Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Sviluppo e confronto di algoritmi di fuzzy clustering

Relatore: Prof. Davide Elio Ciucci
Correlatore: Dott. Andrea Campagner

Relazione della prova finale di:
Andrea D'Amicis
Matricola 869008

Anno Accademico 2022-2023

Indice

Capitolo 1

Introduzione

In questa relazione verranno analizzati quali sono state le sfide che ho dovuto affrontare nel corso del mio tirocinio, le tecniche che sono state impiegate per superarle, e le tecnologie che sono state adottate in campo per arrivare a una soluzione.

1.1 Progetto e obiettivi

L'obiettivo del tirocinio è stato quello di ricercare e sviluppare algoritmi di fuzzy clustering con lo scopo di effettuare dei confronti e analizzare le prestazioni di ognuno di essi.

1.2 Struttura della tesi

La tesi è articolata nei seguenti sei punti:

- Clustering: sezione in cui vi sono le nozioni teoriche su cui si basa l'elaborato.
- Ricerca su Scopus: sezione in cui è illustrato come è stata effettuata la ricerca degli algoritmi da implementare.
- Tecnologie: sezione in cui vi è una spiegazione degli ambienti di sviluppo che sono stati utilizzati.
- Descrizione e implementazione degli algoritmi: sezione in cui viene descritta la logica applicativa di ogni algoritmo.
- Valutazione: sezione in cui vengono valutati i risultati ottenuti dagli algoritmi nel processo di clustering.
- Conclusione: valutazione personale dell'esperienza di tirocinio e possibili sviluppi futuri.

Capitolo 2

Clustering

In questo capitolo verranno definiti quali sono i concetti teorici fondamentali che riguardano sviluppo degli algoritmi di clustering.

2.1 Cos'è il Clustering

Il Clustering è un tipo di **Machine Learning**[1] (apprendimento automatico) e lavora quindi coi **dataset**, cioè una struttura dati che contiene dati omogenei tra loro, in cui ogni istanza è descritta da delle **features** o caratteristiche. Il Clustering è un tipo **apprendimento automatico non supervisionato** che si pone l'obiettivo di creare gruppi omogenei di dati, in modo che gli oggetti all'interno dello stesso cluster siano più simili tra loro, mentre gli oggetti appartenenti a cluster differenti siano più diversi tra loro. La sfida è apprendere in quanti e quali cluster è possibile dividere il dataset iniziale definendo delle tecniche di selezione e raggruppamento, ed assegnando, per ogni istanza del dataset, un **grado di appartenenza** (membership degree) che quell'istanza appartenga a quel determinato cluster. Definire le tecniche di clustering è un lavoro tutt'altro che facile, infatti anche bisogna utilizzare delle metriche con cui calcolare le distanze tra ogni istanza del dataset ed il punto di riferimento di ogni cluster, ovvero il **centroide**. Il motivo per il quale è definito come apprendimento non supervisionato, sta proprio nel fatto che esaminiamo dei dati "grezzi", che non sono stati già classificati precedentemente. Quindi il processo di clustering si baserà solo ed esclusivamente sulle features dei dati.[6]

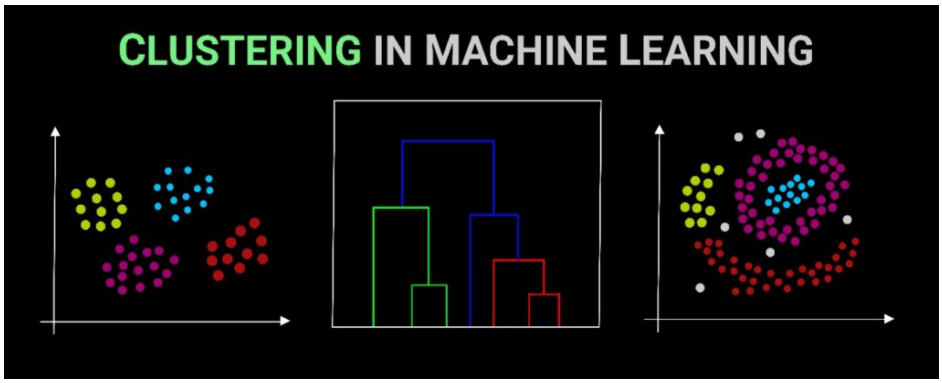

Figura 2.1: Esempi di clustering

[1] Il Machine Learning utilizza metodi statistici per migliorare la performance di un algoritmo nell'identificare pattern nei dati[2]

2.2 I diversi tipi di Clustering

A seconda delle esigenze di sviluppo e ai diversi tipi di dataset che esistono si sono sviluppate diverse tecniche di Clustering:

- **Clustering partizionale**, si tratta del tipo di Clustering più diffuso ed utilizzato in cui ogni punto dato verrà assegnato al cluster in modo esclusivo in modo tale da avere un cluster disgiunto dall'altro.

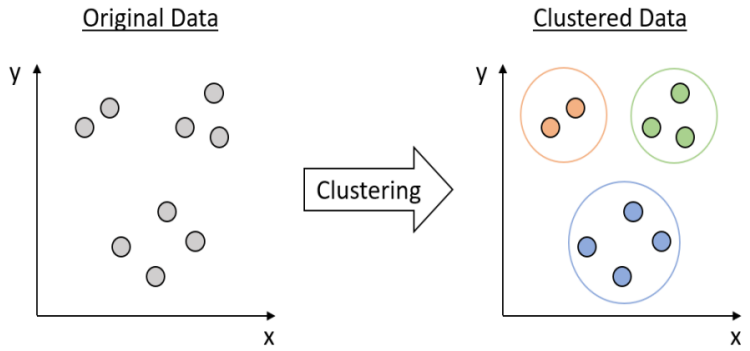

Figura 2.2: Esempio di partitional clustering

- **Clustering gerarchico**, è un tipo di Clustering simile al precedente ma dove è, invece, possibile avere una rappresentazione gerarchica ad albero degli elementi tramite un dendogramma.

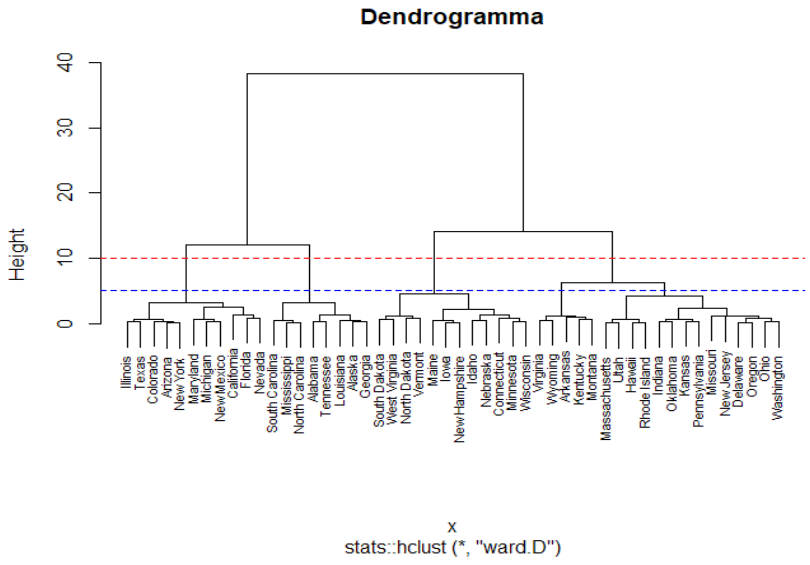

Figura 2.3: Esempio di Gerarchical Clustering

- **Density clustering**, è un tipo di Clustering che si basa sulle **densità** o la concentrazione dei punti dati. Verranno creati quindi dei cluster per densità oppure alcuni dati verrano identificati come dati outlier/rumore.[11]

