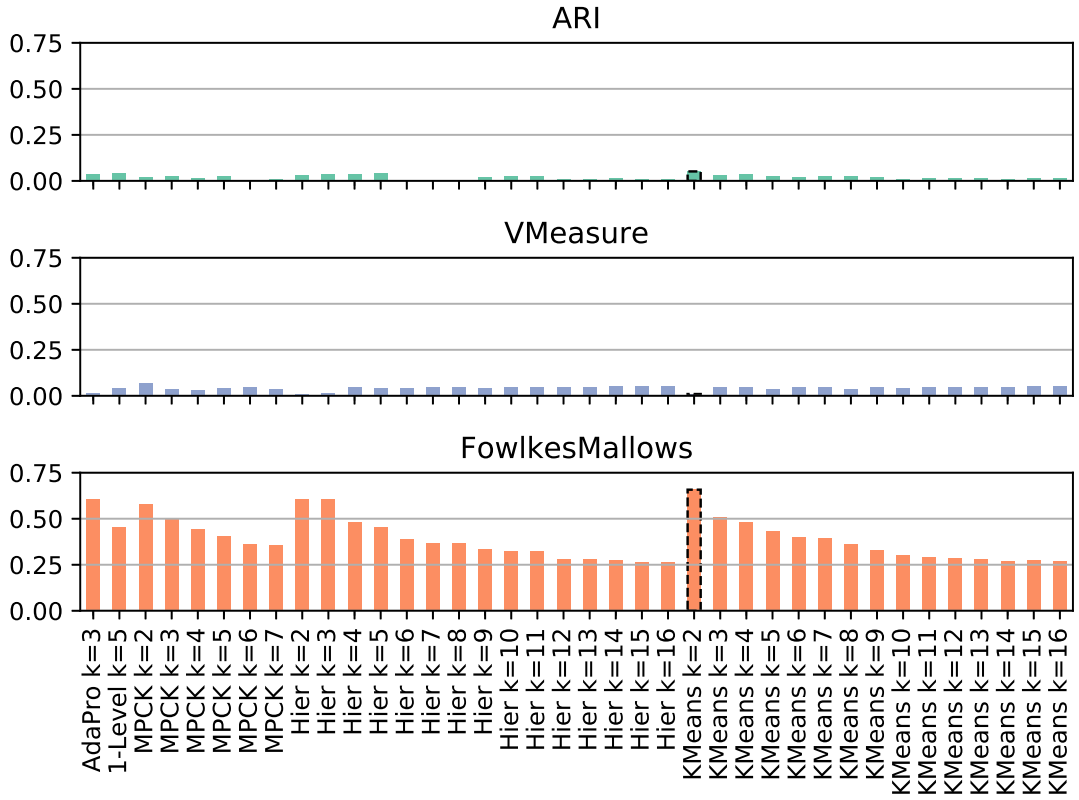


Figura 6.12: Andamento risultati su *transfusion* con 1% di prototipi per classe

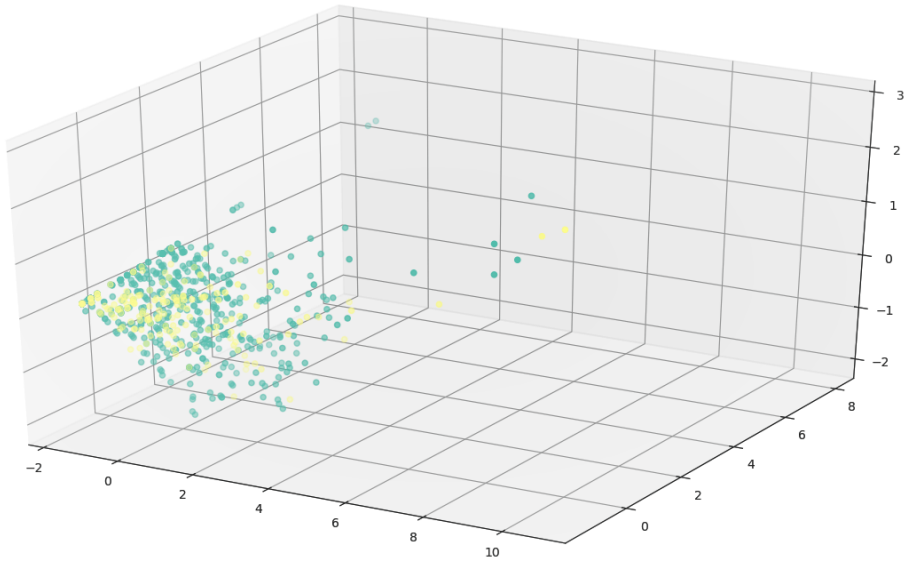


Figura 6.13: PCA3D di *transfusion*. Ogni colore rappresenta una classe del *ground truth*

6.1.7 *electrical grid stability* con 1% di prototipi

Anche le analisi di *electrical grid stability* sono caratterizzate da un basso valore delle misure *ARI* e *V-Measure* (vedi Figura 6.14). Anche in questo caso, i bassi valori si possono spiegare con una difficoltà a suddividere la collezione in *cluster*. A prova di questo, si veda la Figura 6.15 nella quale sono rappresentate le classi della collezione nello spazio dei tre componenti principali (PCA3D): tutti i dati sembrano essere ammassati in un grande agglomerato globulare che rende davvero difficile distinguere diversi *cluster*. Invece, il valore di *Fowlkes-Mallows* risulta massimo per le due versioni di AdaPro e per il *clustering* gerarchico con $k = 2$ (vedi Tabella 6.8). Anche qui, AdaPro e 1-Level-AdaPro ottengono risultati migliori di MPCK-Means (per quanto possiamo notare dal *Fowlkes-Mallows*).

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=2	**0.030**	0.051	**0.544**	0.052	**0.051**
1-Level k=2	**0.030**	0.051	**0.544**	0.052	**0.051**
MPCK k=2	0.0001	0.0002	0.519	0.0002	0.0002
Hier k=2	**0.030**	0.051	**0.544**	0.052	**0.051**
KMeans k=2	0.036	0.029	0.536	0.030	0.029

Tabella 6.8: Risultati migliori su *electrical grid stability* con 1% di prototipi per classe

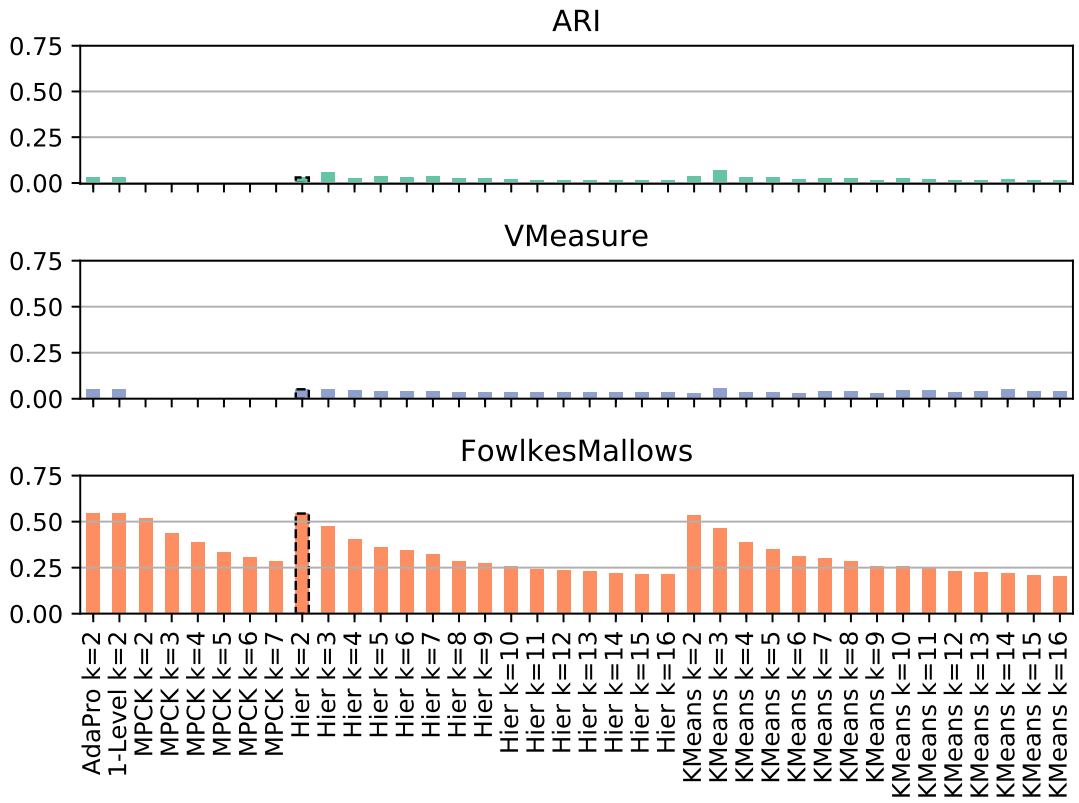


Figura 6.14: Andamento risultati su *electrical grid stability* con 1% di prototipi per classe

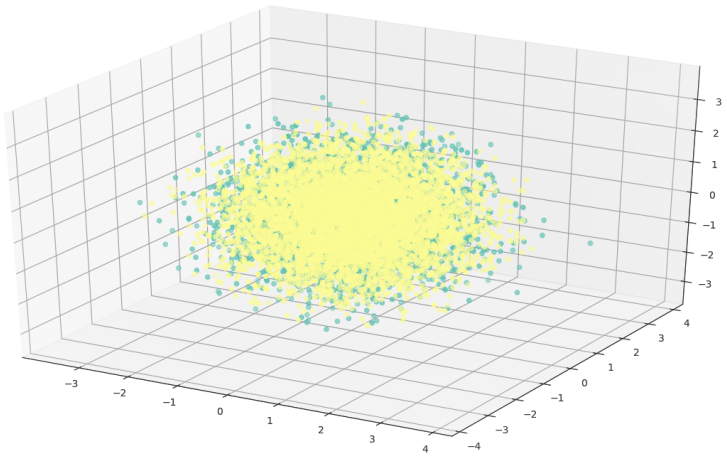


Figura 6.15: PCA3D di *electrical grid stability*. Ogni colore rappresenta una classe del *ground truth*

6.1.8 *seeds* con 1% di prototipi

La collezione di dati *seeds* ottiene punteggio massimo con il *clustering* gerarchico con $k = 2$ (vedi Figura 6.16), anche se la cardinalità delle classi appartenenti alla collezione ammonta a 3. Anche K-Means con $k = 2$ ottiene dei risultati molto buoni. AdaPro e 1-Level-AdaPro sembra invece che diano maggiore importanza ad ottenere *cluster* più puri, con alta *Homogeneity*, generando 4 *cluster* (vedi Tabella 6.9). Quanto a *performance*, in coda troviamo MPCK-Means.

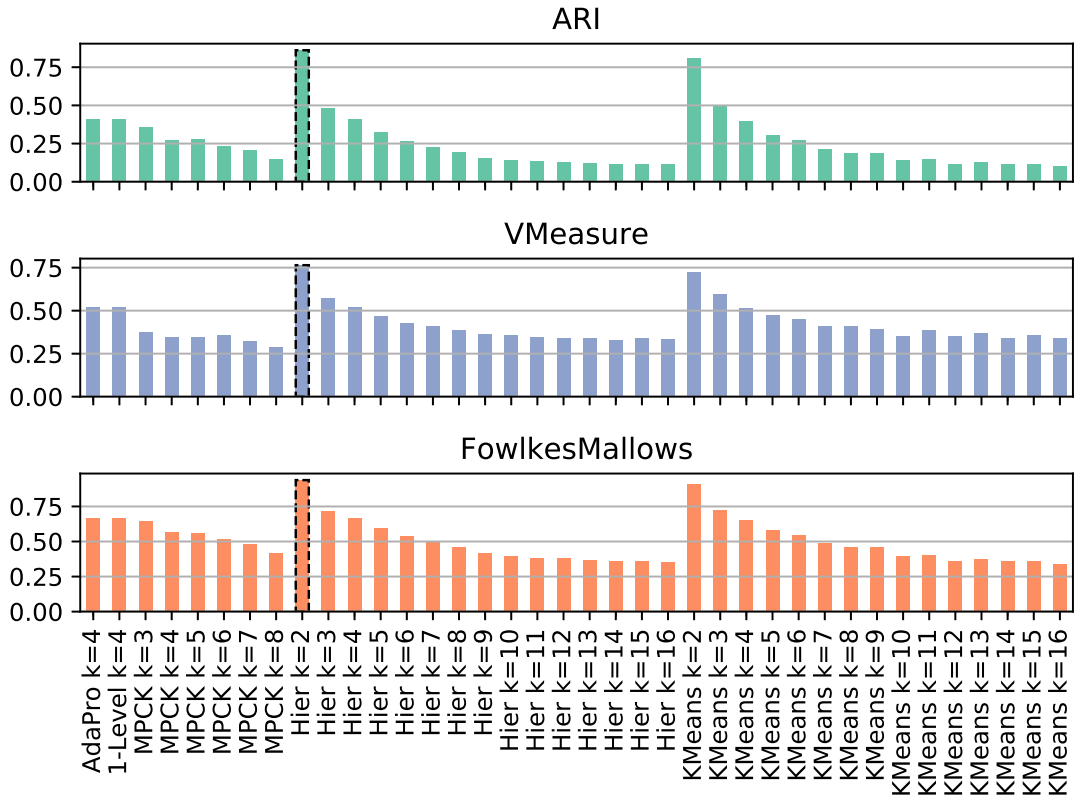


Figura 6.16: Andamento risultati su *seeds* con 1% di prototipi per classe

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=4	0.408	0.387	0.666	0.791	0.520
1-Level k=4	0.408	0.387	0.666	0.791	0.520
MPCK k=3	0.356	0.295	0.645	0.506	0.373
Hier k=2	**0.861**	0.762	**0.938**	0.766	**0.764**
KMeans k=2	0.806	0.710	0.911	0.737	0.723

Tabella 6.9: Risultati migliori su *seeds* con 1% di prototipi per classe

6.1.9 *pop failures* con 1% di prototipi

In Figura 6.17 troviamo l'andamento dei risultati del *clustering* effettuato sulla collezione di dati *pop failures*. Le *performance* migliori sono ottenute da AdaPro, MPCK-Means, *clustering* gerarchico e K-Means con $k = 2$.
In Tabella 6.10 osserviamo che i risultati migliori sono ottenuti dal *clustering* gerarchico, che raggiunge un valore di *Fowlkes-Mallows* pari a 0.704. Sono praticamente paragonabili anche i risultati di AdaPro che, se da una parte ottiene un punteggio di *Fowlkes-Mallows* leggermente inferiore, dall'altra parte guadagna in termini di *ARI* e *V-Measure*.
I bassissimi punteggi ottenuti per *ARI* e *V-Measure* sono spiegabili nella difficoltà ad individuare i *cluster* a causa della distribuzione dei dati. A tal proposito, si veda la Figura 6.18 nella quale sono rappresentate le classi della collezione nelle prime tre componenti principali.