Figura 2.4: Esempio di Density Clustering

- **Hard Clustering**, è un tipo di Clustering molto selettivo in cui ogni istanza di input appartiene **completamente o meno** a un cluster in modo esclusivo.
- **Soft Clustering**, è un tipo di Clustering in cui ad ogni istanza viene assegnato un **grado di appartenenza** ad uno o più cluster.[7]

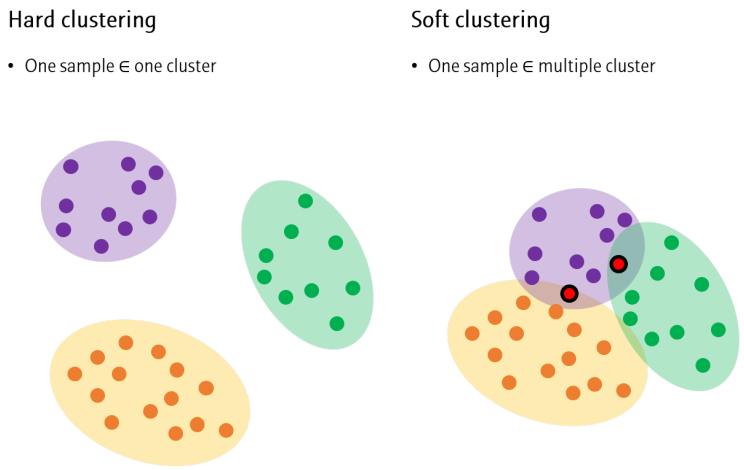

Figura 2.5: Esempio di Soft e Hard Clustering

- **Rough Clustering**, è un tipo di Soft Clustering che tiene conto dell'approssimazione e dell'incertezza nei dati, consentendo un raggruppamento flessibile dei punti in base alle informazioni disponibili. Si basa sul concetto di approssimazione appreso dalla teoria degli insiemi approssimati, sviluppata da Zdzisław Pawlak.

2.3 Cos'è la teoria e la logica Fuzzy

La teoria degli insiemi fuzzy, sviluppata da Lofti Zadeh negli anni '60, è una **teoria matematica** che estende la logica booleana, che si occupa di modellare e gestire l'**incertezza** e la vaghezza delle informazioni. A differenza della logica classica binaria (vero/falso), la logica fuzzy consente di gestire valori intermedi tra 0 e 1, che rappresentano il grado di appartenenza di un elemento a un insieme. La teoria fuzzy si basa sul concetto di insiemi fuzzy, che sono insiemi in cui gli elementi possono avere un grado di appartenenza sfocato: invece di assegnare un valore di appartenenza binario (0 o 1), si utilizzano funzioni di appartenenza che assegnano **un valore tra 0 e 1 a ciascun elemento**, indicando il grado di appartenenza a quell'insieme. Le operazioni di base nella teoria

fuzzy includono l'unione, l'intersezione e la negazione di insiemi fuzzy. Queste operazioni sfocate consentono di combinare e manipolare insiemi fuzzy per ottenere risultati sfocati. Questo tipo di logica consente, quindi, di modellare e trattare l'incertezza delle informazioni nell'ambito dell'intelligenza artificiale, offrendo un modo flessibile e potente per rappresentare e analizzare situazioni complesse in cui le informazioni sono incomplete o incerte. Utilizzando la logica fuzzy, è inoltre possibile formulare regole e inferenze sfocate che riflettono il ragionamento umano basato su informazioni vaghe o incerte.[12][3]

2.4 Cos'è il Fuzzy Clustering

Il Fuzzy Clustering è quindi un tipo di Soft Clustering in cui ogni punto dato può appartenere a più di un cluster con un **grado di appartenenza**. Ogni istanza del dataset, quindi, avrà un vettore di appartenenza che rappresenta il suo grado di appartenenza a **ciascun cluster**. Questi valori di appartenenza possono variare da 0 a 1, indicando quanto un punto sia associato a un determinato cluster. Un valore di appartenenza vicino a 1 indica un'alta probabilità di appartenenza al cluster, mentre un valore vicino a 0 indica una bassa probabilità di appartenenza. La parola **fuzzy**[2] del Fuzzy Clustering va proprio a instaurare un grado di **incertezza** intrinseca nell'associare un oggetto a un gruppo di cose. Infatti, con la parola fuzzy, vogliamo proprio sottolineare il fatto che non avremo un "confine" ben preciso tra un cluster e l'altro, bensì una distinzione sfocata tra i vari cluster.[8]

2.5 Funzionamento di un algoritmo di Clustering

Il funzionamento di un algoritmo di Clustering prevede solitamente l'esecuzione dei seguenti punti:

1. **Preparazione dei dati**: In questa fase iniziale, i dati vengono analizzati e preparati per il Clustering. Questo può includere la normalizzazione dei dati, la gestione dei valori mancanti o la riduzione delle dimensione del dataset.

2. **Selezione del numero di cluster**: In alcuni algoritmi di clustering partizionale è necessario specificare in anticipo il **numero desiderato di cluster** (**iperparametro** dell'algoritmo). In altri algoritmi, come il Clustering gerarchico, il numero di cluster può essere determinato durante l'esecuzione dell'algoritmo.

3. **Inizializzazione**: Gli algoritmi di Clustering richiedono solitamente una fase di inizializzazione in cui vengono assegnati inizialmente i centroidi dei cluster. In molti casi questi vengono scelti randomicamente.

4. **Assegnazione dei punti**: In questa fase, ogni punto dati viene assegnato a un cluster specifico sulla base di una **misura di similarità o di distanza** tra i punti. L'obiettivo è assegnare i punti al cluster più simile o più vicino in base a un criterio specifico.

5. **Aggiornamento dei centroidi o dei punti di riferimento dei cluster**: Dopo l'assegnazione iniziale dei punti, viene **ricalcolato il centroide** per ciascun cluster

[2] fuzzy = sfocato

sulla base dei nuovi punti che sono stati aggiunti al cluster. Questo può essere fatto come la media dei punti appartenenti a quel cluster o tramite altri metodi specifici dell'algoritmo di Clustering utilizzato.

6. **Iterazione**: I passaggi di assegnazione e aggiornamento vengono ripetuti iterativamente fino a quando non si verifica una condizione di terminazione predefinita. Questa condizione può essere raggiunta quando i centroidi dei cluster arrivano a un **punto di convergenza** o quando viene raggiunto un **numero massimo di iterazioni**. Come punto di convergenza solitamente viene stabilito un valore minimo di spostamento del centroide.

7. **Valutazione dei risultati**: Una volta completato l'algoritmo di Clustering, è importante valutare la qualità dei risultati ottenuti. Infatti è possibile effettuare una valutazione interna (nel caso non si abbia un dataset che non sia stato in precedenza classificato) valutando la statistica dei dati oppure è possibile confrontare i dati clusterizzati con quelli classificati in precedenza.