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=3	**0.027**	0.008	**0.682**	0.022	**0.011**
1-Level k=5	-0.0003	0.011	0.431	0.059	0.0178
MPCK k=2	0.001	0.005	0.653	0.012	0.007
Hier k=2	**0.016**	0.002	**0.704**	0.004	**0.002**
KMeans k=2	0.029	0.065	0.664	0.162	0.092

Tabella 6.10: Risultati migliori su *pop failures* con 1% di prototipi per classe

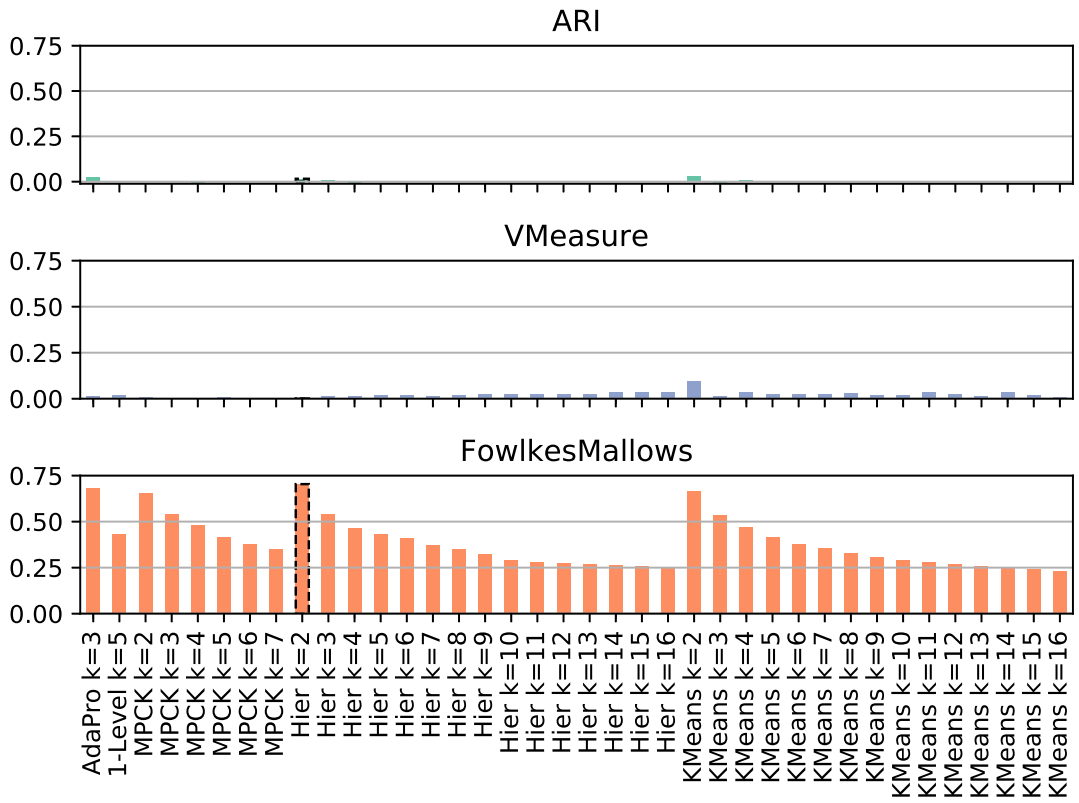


Figura 6.17: Andamento risultati su *pop failures* con 1% di prototipi per classe

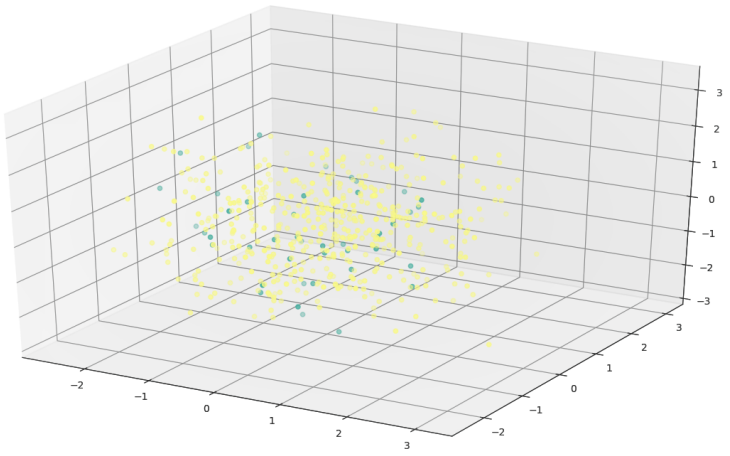


Figura 6.18: PCA3D di *pop failures*. Ogni colore rappresenta una classe del *ground truth*

6.1.10 *wdbc* con 1% di prototipi

Prendiamo in esame la collezione di dati *wdbc*. In Figura 6.19 troviamo l'andamento dei risultati dei vari algoritmi di *clustering*. Ad ottenere il risultato migliore è K-Means con $k = 2$, insieme al *clustering* gerarchico con $k = 2$ (vedi Tabella 6.11).

In questo caso, notiamo anche che i risultati ottenuti da 1-Level-AdaPro risultano essere maggiori della versione completa AdaPro. Questo si può spiegare col fatto che 1-Level-AdaPro genera 3 *cluster*, contro i 5 generati da AdaPro: il numero reale di classi era invece 2.

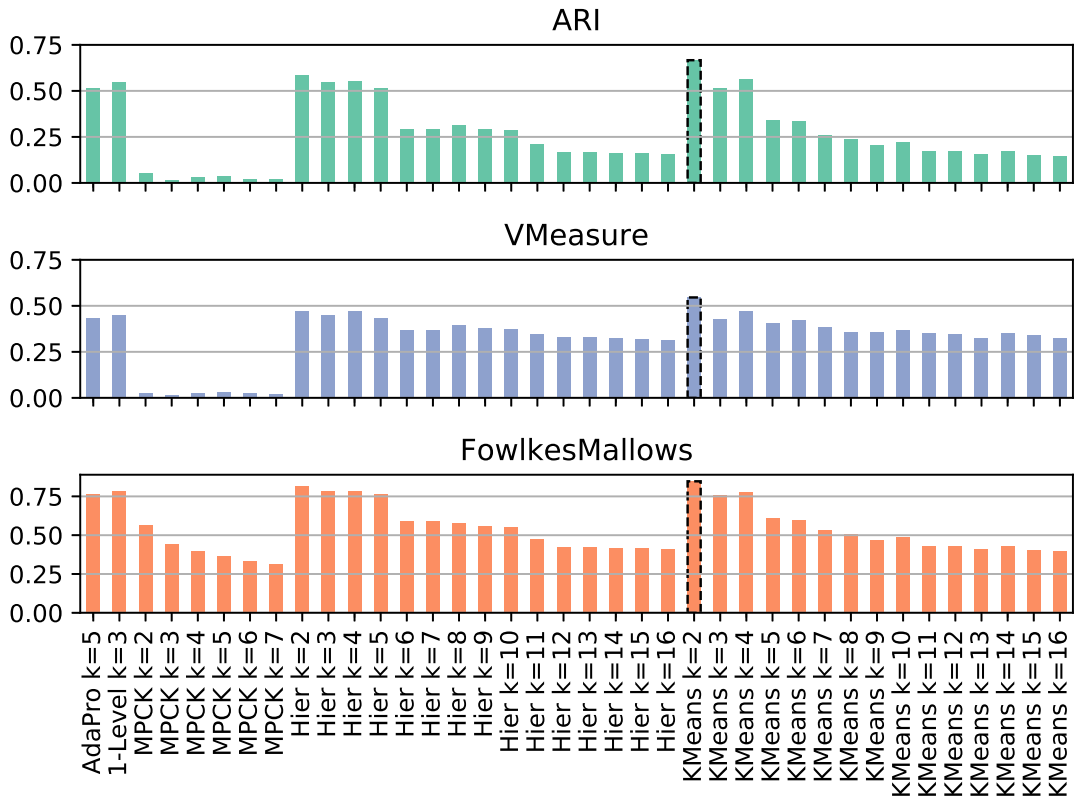


Figura 6.19: Andamento risultati su *wdbc* con 1% di prototipi per classe

	ARI	Completen.	FMI	Homogen.	V-Measure
AdaPro k=5	0.513	0.354	0.762	0.560	0.434
1-Level k=3	0.546	0.401	0.783	0.512	0.450
MPCK k=2	0.052	0.024	0.566	0.023	0.024
Hier k=2	**0.586**	0.479	**0.813**	0.457	**0.467**
KMeans k=2	**0.678**	0.571	**0.853**	0.551	**0.561**

Tabella 6.11: Risultati migliori su *wdbc* con 1% di prototipi per classe

6.1.11 *seismic bumps* con 1% di prototipi

I risultati del *clustering* effettuato sulla collezione di dati *seismic bumps* (vedi Figura 6.20) mostrano che i risultati migliori si hanno per AdaPro, *clustering* gerarchico e K-Means con $k = 2$.

Anche per questa collezione i valori di *ARI* e *V-Measure* risultano particolarmente bassi, come lo sono anche la *Homogeneity* e la *Completeness* sulla base delle quali vengono calcolati i precedenti. Attribuiamo questo alla difficoltà nell'individuare *cluster* nella collezione di dati (si veda la rappresentazione nelle tre componenti principali in Figura 6.21)

Osservando la Tabella 6.12 vediamo che il miglior risultato è stato ottenuto da AdaPro e dal *clustering* gerarchico con $k = 2$, raggiungendo un valore di *Fowlkes-Mallows* di 0.888. K-Means invece, seppur raggiungendo un valore leggermente inferiore di *Fowlkes-Mallows*, è l'algoritmo che nel complesso ottiene i risultati migliori, anche considerando *ARI* e *V-Measure*.

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=2	0.069	0.010	**0.888**	0.010	0.010
1-Level k=9	0.031	0.014	0.465	0.099	0.0239
MPCK k=2	0.105	0.026	0.792	0.056	0.035
Hier k=2	0.069	0.010	**0.888**	0.010	0.010
KMeans k=2	**0.181**	0.053	**0.844**	0.094	**0.068**

Tabella 6.12: Risultati migliori su *seismic bumps* con 1% di prototipi per classe

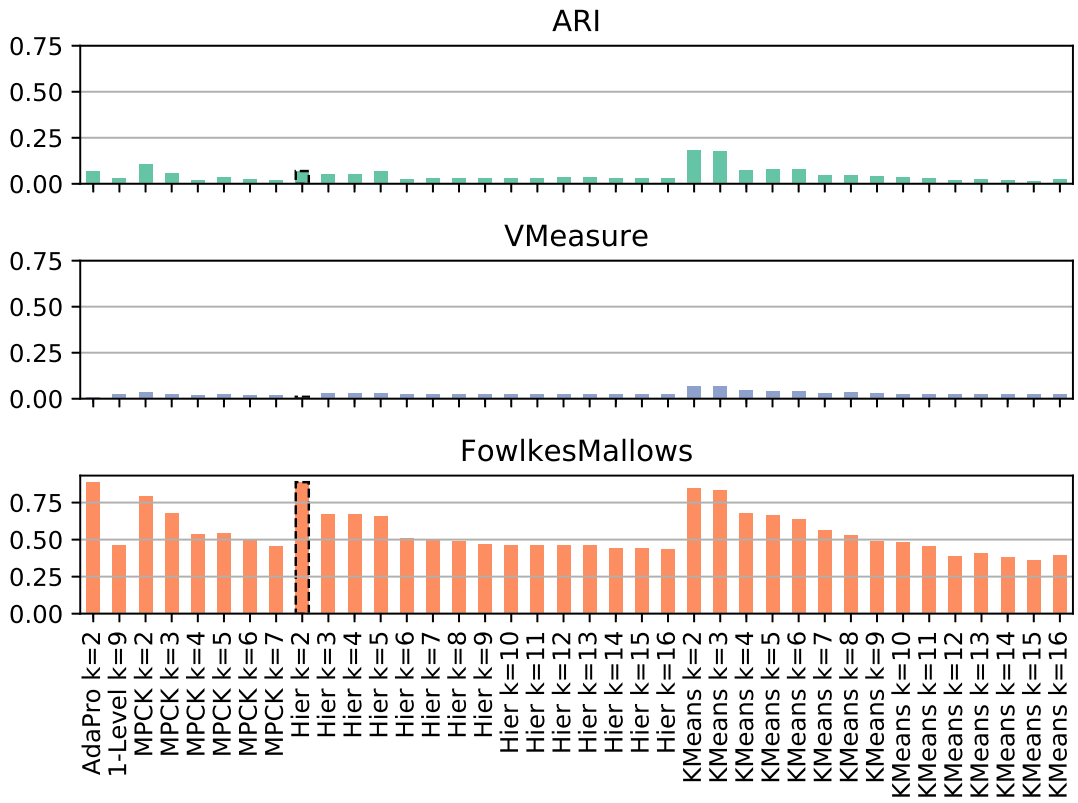


Figura 6.20: Andamento risultati su *seismic bumps* con 1% di prototipi per classe

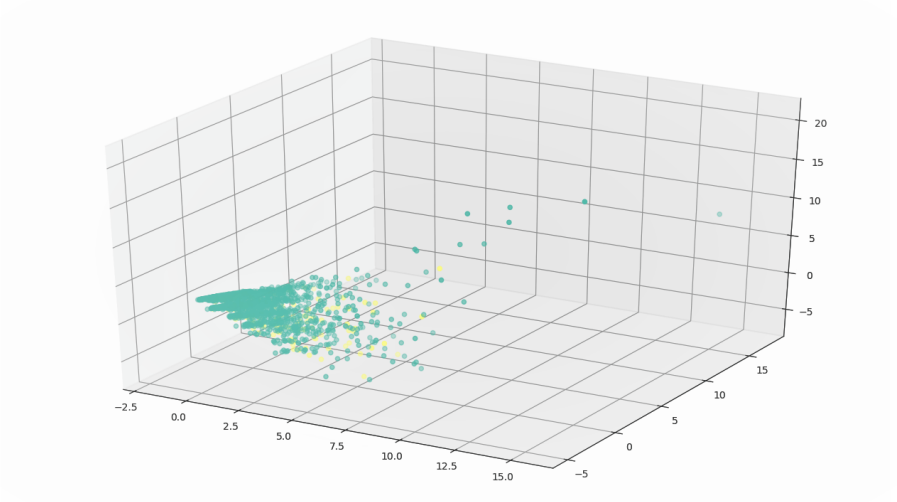


Figura 6.21: PCA3D di *seismic bumps*. Ogni colore rappresenta una classe del *ground truth*

6.1.12 *avila* con 1% di prototipi

In Figura 6.22 possiamo osservare l'andamento dei risultati per la collezione di dati *avila*. Sebbene il risultato maggiore in termini di *Fowlkes-Mallows* lo abbia il *clustering* gerarchico e K-Means con $k = 2$, tenendo conto di tutti e tre gli indici che abbiamo usato, notiamo che gli algoritmi ad ottenere i risultati migliori sono 1-Level-AdaPro, che trova 11 *cluster* diversi e il *clustering* gerarchico con $k = 11$ (vedi Tabella 6.13). Questi risultati assumono un significato maggiore se pensiamo che la collezione aveva ben 12 classi diverse: gli algoritmi appena citati che hanno individuato 11 *cluster* diversi hanno un significato maggiore rispetto agli altri che hanno alti valori di *Fowlkes-Mallows* e bassi valori di *ARI* e *V-Measure* (parliamo di AdaPro, *clustering* gerarchico e K-Means con $k = 2$). In un caso reale però, solo 1-Level-AdaPro avrebbe potuto restituirmi il risultato migliore: se non avessimo avuto a disposizione le etichette di classe per l'intera collezione di dati, non avremmo saputo quale tra gli esperimenti fatti col *clustering* gerarchico scegliere.
Infine, in Figura 6.23 troviamo la collezione rappresentata nelle sue tre componenti principali.