È importante sottolineare che gli algoritmi di Clustering possono variare notevolmente nel modo in cui misurano la similarità o la distanza tra i punti, come aggiornano i centroidi dei cluster e come convergono verso una soluzione. Pertanto, la scelta dell'algoritmo di Clustering più adatto dipende dal contesto del problema e dalle caratteristiche dei dati. Di seguito vi è una immagine illustrativa che mostra concretamente come si "muovono" i centroidi durante ogni iterazione di un algoritmo di Clustering.

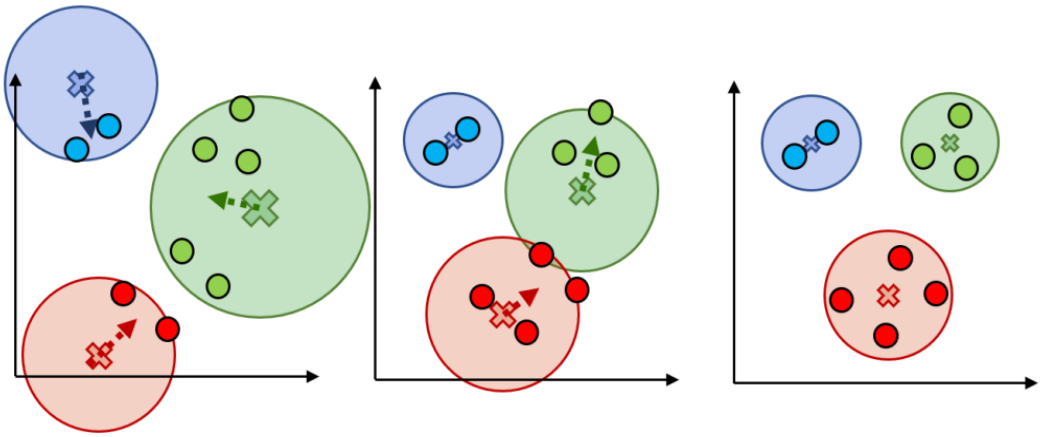

Figura 2.6: Esempio di aggiornamento dei centroidi

2.6 Fuzzy C-Means

Fuzzy C-Means è quindi un algoritmo di Soft Clustering basato sulla teoria Fuzzy che funziona, a livello logico, come quanto detto sopra. Questo algoritmo è stato la base di partenza della mia ricerca. Di seguito è riportato lo pseudocodice con le formule di aggiornamento dei centroidi ed il calcolo dei gradi di membership.

Algorithm 1 Fuzzy C-Means

Inputs: X, c, m, τ

for $t = 1$ **to** τ **do**

$$u_{ij} = \left[\sum_{k=1}^{c} \left(\frac{||x_i - v_j||^2}{||x_i - v_k||^2} \right)^{\frac{1}{m-1}} \right]^{-1} \forall i \in [1, n], \forall j \in [1, c]$$

$$v_j = \frac{\sum_{i=1}^{n} u_{ij}^m x_i}{\sum_{i=1}^{n} u_{ij}^m} \ \forall j \in [1, c]$$

end for

Outputs: D

Legenda:

- X: dataset
- c: numero di cluster
- m: grado di fuzziness (tipicamente è 2)
- τ: numero di iterazioni
- v_j: centroide j-esimo
- u_{ij}: grado di membership tra i-esimo elemento del dataset e j-esimo centroide
- D: matrice dei gradi di membership

Capitolo 3

Ricerca su Scopus

La seguente sezione mostra la modalità con cui sono stati individuati e classificati ai fini dei nostri interessi gli algoritmi di Fuzzy clustering.

3.1 Obiettivo

L'obiettivo di questa fase era quello di trovare delle dipendenze o delle estensioni di varianti di Fuzzy Clustering partendo dall' algoritmo Fuzzy C-Means.

3.2 Procedimento

Innanzitutto è stata utilizzata la piattaforma Scopus per estrarre tutte quelle pubblicazioni inerenti al Fuzzy Clustering attraverso un'interrogazione e quindi tramite l'utilizzo della seguente query:

```
TITLE(fuzzy AND clustering AND (survey OR review))
```

Una volta ottenuta la lista di articoli è stata fatta una analisi della descrizione (abstract) degli articoli ed è quindi stata fatta una scrematura. Le pubblicazioni ritenute come utili sono quelle che nominavano degli algoritmi di Fuzzy Clustering, mentre tutte quelle pubblicazioni che parlavano solo della teoria del Clustering o di domini di applicazione troppo specifici sono state scartate. Successivamente sono, quindi, rimasti tutti quegli articoli che contenevano degli algoritmi che proponevano una soluzione su un dominio di dati generico e non troppo specifico. Una volta ottenuto il pool di algoritmi, la fase successiva è stata quella di classificarli in modo da vedere se fossero correlati tra loro andando a capire se vi fosse un algoritmo "padre" da cui si diramassero altre versioni con metriche o logiche differenti.

3.3 Risultato

La query ha riportato come risultato 42 articoli di natura prevalentemente accademica. Una volta eseguita la scrematura di essi, sono rimasti solamente 3 articoli che parlavano effettivamente di varianti di algoritmi di Fuzzy Clustering:

- *Density-based IFCM along with its interval valued and probabilistic extensions, and a review of intuitionistic fuzzy clustering methods*[20]
  Questo articolo espone il problema che la maggior parte degli algoritmi C-Means come FCM (Fuzzy C-Means), IFCM (Intuitionistic Fuzzy C-Means) e il PIFCM (Probabilistic Intuitionistic Fuzzy C-Means) inizializzano in modo casuale i centroidi del cluster con la conseguente problematica le prestazioni dell'algoritmo dipendono molto dai centroidi iniziali. Propone come soluzione un algoritmo basato sulla densità: DPIFCM (Density Based Probabilistic Intuitionistic Fuzzy C-Means) che però non era nei nostri interessi in quanto non utilizzava la logica fuzzy.

- *Fuzzy C-Means clustering algorithm for data with unequal cluster sizes and contaminated with noise and outliers: Review and development*[1]
  Questo articolo espone, invece, dei problemi che si porta dietro FCM (Fuzzy C-Means) riguardo ad outliers, rumore, valori anomali e alla disuguaglianza in termini di dimensioni tra i vari cluster, che porta a muovere i centroidi dei cluster "piccoli" verso quelli "grandi". Propone quindi una versione rivisitata in cui utilizza funzioni esponenziali adattive per eliminare gli impatti del rumore e dei valori anomali sui centroidi dei cluster, modifica il vincolo dell'algoritmo FCM per impedire quindi ai cluster "grandi" o "più pesanti" di attrarre i centroidi di "piccoli" cluster verso di loro. Analizza, quindi, come soluzioni i seguenti algoritmi:

  - Possibilistic Fuzzy C-Means (PFCM)
  - Possibilistic C-Means (PCM)
  - Robust Fuzzy C-Means (FCM-σ)
  - Noise Clustering (NC)
  - Kernel Fuzzy C-Means (KFCM)
  - Intuitionistic Fuzzy C-Means (IFCM)
  - Robust Kernel Fuzzy C-Mean (KFCM-σ)
  - Kernel Intuitionistic Fuzzy C-Means (KIFCM)
  - Robust Kernel Intuitionistic Fuzzy C-Means (KIFCM-σ)
  - Credibilistic Fuzzy C-Means (CFCM)
  - Size-insensitive integrity-based Fuzzy C-Means (siibFCM)
  - Size-insensitive Fuzzy C-Means (csiFCM)
  - Subtractive Clustering (SC)
  - Density Based Spatial Clustering of Applications with Noise (DBSCAN)
  - Gaussian Mixture Models (GMM)
  - Spectral clustering and Outlier Removal Clustering (ORC)