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=2	0	0.081	**0.480**	0	0
1-Level k=11	**0.1176**	0.218	**0.360**	0.190	**0.203**
MPCK k=12	0	0.001	0.170	0.002	0.002
Hier k=2	0	0.081	0.480	0	0
Hier k=11	**0.1172**	0.217	**0.360**	0.189	**0.202**
KMeans k=2	0	0.081	**0.480**	0	0
KMeans k=12	0.063	0.128	0.241	0.144	0.136

Tabella 6.13: Risultati migliori su *avila* con 1% di prototipi per classe

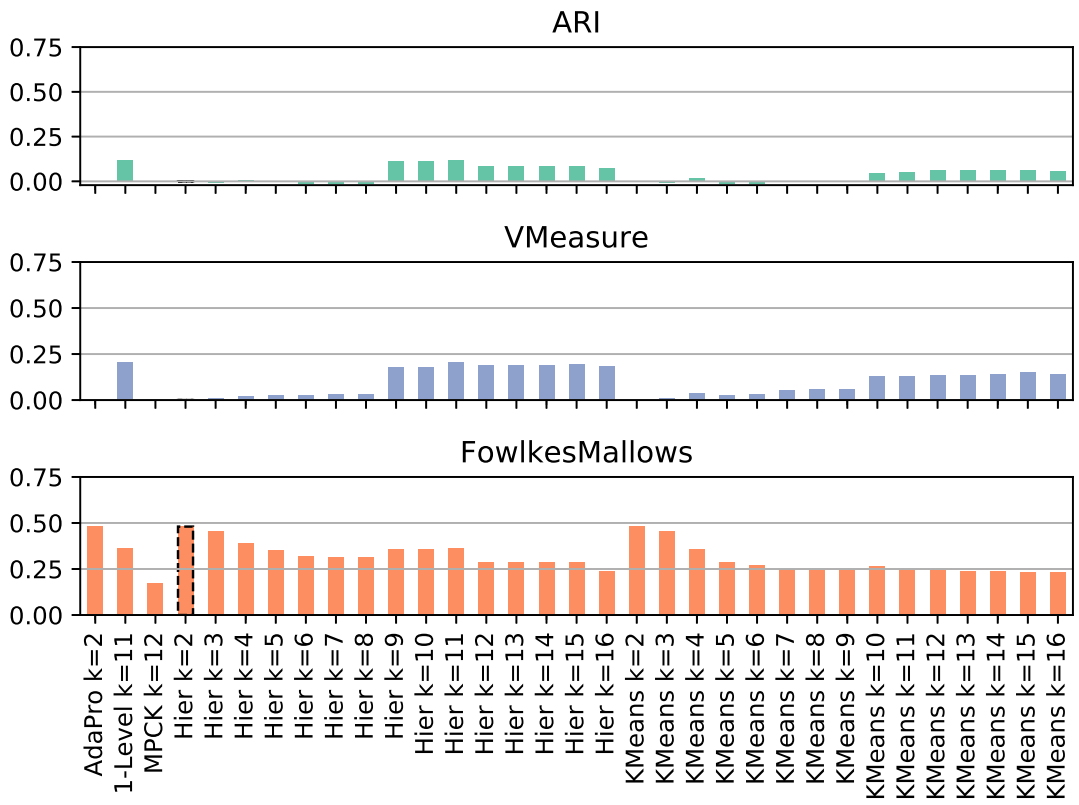


Figura 6.22: Andamento risultati su *avila* con 1% di prototipi per classe

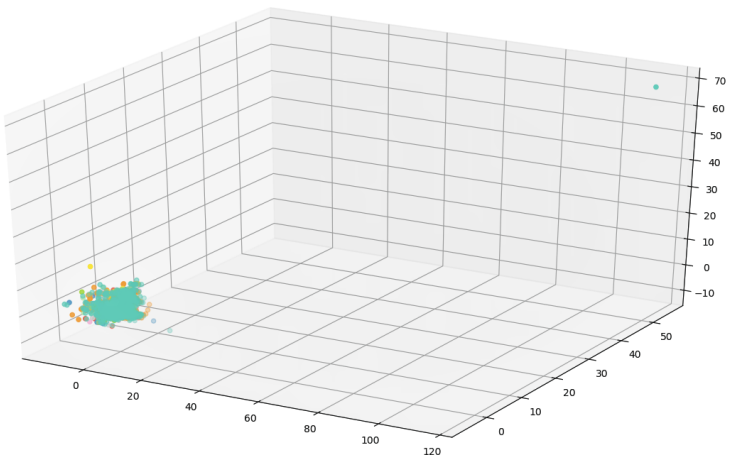


Figura 6.23: PCA3D di *avila*. Ogni colore rappresenta una classe del *ground truth*

6.1.13 *data banknote authentication* con 1% di prototipi

Osservando i risultati in Figura 6.24 e in Tabella 6.14 possiamo notare che i due algoritmi che ottengono i risultati migliori sono 1-Level-AdaPro, il *clustering* gerarchico con $k = 5$.

Inoltre, notiamo in Figura 6.24 che tutti gli algoritmi tendono ad ottenere valori bassi di *ARI* e *V-Measure* (e quindi di *Homogeneity* e *Completeness*) per bassi valori di k, ottenendo una forte crescita per $k \geq 5$. Probabilmente questo è dovuto alla distribuzione dei dati: anche se nominalmente la collezione di dati ha solo due possibili classi, le distanze geometriche tra i dati sembrano dire che in realtà i dati siano ben divisi con 5 o più classi. Secondo AdaPro, che ottiene i risultati più alti in termini di *Homogeneity*, i dati sono ben divisi in 9 *cluster*.

Infine, in Figura 6.25 troviamo i *cluster* generati da 1-Level-AdaPro e le classi del *ground truth* della collezione rappresentati nelle tre componenti principali.

	ARI	Complet.	FMI	Homogen.	V-Measure
AdaPro k=9	0.311	0.266	0.563	**0.725**	**0.389**
1-Level k=5	**0.331**	**0.275**	**0.584**	**0.621**	**0.381**
MPCK k=5	0.204	0.188	0.481	0.437	0.263
Hier k=5	**0.331**	**0.275**	**0.584**	**0.621**	**0.381**
KMeans k=2	0.013	0.011	0.509	0.011	0.011
KMeans k=5	0.212	0.170	0.491	0.388	0.237

Tabella 6.14: Risultati migliori su *data banknote authentication* con 1% di prototipi per classe

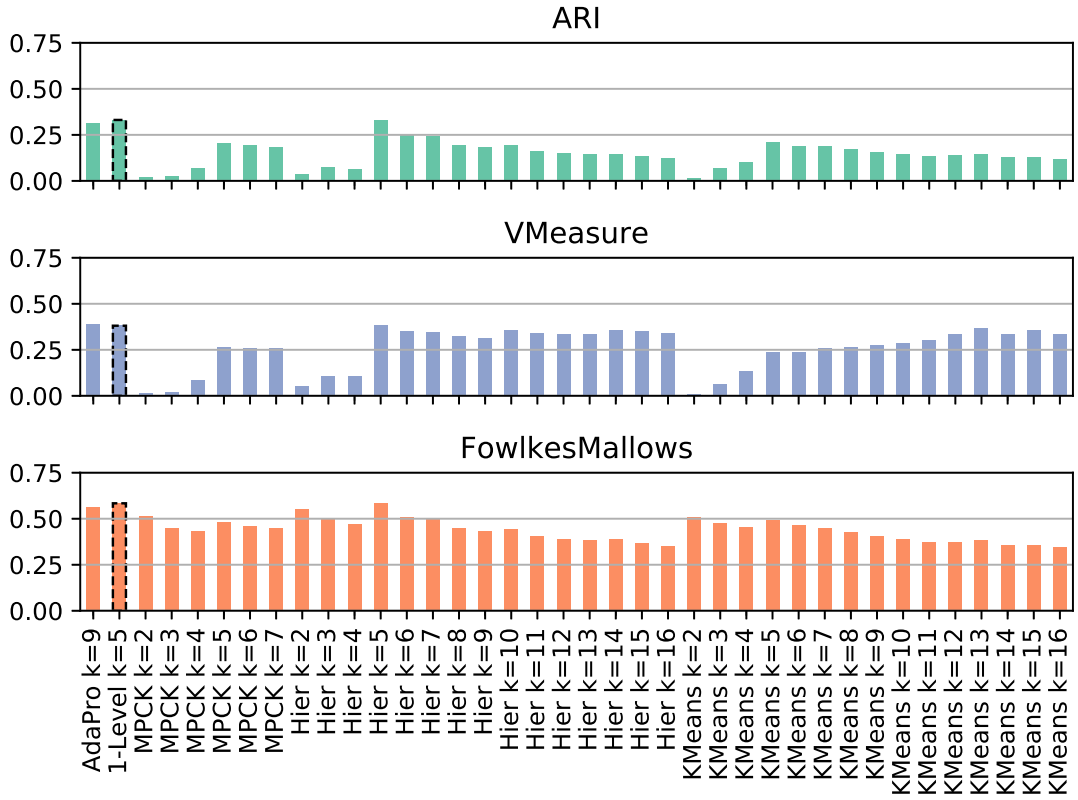
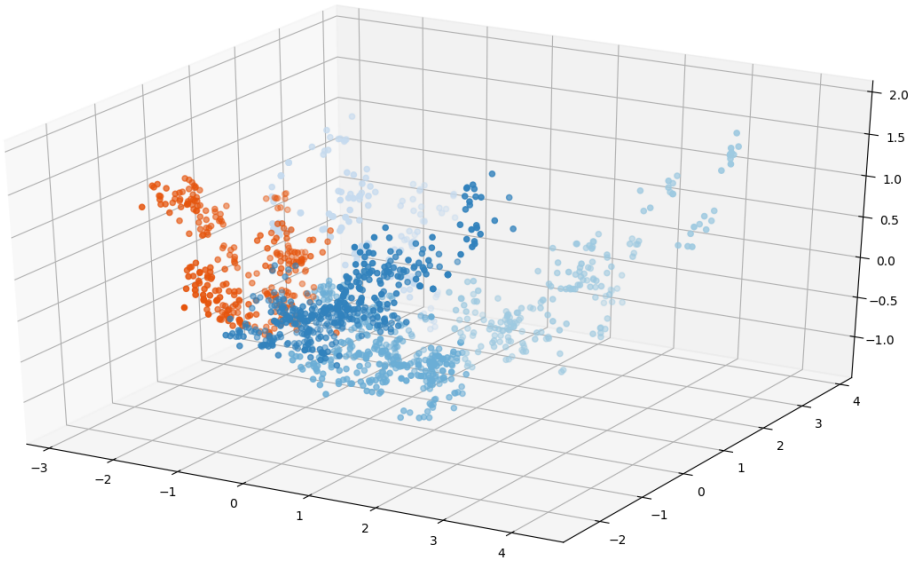
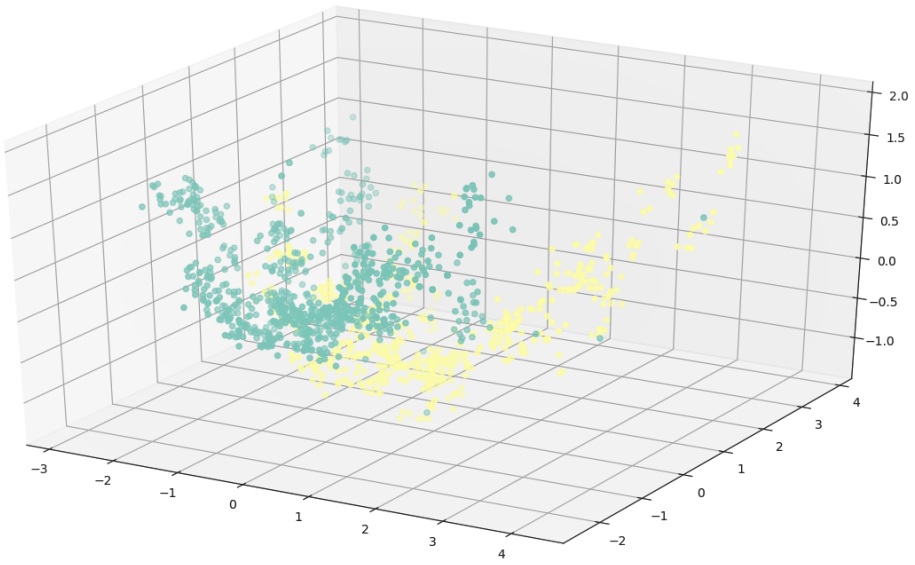


Figura 6.24: Andamento risultati su *data banknote* con 1% di prototipi per classe



(a) *cluster* generati da 1-Level-AdaPro



(b) Classi del *ground truth*

Figura 6.25: PCA3D di *data banknote authentication*

6.2 Tempistiche

Nel seguente paragrafo sono riportate le tabelle contenenti le tempistiche richieste da ogni algoritmo per eseguire le analisi su ogni collezione di dati.
Osservando la tendenza generale su tutte le collezioni, vediamo che gli algoritmi più veloci sono K-Means e il *clustering* gerarchico (ciò rispetta le aspettative, dal momento che sono algoritmi non supervisionati). Subito dopo troviamo AdaPro insieme alla sua variante 1-Level-AdaPro. Come ultimo classificato, sempre in termini di tempi di esecuzione, abbiamo MPCK-Means.
MPCK-Means richiede il maggior tempo soprattutto quando la cardinalità delle collezioni aumenta, come nel caso di *avila* (vedi Tabella 6.26).
AdaPro, anche nei casi di collezioni molto grandi, riesce a restare in tempi molto minori; si osservino i risultati per avila in Tabella 6.26: mentre MPCK-Means ha richiesto 22143 secondi (cioè più di 6 ore), AdaPro ha completato le analisi in 368 secondi (poco più di 6 minuti). A questi tempi minori, come abbiamo potuto constatare nel paragrafo precedente, corrispondono anche risultati migliori, facendo di AdaPro un algoritmo semi-supervisionato veloce e performante.

frogs (Family) con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	55.361	Hier k=13	2.093
1-level	64.624	Hier k=14	2.053
MPCK k=4	2709.844	Hier k=15	2.105
MPCK k=5	2697.888	Hier k=16	2.115
MPCK k=6	2695.323	Kmeans k=2	0.110
MPCK k=7	2716.935	Kmeans k=3	0.140
MPCK k=8	2703.145	Kmeans k=4	0.157
MPCK k=9	2721.901	Kmeans k=5	0.171
Hier k=2	1.949	Kmeans k=6	0.222
Hier k=3	2.082	Kmeans k=7	0.234
Hier k=4	2.128	Kmeans k=8	0.250
Hier k=5	2.089	Kmeans k=9	0.313
Hier k=6	2.102	Kmeans k=10	0.328
Hier k=7	2.075	Kmeans k=11	0.359
Hier k=8	2.156	Kmeans k=12	0.407
Hier k=9	2.105	Kmeans k=13	0.408
Hier k=10	2.088	Kmeans k=14	0.455
Hier k=11	2.105	Kmeans k=15	0.458
Hier k=12	2.096	Kmeans k=16	0.463

Tabella 6.15: Tempi di esecuzione (in secondi) per la collezione *frogs (Family)* con 1% di prototipi per classe

frogs (Genus) con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	60.105	Hier k=13	2.093
1-level	64.565	Hier k=14	2.053
MPCK k=8	2728.702	Hier k=15	2.105
MPCK k=9	2737.176	Hier k=16	2.115
MPCK k=10	2729.513	Kmeans k=2	0.110
MPCK k=11	2757.416	Kmeans k=3	0.140
MPCK k=12	2758.743	Kmeans k=4	0.157
MPCK k=13	2772.742	Kmeans k=5	0.171
Hier k=2	1.949	Kmeans k=6	0.222
Hier k=3	2.082	Kmeans k=7	0.234
Hier k=4	2.128	Kmeans k=8	0.250
Hier k=5	2.089	Kmeans k=9	0.313
Hier k=6	2.102	Kmeans k=10	0.328
Hier k=7	2.075	Kmeans k=11	0.359
Hier k=8	2.156	Kmeans k=12	0.407
Hier k=9	2.105	Kmeans k=13	0.408
Hier k=10	2.088	Kmeans k=14	0.455
Hier k=11	2.105	Kmeans k=15	0.458
Hier k=12	2.096	Kmeans k=16	0.463