- *Performance Analysis of Various Fuzzy Clustering Algorithms: A Review*[14]
  Questo articolo, invece, elenca e confronta i seguenti algoritmi:

- Fuzzy C-Means (FCM)
- Possibilistic C-Means (PCM)
- Possibilistic Fuzzy C-Means (PFCM)
- Robust Fuzzy C-Means (FCM-σ)
- Type-2 Fuzzy C-Mean (T2FCM)
- Kernel Type-2 Fuzzy C-Means (KT2FCM)
- Intuitionistic Fuzzy C-Means (IFCM)
- Kernel Intuitionistic Fuzzy C-Means (KIFCM)
- Robust Intuitionistic Fuzzy C-Means (IFCM-σ)
- Robust Kernel Intuitionistic Fuzzy C-Mean (KIFCM-σ)
- Normalized Cuts (NC)
- Credibilistic Fuzzy C-Means (CFCM)

Per ragioni di interesse personale e dei docenti relatori della tesi è stato quindi deciso di sviluppare, con lo scopo di effettuare dei confronti, i seguenti quattro algoritmi:

- **Robust Fuzzy C-Means (FCM-σ)**
- **Kernel Fuzzy C-Means (KFCM)**
- **Credibilistic Fuzzy C-Means (CFCM)**
- **Size-insensitive integrity-based Fuzzy C-Means (siibFCM)**

Di seguito vi è il diagramma delle dipendenze degli algoritmi che sono stati esaminati:

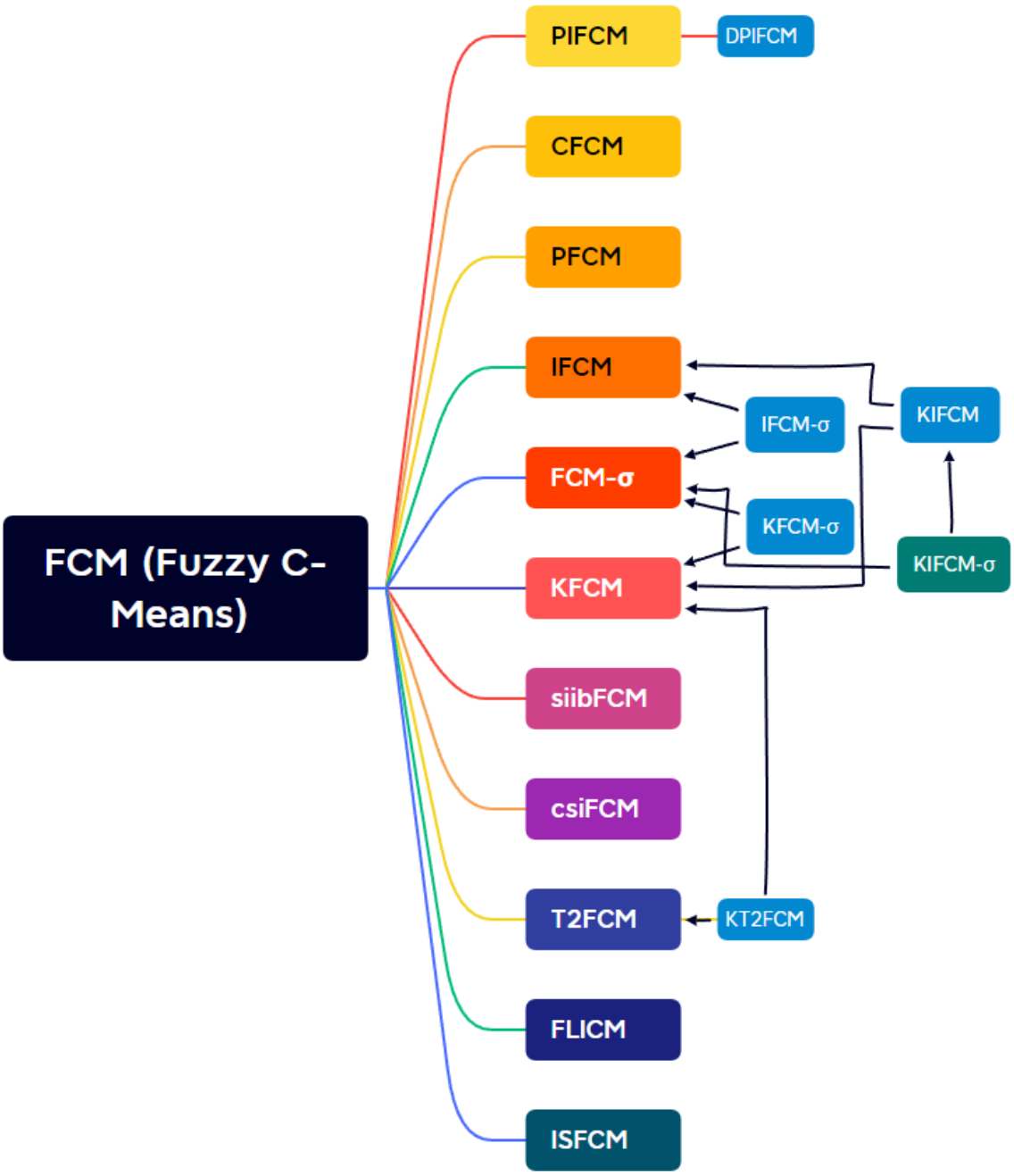

Figura 3.1: Dipendenze

Capitolo 4

Tecnologie

La seguente sezione mostra le tecnologie adottate per lo sviluppo degli algoritmi.

4.1 Python

Figura 4.1: Il logo di Python

Python è un linguaggio di programmazione multi-paradigma ad alto livello, interpretato e orientato agli oggetti. È noto per la sua semplicità, leggibilità e per la sua flessibilità, che favorisce la scrittura di codice chiaro e conciso. Python è ampiamente utilizzato in vari settori, come lo sviluppo di software, l'analisi dei dati, l'intelligenza artificiale, il web development e molto altro. Ecco alcune caratteristiche chiave del linguaggio Python:

- **Sintassi leggibile**: Python si distingue per la sua sintassi pulita e intuitiva. Utilizza l'indentazione significativa, ovvero l'uso di spazi o tabulazioni per definire i blocchi di codice, rendendo il codice più leggibile.
- **Orientamento agli oggetti**: Python supporta la programmazione orientata agli oggetti (OOP) consentendo la definizione di classi, oggetti e metodi. Questo approccio favorisce la modularità, il riutilizzo del codice e la gestione efficiente delle complessità.
- **Ampia libreria standard**: Python dispone di una vasta libreria standard che offre una vasta gamma di moduli e funzionalità predefinite. Ciò semplifica lo sviluppo di applicazioni, poiché molti strumenti e utilità comuni sono già inclusi nel linguaggio.
- **Community attiva**: Python ha una community di sviluppatori molto attiva e numerosi pacchetti di terze parti. Questi pacchetti estendono le funzionalità di

base di Python, consentendo di affrontare una varietà di compiti specializzati come l'analisi dei dati, l'apprendimento automatico, il web scraping e molto altro.

- **Portabilità**: Python è un linguaggio multipiattaforma, il che significa che il codice scritto in Python può essere eseguito su diverse piattaforme, come Windows, macOS e Linux, senza dover apportare modifiche significative. Infatti essendo precompilato[1] in bytecode offre la possibilità di distribuire programmi Python direttamente in bytecode, saltando totalmente la fase di interpretazione[2] da parte dell'utilizzatore finale.

- **Facilità di apprendimento**: Python è considerato uno dei linguaggi più semplici da imparare per i principianti. La sua sintassi leggibile e la natura intuitiva lo rendono accessibile anche a coloro che non hanno esperienza di programmazione. Basti pensare che le variabili non sono tipizzate.

L'implementazione, infatti, è stata realizzata in Python proprio per la sua facilità di comprensione, scrittura e flessibilità.[17]

4.2 Jupyter

Figura 4.2: Il logo di Jupyter Notebook

Jupyter Notebook è un'applicazione web open-source che consente di creare e condividere documenti interattivi contenenti codice Python, testo formattato, immagini, grafici e altro ancora. È ampiamente utilizzato nel campo della scienza dei dati, dell'analisi e dell'apprendimento automatico. Ecco alcune caratteristiche chiave di Jupyter Notebook:

- **Interfaccia web interattiva**: Jupyter Notebook offre un'interfaccia basata sul web in cui è possibile creare e modificare i propri notebook. Puoi accedere a Jupyter Notebook tramite il tuo browser preferito senza dover installare software aggiuntivo.

- **Cell-based**: I notebook Jupyter sono organizzati in celle (cells). Ogni cella può contenere codice Python, testo formattato utilizzando la sintassi Markdown o anche contenuti multimediali come immagini e video. Le celle possono essere eseguite individualmente o in sequenza.

[1] il processo di compilazione traduce una serie di istruzioni scritte in un determinato linguaggio di programmazione (codice sorgente) in istruzioni di un altro linguaggio (codice oggetto)[10]

[2] il processo di interpretazione esegue le istruzioni nel linguaggio usato traducendole di volta in volta in istruzioni in linguaggio macchina del processore[15]

- **Esecuzione del codice interattiva**: Con Jupyter Notebook, è possibile eseguire il codice Python direttamente nelle celle. Ciò significa che puoi scrivere e testare porzioni di codice in modo interattivo, visualizzando i risultati immediatamente sotto la cella corrispondente. Questa caratteristica è particolarmente utile per l'esplorazione e l'analisi dei dati.

- **Visualizzazioni interattive**: Jupyter Notebook supporta la visualizzazione di grafici e altre visualizzazioni interattive, consentendo di esplorare i dati in modo dinamico. Puoi utilizzare librerie di visualizzazione come Matplotlib, Seaborn e Plotly per creare grafici interattivi direttamente nel notebook.

- **Documentazione ricca**: Oltre al codice, puoi utilizzare il Markdown per formattare il testo nel notebook, inserire formule matematiche, creare elenchi puntati, tabelle e molto altro ancora. Questa funzionalità ti consente di creare documenti interattivi e ben strutturati che combinano testo esplicativo, codice e risultati.

- **Condivisione e collaborazione**: I notebook Jupyter possono essere facilmente condivisi con altre persone. Puoi esportare i notebook in diversi formati, tra cui HTML, PDF e Markdown, o condividerli direttamente su piattaforme come GitHub o Jupyter Notebook Viewer. Inoltre, Jupyter supporta la collaborazione multiutente, consentendo a più persone di lavorare contemporaneamente su un notebook.

- **Estensibilità**: Jupyter Notebook è altamente estensibile grazie all'ampia gamma di estensioni e plugin disponibili. Puoi personalizzare l'esperienza di Jupyter utilizzando temi diversi, aggiungendo funzionalità aggiuntive o integrando altre librerie e strumenti.

Jupyter Notebook, invece, è stato utilizzato nel progetto proprio nella fase di testing e valutazione proprio perchè risultava molto più facile e interattivo da usare per l'analisi dei dati.

4.3 Visual Studio Code

Figura 4.3: Il logo di Visual Studio Code

Visual Studio Code (VS Code) è una **applicazione open source** sviluppata da Microsoft per Windows, macOS e Linux e che fa da **editor di codice sorgente**. VS Code è stato progettato con l'idea di essere il più leggero, estensibile e altamente personalizzabile per garantire un'esperienza di sviluppo moderna e per una vasta gamma di linguaggi di programmazione. VS Code fornisce, infatti, funzionalità avanzate per lo sviluppo di software, compresi strumenti per la scrittura, l'editing e il debug del codice. Supporta molti linguaggi di programmazione popolari, come Python, JavaScript, C++, Java, PHP, Go e molti altri, e offre funzionalità di highlighting[3] della sintassi, completamento automatico, formattazione del codice e navigazione tra i file. Il punto di forza di VS Code è la sua architettura basata su **estensioni** in cui gli sviluppatori possono aggiungere funzionalità personalizzate all'editor per integrare nuovi strumenti, temi, linguaggi, framework e molto altro ancora. VS Code, inoltre, supporta anche integrazioni con sistemi di controllo del codice sorgente come Git offrendo le funzionalità per il lavoro collaborativo, il controllo delle versioni e la gestione delle repository. Grazie alla sua popolarità e alla sua vasta comunità di sviluppatori, VS Code è diventato uno degli editor di codice più utilizzati e apprezzati nell'ambito dello sviluppo software. Proprio per la sua facilità d'utilizzo nell'aggiungere estensioni, infatti, mi è bastato semplicemente aggiungere a VS Code le estensioni di Python, per la parte di sviluppo degli algoritmi, e Jupyter Notebook, per la parte di analisi e valutazione.[9]

[3]consente di visualizzare il codice con differenti colori e font in base a particolari regole sintattiche

Capitolo 5

Descrizione ed implementazione degli algoritmi

Questa sezione spiega alcuni concetti fondamentali utili alla fase implementativa e le varianti degli algoritmi implementati.

5.1 Dettagli implementativi

Qui di seguito vengono illustrati i dettagli implementativi utili per capire al meglio lo pseudocodice e le varianti di FCM. L'idea è quella di creare una classe python per ogni variante di algoritmo che si andrà ad implementare e di seguire le basi strutturali di progettazione con cui è stata implementata una delle migliori librerie del Machine Learning di Python, cioè **Scikit-learn (sklearn)**. Infatti, seguendo la base degli algoritmi di sklearn, si andranno a implementare, per ogni algoritmo, quelli che sono i seguenti metodi:

- **init(self, n_clusters, epsilon, iters, random_state=None, n_init=10, metric='euclidean', m=2, method='fuzzy')**: metodo che inizializza la classe e che fa da "costruttore" andando a specificare il numero di cluster, il numero di iterazioni, la metrica di distanza ed il grado di fuzziness.
- **fit(X)**: metodo che adatterà il modello alle istanze di addestramento di input, ovvero quelle del **training set**.
- **predict(X)**: metodo che eseguirà previsioni sulle istanze di test ovvero quelle del **validation test**, in base ai parametri appresi durante la fase di addestramento (metodo fit).
- **fit_predict(X)**: richiama prima la fit e poi la predict.

Le classi che si andranno ad implementare, inoltre, utilizzeranno nello specifico le seguenti funzioni provenienti dalle librerie di Numpy ed sklearn:

- **pairwise_distances(X, self.centroids, metric=self.metric)**: che consente di calcolare facilmente la matrice delle distanze tra i centroidi e i punti dati del dataset prendendo come parametri di input il dataset e la matrice dei centroidi.
- **resample(X)**: consente di ricampionare il dataset in base al numero di cluster che abbiamo impostato.