Tabella 6.16: Tempi di esecuzione (in secondi) per la collezione *frogs (Genus)* con 1% di prototipi per classe

frogs (Species) con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	60.725	Hier k=13	2.093
1-level	64.639	Hier k=14	2.053
MPCK k=10	2626.594	Hier k=15	2.105
MPCK k=11	2599.322	Hier k=16	2.115
MPCK k=12	2616.308	Kmeans k=2	0.110
MPCK k=13	2639.495	Kmeans k=3	0.140
MPCK k=14	2634.202	Kmeans k=4	0.157
MPCK k=15	2641.600	Kmeans k=5	0.171
Hier k=2	1.949	Kmeans k=6	0.222
Hier k=3	2.082	Kmeans k=7	0.234
Hier k=4	2.128	Kmeans k=8	0.250
Hier k=5	2.089	Kmeans k=9	0.313
Hier k=6	2.102	Kmeans k=10	0.328
Hier k=7	2.075	Kmeans k=11	0.359
Hier k=8	2.156	Kmeans k=12	0.407
Hier k=9	2.105	Kmeans k=13	0.408
Hier k=10	2.088	Kmeans k=14	0.455
Hier k=11	2.105	Kmeans k=15	0.458
Hier k=12	2.096	Kmeans k=16	0.463

Tabella 6.17: Tempi di esecuzione (in secondi) per la collezione *frogs (Species)* con 1% di prototipi per classe

iris con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	5.176	Hier k=13	0.002
1-level	4.540	Hier k=14	0.001
MPCK k=3	2.059	Hier k=15	0.002
MPCK k=4	3.701	Hier k=16	0.005
MPCK k=5	4.045	Kmeans k=2	0.008
MPCK k=6	3.881	Kmeans k=3	0.039
MPCK k=7	3.951	Kmeans k=4	0.047
MPCK k=8	4.068	Kmeans k=5	0.038
Hier k=2	0.004	Kmeans k=6	0.057
Hier k=3	0.006	Kmeans k=7	0.049
Hier k=4	0.007	Kmeans k=8	0.061
Hier k=5	0.006	Kmeans k=9	0.049
Hier k=6	0.007	Kmeans k=10	0.084
Hier k=7	0.003	Kmeans k=11	0.073
Hier k=8	0.004	Kmeans k=12	0.075
Hier k=9	0.003	Kmeans k=13	0.102
Hier k=10	0.006	Kmeans k=14	0.098
Hier k=11	0.001	Kmeans k=15	0.095
Hier k=12	0.002	Kmeans k=16	0.141

Tabella 6.18: Tempi di esecuzione (in secondi) per la collezione iris con 1% di prototipi per classe

wine con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	2.140	Hier k=13	0.016
1-level	1.982	Hier k=14	0
MPCK k=3	1.895	Hier k=15	0
MPCK k=4	2.025	Hier k=16	0.016
MPCK k=5	1.980	Kmeans k=2	0.032
MPCK k=6	2.097	Kmeans k=3	0.015
MPCK k=7	2.145	Kmeans k=4	0.016
MPCK k=8	1.511	Kmeans k=5	0.031
Hier k=2	0.015	Kmeans k=6	0.033
Hier k=3	0	Kmeans k=7	0.031
Hier k=4	0	Kmeans k=8	0.031
Hier k=5	0.016	Kmeans k=9	0.047
Hier k=6	0	Kmeans k=10	0.031
Hier k=7	0	Kmeans k=11	0.047
Hier k=8	0.016	Kmeans k=12	0.031
Hier k=9	0	Kmeans k=13	0.047
Hier k=10	0.015	Kmeans k=14	0.048
Hier k=11	0	Kmeans k=15	0.031
Hier k=12	0	Kmeans k=16	0.047

Tabella 6.19: Tempi di esecuzione (in secondi) per la collezione *wine* con 1% di prototipi per classe

transfusion con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	7.805	Hier k=13	0.016
1-level	6.165	Hier k=14	0.031
MPCK k=2	19.927	Hier k=15	0.015
MPCK k=3	29.052	Hier k=16	0.016
MPCK k=4	29.469	Kmeans k=2	0.031
MPCK k=5	29.517	Kmeans k=3	0.047
MPCK k=6	29.758	Kmeans k=4	0.062
MPCK k=7	29.972	Kmeans k=5	0.047
Hier k=2	0.015	Kmeans k=6	0.063
Hier k=3	0.013	Kmeans k=7	0.062
Hier k=4	0.015	Kmeans k=8	0.063
Hier k=5	0.031	Kmeans k=9	0.062
Hier k=6	0.032	Kmeans k=10	0.075
Hier k=7	0.015	Kmeans k=11	0.094
Hier k=8	0.032	Kmeans k=12	0.890
Hier k=9	0.046	Kmeans k=13	0.109
Hier k=10	0.032	Kmeans k=14	0.108
Hier k=11	0.031	Kmeans k=15	0.770
Hier k=12	0.016	Kmeans k=16	0.078

Tabella 6.20: Tempi di esecuzione (in secondi) per la collezione *transfusion* con 1% di prototipi per classe

electrical grid stability con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	85.780	Hier k=13	3.610
1-level	100.437	Hier k=14	3.644
MPCK k=2	5041.713	Hier k=15	3.626
MPCK k=3	5041.764	Hier k=16	3.610
MPCK k=4	5026.008	Kmeans k=2	0.189
MPCK k=5	5042.506	Kmeans k=3	0.416
MPCK k=6	5055.763	Kmeans k=4	0.613
MPCK k=7	5044.647	Kmeans k=5	0.698
Hier k=2	3.689	Kmeans k=6	0.945
Hier k=3	3.638	Kmeans k=7	1.307
Hier k=4	3.640	Kmeans k=8	1.045
Hier k=5	3.617	Kmeans k=9	1.185
Hier k=6	3.618	Kmeans k=10	1.122
Hier k=7	3.643	Kmeans k=11	1.418
Hier k=8	3.645	Kmeans k=12	1.669
Hier k=9	3.612	Kmeans k=13	1.378
Hier k=10	3.631	Kmeans k=14	2.724
Hier k=11	3.621	Kmeans k=15	2.583
Hier k=12	3.642	Kmeans k=16	2.565

Tabella 6.21: Tempi di esecuzione (in secondi) per la collezione *electrical grid stability* con 1% di prototipi per classe

seeds con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	2.542	Hier k=13	0
1-level	2.135	Hier k=14	0.016
MPCK k=3	2.450	Hier k=15	0
MPCK k=4	2.435	Hier k=16	0
MPCK k=5	2.560	Kmeans k=2	0.016
MPCK k=6	2.598	Kmeans k=3	0.015
MPCK k=7	2.607	Kmeans k=4	0.031
MPCK k=8	2.782	Kmeans k=5	0.032
Hier k=2	0.016	Kmeans k=6	0.031
Hier k=3	0	Kmeans k=7	0.031
Hier k=4	0	Kmeans k=8	0.031
Hier k=5	0	Kmeans k=9	0.047
Hier k=6	0.015	Kmeans k=10	0.032
Hier k=7	0	Kmeans k=11	0.031
Hier k=8	0	Kmeans k=12	0.047
Hier k=9	0.016	Kmeans k=13	0.048
Hier k=10	0	Kmeans k=14	0.031
Hier k=11	0	Kmeans k=15	0.047
Hier k=12	0.015	Kmeans k=16	0.031

Tabella 6.22: Tempi di esecuzione (in secondi) per la collezione *seeds* con 1% di prototipi per classe

pop failures con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	4.034	Hier k=13	0.016
1-level	3.939	Hier k=14	0.015
MPCK k=2	16.678	Hier k=15	0.016
MPCK k=3	17.987	Hier k=16	0.016
MPCK k=4	18.341	Kmeans k=2	0.047
MPCK k=5	19.074	Kmeans k=3	0.047
MPCK k=6	20.137	Kmeans k=4	0.062
MPCK k=7	20.574	Kmeans k=5	0.047
Hier k=2	0.016	Kmeans k=6	0.063
Hier k=3	0.015	Kmeans k=7	0.078
Hier k=4	0	Kmeans k=8	0.062
Hier k=5	0.016	Kmeans k=9	0.078
Hier k=6	0.016	Kmeans k=10	0.079
Hier k=7	0.015	Kmeans k=11	0.078
Hier k=8	0.017	Kmeans k=12	0.093
Hier k=9	0.016	Kmeans k=13	0.094
Hier k=10	0.015	Kmeans k=14	0.078
Hier k=11	0.019	Kmeans k=15	0.094
Hier k=12	0.006	Kmeans k=16	0.094