Per implementare gli algoritmi sono state utilizzate 3 matrici:

- **X**: matrice che rappresenta il nostro dataset e sarebbe una matrice Numpy.
  X è una matrice di dimensioni nxd in cui **n** sono le **istanze** del dataset e **d** le **features**.

- **V**: matrice che contiene i valori delle **distanze euclidee** di una feature in funzione al centroide a cui si fa riferimento.
  V è matrice di dimensioni cxd in cui **c** è il **numero di cluster** e **d** le **features**. Nel codice corrisponde alla matrice **self.centroids**.

- **D**: matrice che contiene i **gradi di membership** tra il generico punto del dataset x_i e il centroide v_j. D è una matrice di dimensioni nxc in cui **n** sono le **istanze** e **c** è il **numero di cluster**. Nel codice corrisponde alla matrice **self.cluster_assignments**

In particolare, per ogni algoritmo, viene utilizzata la matrice **dists** che sarebbe una matrice nxc e che viene restituita dalla funzione **pairwise_distances(X, self.centroids, metric=self.metric)**. Questa funzione, infatti, ci permette di calcolare facilmente tutte le distanze euclidee nella matrice dists, che verrà poi utilizzata come metrica per calcolare i membership degree nella matrice self.cluster_assignments e, successivamente, per ricalcolare i centroidi in self.centroids.

5.2 Robust Fuzzy C-Means - FCM-σ

FCM-σ (Robust Fuzzy C-Means) è un algoritmo di clustering che è un'estensione dell'algoritmo Fuzzy C-Means. L'algoritmo FCM-σ è una versione robusta dell'algoritmo Fuzzy C-Means, che è **meno sensibile ai valori anomali** o al rumore nei dati. Raggiunge questo obiettivo utilizzando una **funzione robusta**, che riduce l'influenza dei valori anomali sui risultati del clustering. Utilizza un criterio di soglia basato sulla **deviazione standard** dei dati per **identificare i dati anomali** e li rimuove dal processo di clustering. Inoltre, tramite una funzione di peso, **assegna maggior peso ai dati che sono meno sensibili agli effetti degli outlier**. Questo aiuta a **diminuire l'effetto degli outlier sui centroidi dei cluster** e quindi a migliorare la qualità del clustering. Questa variante dell'algoritmo, seppur diversa, è molto simile alla versione classica con la peculiarità che utilizza la deviazione standard nel calcolo delle distanze euclidee utili a calcolare il grado di membership.

Algorithm 2 Robust Fuzzy C-Means

Inputs: X - dataset, c - n° cluster, m - grado di fuzziness, τ - n° iterazioni

for $t = 1$ **to** τ **do**

$$\sigma_j = \left(\frac{\sum_{i=1}^{n} u_{ij}^m ||x_i - v_j||^2}{\sum_{i=1}^{n} u_{ij}^m} \right)^{\frac{1}{2}} \quad \forall j \in [1, c]$$

$$u_{ij} = \left[\sum_{k=1}^{c} \left(\frac{||x_i - v_j||^2 / \sigma_j}{||x_i - v_k||^2 / \sigma_k} \right)^{\frac{1}{m-1}} \right]^{-1} \quad \forall i \in [1, n], \forall j \in [1, c]$$

$$v_j = \frac{\sum_{i=1}^{n} u_{ij}^m x_i}{\sum_{i=1}^{n} u_{ij}^m} \quad \forall j \in [1, c]$$

end for

Output: D

5.3 Kernelized Fuzzy C-Means - KFCM

KFCM (Kernelized Fuzzy C-Means) è una variante dell'algoritmo Fuzzy C-Means (FCM) che utilizza una **funzione kernel per mappare i dati in uno spazio di dimensione superiore** dove la separabilità dei cluster può essere migliorata, rendendo applicabile l'algoritmo FCM in modo più efficiente e accurato. La funzione kernel è una **funzione matematica che calcola la similarità tra due punti nel dataset**, in base alla loro distanza nello spazio originale. La funzione di kernel utilizzata in KFCM viene, generalmente, scelta in base alle caratteristiche dei dati analizzati. Per applicare questa funzione occorre, però, per prima cosa, calcolare la **media** dei punti del dataset. In questo caso la libreria numpy ci viene molto utile poichè ci permette di calcolare la media con l'utilizzo della funzione np.mean(X). Una volta fatto ciò, bisogna calcolare la **varianza** delle distanze di tutti i punti del dataset dalla media dei punti precedentemente calcolata. Successivamente sarà possibile calcolare la **deviazione standard** che ci servirà come input per la funzione di kernel. A questo punto occorre applicare la seguente **funzione di kernel** alla matrice delle distanze euclidee ed infine potremo calcolare i gradi di membership e ricalcolare i centroidi.
Di seguito la formula della funzione di kernel:

$$e^{-\frac{||x_i - v_j||^2}{\sigma^2}} \qquad \forall i \in [1, n], \forall j \in [1, c]$$

Infatti abbiamo come esponente del numero di Nepero il rapporto negativo tra la distanza dell'i-esima istanza del dataset ed il j-esimo centroide, divisa per la varianza del dataset.

Algorithm 3 Kernelized Fuzzy C-Means

Inputs: X - dataset, c - n° cluster, m - grado di fuzziness, τ - n° iterazioni

$\bar{v} = \frac{1}{n} \sum_{i=1}^{n} x_i$

$\bar{d} = \frac{1}{n} \sum_{i=1}^{n} \sqrt{||x_i - \bar{v}||^2}$

$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} \left(\sqrt{||x_i - \bar{v}||^2} - \bar{d} \right)^2$

$v_j = \frac{\sum_{i=1}^{n} u_{ij}^m x_i}{\sum_{i=1}^{n} u_{ij}^m} \quad \forall j \in [1, c]$

for $t = 1$ **to** τ **do**

$u_{ij} = \left[\sum_{k=1}^{c} \left(\frac{1-K(x_i, v_j)}{1-K(x_i, v_k)} \right)^{\frac{1}{m-1}} \right]^{-1} \quad \forall i \in [1, n], \forall j \in [1, c]$

$v_j = \frac{\sum_{i=1}^{n} u_{ij}^m K(x_i, v_j) x_i}{\sum_{i=1}^{n} u_{ij}^m K(x_i, v_j)} \quad \forall j \in [1, c]$

end for

Output: D

5.4 Credibilistic Fuzzy C-Means - CFCM

CFCM (Credibilistic Fuzzy C-Means) è un algoritmo di clustering che estende l'algoritmo Fuzzy C-Means (FCM) incorporando anche una nozione di credibilità. A differenza dell'FCM, in cui ogni punto appartiene a ciascun cluster con un grado di appartenenza, in CFCM ogni punto ha una credibilità rispetto a ciascun cluster. Ciò permette di estendere l'algoritmo vedendo quali sono i punti ad **alta o bassa densità** osservando il **q-vicinato** di ogni istanza del dataset. Il paramentro q è ottenuto come prodotto tra il valore di γ (compreso in [0,1] e ottenuto come input) per il rapporto tra il numero di istanze ed il numero di cluster. Il q-vicinato è definito, quindi, come quell'area immaginaria che ha come centro l'i-esima istanza del dataset e che ci serve per calcolare la distanza tra il punto di riferimento che stiamo esaminando e i suoi q punti più vicini. Infatti, come primo passo, possiamo notare che l'algoritmo, a differenza dei precedenti, prende in input un altro iperparametro γ che deve essere compreso in [0,1] e che serve proprio per calcolare q. Per fare ciò su python è stato necessario importare da sk-learn la libreria **Nearest Neighbors** che permette di calcolare facilmente il q-vicinato per ogni punto dato. Infatti, applicando

```
neigh = NearestNeighbors(n_neighbors=q)
neigh.fit(X)
dists_n = neigh.kneighbors(X)
```

è possibile salvarsi nella matrice dist_n **le distanze del punto i-esimo dai suoi q+1 vicini** (dimensioni $nx(q+1)$). Per come funziona la libreria è importante tener presente che occorre lavorare sui q+1 vicini poichè il primo vicino di un elemento è l'elemento stesso. Una volta ottenuta la matrice dei vicini occorre calcolare la θ, ovvero la **media delle distanze di un punto dai suoi vicini**. L'array contenente i valori di θ per ogni punto dato del dataset ci consentirà, infine, di calcolare come segue il **valore credibilità** ρ, associato anch'esso a ciascuna istanza del dataset. Infine basterà calcolare quindi i gradi di membership fuzzy sulla base dei valori calcolati e credibilistici di ρ.

Algorithm 4 Credibilistic Fuzzy C-Means

Inputs: X - dataset, c - n° cluster, m - grado di fuzziness, τ - n° iterazioni γ - valore in [0,1]

$q = \gamma(\frac{n}{c})$

$\theta_i = \frac{\sum_{z=1}^{q} ||y_i^z - x_i||^2}{q} \quad \forall i \in [1,n], \forall z \in [1,q]$

$\rho_i = 1 - \frac{\theta_i - min(\theta_p)}{max(\theta_p) - min(\theta_p)} \quad \forall i, \forall p \in [1,n]$

for $t = 1$ **to** τ **do**

$$u_{ij} = \rho_i \left[\sum_{k=1}^{c} \left(\frac{||x_i - v_j||^2}{||x_i - v_k||^2} \right)^{\frac{1}{m-1}} \right]^{-1} \quad \forall i \in [1,n], \forall j \in [1,c]$$

$v_j = \frac{\sum_{i=1}^{n} u_{ij}^m x_i}{\sum_{i=1}^{n} u_{ij}^m} \quad \forall j \in [1,c]$

end for

Output: D

5.5 Size-insensitive Fuzzy C-Means - csiFCM

CsiFCM (Size-insensitive Fuzzy C-Means) è un algoritmo di clustering che estende FCM (Fuzzy C-Means) e che utilizza una strategia di pesatura. Per gestire la problematica delle differenze di dimensione tra i cluster csiFCM introduce un **parametro di pesatura** che tiene conto della **dimensione dei cluster**, consentendo di **creare cluster con dimensioni più equilibrate**. In questo modo, csiFCM è in grado di gestire dataset con oggetti di diverse dimensioni senza che gli oggetti più grandi abbiano un impatto maggiore sui centroidi dei cluster. Per fare ciò occorre calcolare quanti elementi contiene ogni cluster. Successivamente va calcolata la percentuale di grandezza di ogni cluster e poi va normalizzata. A questo punto è possibile calcolare come segue l'array contenente i valori di ρ che serviranno per calcolare i gradi di membership.

Algorithm 5 Size-insensitive Fuzzy C-Means

Inputs: X - dataset, c - n° cluster, m - grado di fuzziness, τ - n° iterazioni

D = rand(n,c)

for $t = 1$ **to** τ **do**

$$A_j = \left\{ x_i \in X | u_{ij} > u_{ki};\ \forall k \in [1,c];\ k \neq i;\ \forall j \in [1,c] \right\}$$

$$S_j = \frac{|A_j|}{|X|} \quad \forall j \in [1,c]$$

$$\rho_i = \frac{1 - S_j}{min(S_p)} \quad \forall i \in [1,n]; \forall j, \forall p \in [1,c]$$

$$u_{ij} = \rho_i \left[\sum_{k=1}^{c} \left(\frac{||x_i - v_j||^2}{||x_i - v_k||^2} \right)^{\frac{1}{m-1}} \right]^{-1} \quad \forall i \in [1,n], \forall j \in [1,c]$$

$$v_j = \frac{\sum_{i=1}^{n} u_{ij}^m x_i}{\sum_{i=1}^{n} u_{ij}^m} \quad \forall j \in [1,c]$$

end for

Output: D

Capitolo 6

Valutazione

Questa sezione vuole presentare e discutere i risultati ottenuti dalla fase di valutazione degli algoritmi.

6.1 Descrizione

Una volta conclusa la fase di sviluppo degli algoritmi in Python, è stato necessario valutare le performance di quest'ultimi. Per fare ciò è stato utilizzato Jupyter Notebook in quanto consentiva in modo semplice ed intuitivo di andare a testare i vari algoritmi per ogni dataset. Lo scopo di questa fase di valutazione era, quindi, valutare le performance ed effettuare confronti tra i vari algoritmi che sono stati implementati. In particolare si è deciso di effettuare 10 iterazioni di ognuno dei quattro algoritmi sviluppati e per ognuno degli 11 datasets che sono stati scelti. La scelta dei dataset è stata effettuata con lo scopo di andare a toccare diversi tipi di combinazioni possibili tra numero di istanze, features e classi per testare al meglio gli algoritmi in diversi campi di applicazione. La diversa (e in alcuni casi eccessiva) grandezza dei dataset in funzione ai tempi di esecuzione troppo lunghi ha portato alla scelta di dover ricampionare in modo bilanciato alcuni di essi a un massimo di 500 istanze (tramite la funzione resample di sklearn).
Tutti i dataset che sono stati usati come test per i quattro algoritmi sviluppati provengono dalla **UCI Machine Learning Repository**[19] e di seguito vi sono riportati i nomi di ciascuno di essi con le relative dimensioni di **n°istanze, n°features e n°classi**:

- **Chemical Composition of Ceramic**, 88 x 17 x 2
- **Crowdsourced Mapping**, 300 x 28 x 6
- **Ecoli**, 336 x 7 x 8
- **Glass Identification**, 214 x 9 x 7
- **Iris**, 150 x 4 x 3
- **Optical Recognition**, 500 x 64 x 10
- **Breast Cancer Wisconsin**, 569 x 32 x 2
- **Wine**, 178 x 13 x 3
- **Wine-quality-red**, 500 x 11 x 9

- **Wine-quality-white**, 500 x 11 x 9
- **Yeast**, 1483 x 8 x 10

6.2 Indice di Rand

La valutazione che è stata effettuata si basa sull'indice di Rand. Si tratta di una **valutazione esterna** che effettua una **misura di similarità o dissimilarità** tra due insiemi di dati per analizzare e valutare la qualità di un algoritmo di clustering confrontando i risultati ottenuti con un insieme di etichette di riferimento o verità, cioè il **Ground Truth**. Queste etichette sono, solitamente, inserite all'interno dei dataset e corrispondono alle **classi di appartenenza** degli elementi. L'indice di Rand calcola la frazione di coppie di punti che sono classificate in modo coerente tra i due insiemi, rispetto a tutte le possibili coppie. Questo indice restituisce un valore compreso tra 0 e 1, dove 0 indica una completa dissimilarità tra i due insiemi e 1 indica una completa similarità. Per calcolare l'indice di Rand, si considerano le seguenti quattro categorie:

- **True Positive (TP)**: Coppie di punti che sono classificate in modo coerente come "simili" sia nell'insieme di riferimento che nel risultato dell'algoritmo di clustering.
- **True Negative (TN)**: Coppie di punti che sono classificate in modo coerente come "diverse" sia nell'insieme di riferimento che nel risultato dell'algoritmo di clustering.
- **False Positive (FP)**: Coppie di punti che sono classificate come "simili" nell'insieme di riferimento ma come "diverse" nel risultato dell'algoritmo di clustering.
- **False Negative (FN)**: Coppie di punti che sono classificate come "diverse" nell'insieme di riferimento ma come "simili" nel risultato dell'algoritmo di clustering.