Tabella 6.23: Tempi di esecuzione (in secondi) per la collezione *pop failures* con 1% di prototipi per classe

wdbc con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	9.587	Hier k=13	0.023
1-level	5.718	Hier k=14	0.019
MPCK k=2	15.692	Hier k=15	0.025
MPCK k=3	26.117	Hier k=16	0.020
MPCK k=4	27.724	Kmeans k=2	0.040
MPCK k=5	29.835	Kmeans k=3	0.089
MPCK k=6	32.643	Kmeans k=4	0.062
MPCK k=7	35.241	Kmeans k=5	0.078
Hier k=2	0.016	Kmeans k=6	0.078
Hier k=3	0.021	Kmeans k=7	0.110
Hier k=4	0.016	Kmeans k=8	0.078
Hier k=5	0.015	Kmeans k=9	0.109
Hier k=6	0.032	Kmeans k=10	0.094
Hier k=7	0.015	Kmeans k=11	0.094
Hier k=8	0.016	Kmeans k=12	0.125
Hier k=9	0.016	Kmeans k=13	0.117
Hier k=10	0.031	Kmeans k=14	0.094
Hier k=11	0.016	Kmeans k=15	0.125
Hier k=12	0.016	Kmeans k=16	0.127

Tabella 6.24: Tempi di esecuzione (in secondi) per la collezione *wdbc* con 1% di prototipi per classe

seismic bumps con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	15.199	Hier k=13	0.208
1-level	15.137	Hier k=14	0.186
MPCK k=2	357.649	Hier k=15	0.189
MPCK k=3	357.348	Hier k=16	0.196
MPCK k=4	341.239	Kmeans k=2	0.047
MPCK k=5	340.593	Kmeans k=3	0.062
MPCK k=6	343.688	Kmeans k=4	0.063
MPCK k=7	353.937	Kmeans k=5	0.078
Hier k=2	0.234	Kmeans k=6	0.062
Hier k=3	0.204	Kmeans k=7	0.078
Hier k=4	0.203	Kmeans k=8	0.078
Hier k=5	0.188	Kmeans k=9	0.078
Hier k=6	0.187	Kmeans k=10	0.125
Hier k=7	0.200	Kmeans k=11	0.094
Hier k=8	0.191	Kmeans k=12	0.109
Hier k=9	0.187	Kmeans k=13	0.128
Hier k=10	0.189	Kmeans k=14	0.141
Hier k=11	0.203	Kmeans k=15	0.125
Hier k=12	0.199	Kmeans k=16	0.156

Tabella 6.25: Tempi di esecuzione (in secondi) per la collezione *seismic bumps* con 1% di prototipi per classe

avila con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	368.422	Hier k=13	18.953
1-level	384.017	Hier k=14	19.011
MPCK k=2	22142.940	Hier k=15	19.153
MPCK k=3	?	Hier k=16	19.296
MPCK k=4	?	Kmeans k=2	0.250
MPCK k=5	?	Kmeans k=3	0.610
MPCK k=6	?	Kmeans k=4	0.603
MPCK k=7	?	Kmeans k=5	0.504
Hier k=2	21.114	Kmeans k=6	0.595
Hier k=3	19.337	Kmeans k=7	0.893
Hier k=4	19.051	Kmeans k=8	0.925
Hier k=5	18.979	Kmeans k=9	1.460
Hier k=6	19.112	Kmeans k=10	1.629
Hier k=7	19.180	Kmeans k=11	1.798
Hier k=8	19.001	Kmeans k=12	2.547
Hier k=9	18.938	Kmeans k=13	2.535
Hier k=10	19.028	Kmeans k=14	3.104
Hier k=11	19.294	Kmeans k=15	3.097
Hier k=12	19.227	Kmeans k=16	2.793

Tabella 6.26: Tempi di esecuzione (in secondi) per la collezione *avila* con 1% di prototipi per classe

data banknote authentication con 1% di prototipi			
Algoritmo	Tempo	Algoritmo	Tempo
AdaPro	7.969	Hier k=13	0.048
1-level	7.742	Hier k=14	0.062
MPCK k=2	95.398	Hier k=15	0.056
MPCK k=3	94.025	Hier k=16	0.047
MPCK k=4	94.742	Kmeans k=2	0.032
MPCK k=5	95.687	Kmeans k=3	0.031
MPCK k=6	94.704	Kmeans k=4	0.063
MPCK k=7	94.596	Kmeans k=5	0.094
Hier k=2	0.078	Kmeans k=6	0.110
Hier k=3	0.063	Kmeans k=7	0.094
Hier k=4	0.062	Kmeans k=8	0.100
Hier k=5	0.064	Kmeans k=9	0.078
Hier k=6	0.047	Kmeans k=10	0.094
Hier k=7	0.047	Kmeans k=11	0.078
Hier k=8	0.031	Kmeans k=12	0.109
Hier k=9	0.032	Kmeans k=13	0.094
Hier k=10	0.047	Kmeans k=14	0.109
Hier k=11	0.046	Kmeans k=15	0.111
Hier k=12	0.032	Kmeans k=16	0.094

Tabella 6.27: Tempi di esecuzione (in secondi) per la collezione *data banknote authentication* con 1% di prototipi per classe

Capitolo 7

Conclusione

Nel corso di questa tesi di laurea magistrale abbiamo ripercorso quali siano tutti i più importanti algoritmi di *clustering* e gli strumenti che abbiamo a disposizione per valutarne le *performance*. Abbiamo introdotto il concetto di algoritmo semi-supervisionato, mostrando esempi come MPCK-Means e come il nostro algoritmo AdaPro.

AdaPro risulta essere molto comodo dal lato utente, che non deve preoccuparsi di eseguire molteplici esperimenti per selezionare i valori migliori degli iperparametri, ovvero nel nostro caso i valori ai quali effettuare il taglio del dendrogramma. AdaPro, in maniera del tutto automatica, è infatti in grado di utilizzare le informazioni racchiuse nei pochi dati etichettati per decidere se effettuare o no un'ulteriore divisione dei *cluster* ottenuti dopo la prima fase, i *super-cluster*, e a decidere eventualmente a quale altezza del dendrogramma eseguire il taglio.

A seguito delle analisi condotte su varie collezioni di dati, abbiamo appurato che, nella quasi totalità dei casi, AdaPro si pone tra gli algoritmi che ottengono le *performance* migliori, in termini di *Fowlkes-Mallows* e *Homogeneity*.

Tutto ciò, considerando che AdaPro risulta essere mediamente un algoritmo molto veloce, soprattutto se paragonato con i tempi del suo rivale semi-supervisionato, MPCK-Means.

La valutazione sperimentale descritta nel Capitolo 6, infine, risulta essere uno strumento per comprendere meglio le qualità dei vari algoritmi, in modo da facilitare la scelta, nel caso in cui il lettore si trovi di fronte a problemi simili da risolvere.

Bibliografia

[1] Thomas M. Mitchell. 1997. *Machine Learning (1st. ed.)*. McGraw-Hill, Inc., USA.

[2] Johannes Ledolter, *Data Mining and Business Analytics with R*, John Wiley and Sons, 2013

[3] Andrew R. Webb, Keith D. Copsey, *Statistical Pattern Recognition, Third Edition*, John Wiley and Sons, 2011

[4] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, *Introduction to Data Mining*, Pearson Education, 2006

[5] Mikhail Bilenko, Sugato Basu, Raymond J. Mooney, *"Integrating constraints and metric learning in semi-supervised clustering." Proceedings of the twenty-first international conference on Machine learning*, 2004

[6] Julia Bell Hirschberg, Andrew Rosenberg, *V-Measure: A Conditional Entropy-based External Cluster Evaluation*, 2007

[7] E. B. Fowlkes, C. L. Mallows *A Method for Comparing Two Hierarchical Clusterings*, Journal of the American Statistical Association 78.383 pp.553-69, 1983

[8] A. Pasini, E. Baralis, P. Garza, D. Floriello, M. Idiomi, A. Ortenzi, S. Ricci, *Adaptive Hierarchical Clustering for Petrographic Image Analysis*, In EDBT/ICDT Workshops, 2019