Di seguito la formula per calcolare l'indice di Rand:

$$\text{Indice di Rand} = \frac{TP+TN}{TP+TN+FP+FN}$$

In generale valori più alti dell'indice di Rand indicano una maggiore similarità tra i due insiemi di dati e quindi una migliore qualità del clustering. Tuttavia, nella nostra implementazione, il valore di errore è stato calcolato come 1 - indice di Rand e quindi più è vicino a 0, più è buona la performance dell'algoritmo di clustering. [18]

6.3 Risultati

Durante la fase di testing sono stati raccolti 2 tipologie di dati:

- **Tempo di esecuzione**: tempo medio di esecuzione di ogni algoritmo calcolato sulla performance ottenuta in ognuna delle 10 iterazioni e la relativa deviazione standard calcolata analogamente.
- **Errore**: questo valore viene calcolato convertendo il Fuzzy Clustering in una **distribuzione di probabilità sul Rough Clustering**. Infatti, data una metrica di distanza, un **Hard Clustering Ground Truth** (che consente di valutare l'accuratezza di ogni algoritmo con l'indice di Rand) e un Rough Clustering possiamo calcolare i due seguenti parametri:

- **Low**, sarebbe la distanza nel caso migliore (ossia, l'Hard Clustering compatibile con il Rough Clustering che è meno diverso dal Clustering Ground Truth)
- **Upp**, la distanza nel caso peggiore (ossia, l'Hard Clustering compatibile con il Rough Clustering che è più diverso dal Clustering Ground Truth).

A questo punto i due valori vengono calcolati come le medie pesate dei low e upp per i Rough Clustering compatibili con il Fuzzy Clustering di partenza. Normalmente questo calcolo viene fatto attraverso l'enumerazione di tutti i possibili Rough Clustering compatibili con il Fuzzy di partenza e le rispettive probabilità. Tuttavia è stato deciso di applicare una approssimazione, che si basa comunque sull'indice di Rand, in cui vengono campionati un certo numero di Rough Clustering compatibili con le rispettive probabilità, che sono le frequenze empiriche date dal numero di volte che un certo valore di distanza è uscito diviso per il numero totale di ricampionamenti. Nel caso del Fuzzy Clustering, questo può essere visto come distribuzione di Hard Clustering perché i Rough Clustering compatibili sono sempre di tipo Hard. Le probabilità vengono calcolate con l'idea che per ogni istanza si sceglie uno dei cluster col grado di membership > 0 e la probabilità di quell'assegnamento è il prodotto di quei gradi di membership e, di conseguenza, si avrà che Low e Upp saranno sempre uguali perché sia il Clustering Ground Truth, che i clustering con cui lo si confronta, sono sempre di tipo Hard. Di seguito la formula del calcolo appena descritto:

$$P_{fuzzy} = \prod_{z \in X} \mu_x(C(X))$$

in cui X è il dataset e C(X) sarebbe la rappresentazione del Soft Clustering come una distribuzione di Hard Clustering. Con questo calcolo delle probabilità si otterranno, quindi, i valori di Low e Upp che consentono di effettuare delle valutazioni sull'algoritmo che stiamo prendendo in esame. Infatti più si avvicinano a 0 e più l'algoritmo è buono: per semplicità chiameremo più comunemente **errore** questo valore che è stato appena calcolato. [4] [5]

Ecco i risultati che ho ottenuto relativi ai tempi di esecuzione e al valore di errore dopo aver eseguito tutti e quattro gli algoritmi per ognuno degli undici dataset impostando a dieci il numero di iterazioni. Di seguito i risultati di media e deviazione standard dei tempi di esecuzione in forma tabellare:

Time	**FCM-σ**	**KFCM**	**CFCM**	**csiFCM**
Ceramic	0.046 ± 0.004	0.039 ± 0.001	0.066 ± 0.001	0.069 ± 0.0
Crowdsourced	0.759 ± 0.007	0.573 ± 0.005	0.552 ± 0.021	0.604 ± 0.006
Ecoli	1.412 ± 0.019	1.07 ± 0.009	0.912 ± 0.008	1.045 ± 0.01
Glass	0.718 ± 0.007	0.553 ± 0.02	0.465 ± 0.005	0.549 ± 0.005
Iris	0.124 ± 0.002	0.098 ± 0.002	0.092 ± 0.001	0.144 ± 0.002
Wdbc	0.23 ± 0.006	0.196 ± 0.008	0.28 ± 0.007	0.371 ± 0.003
Wine	0.149 ± 0.005	0.116 ± 0.002	0.109 ± 0.003	0.171 ± 0.003
Winequality-red	2.821 ± 0.02	2.009 ±0.023	1.667 ± 0.01	1.863 ± 0.007
Winequality-white	2.588 ± 0.022	1.953 ± 0.011	1.642 ± 0.01	1.918 ± 0.041
Yeast	9.364 ± 0.108	6.953 ± 0.071	5.841 ± 0.041	6.39 ± 0.03
Optdigits	3.486 ± 0.023	2.566 ± 0.024	2.067 ± 0.018	2.184 ± 0.02

Dai risultati ottenuti è possibile affermare che i tempi di esecuzione siano in qualche modo proporzionali alla complessità del dataset che si prende in esame. Inoltre nell'81% dei casi l'algoritmo **CFCM è risultato essere il più veloce** nel processo di clustering. Invece nel 63% dei casi l'algoritmo **FCM-σ è risultato essere il più lento**. Le cause di ciò sono dovute al fatto che calcolare la funzione robusta di FCM-σ per ogni iterazione risulta computazionalmente molto più oneroso che controllare il q-vicinato di CFCM una sola volta.

Di seguito i risultati di media e deviazione standard del valore di errore in forma tabellare:

Error	**FCM-σ**	**KFCM**	**CFCM**	**csiFCM**
Ceramic	0.431 ± 0.056	0.5 ± 0.001	0.442 ± 0.027	0.464 ± 0.044
Crowdsourced	0.284 ± 0.0	0.284 ± 0.0	0.285 ± 0.002	0.285 ± 0.001
Ecoli	0.323 ± 0.001	0.327 ± 0.0	0.317 ± 0.002	0.318 ± 0.003
Glass	0.327 ± 0.004	0.328 ± 0.0	0.325 ± 0.004	0.327 ± 0.003
Iris	0.365 ± 0.039	0.403 ± 0.038	0.372 ± 0.022	0.412 ± 0.035
Wdbc	0.41 ± 0.011	0.5 ± 0.0	0.428 ± 0.027	0.409 ± 0.075
Wine	0.404 ± 0.016	0.446 ± 0.0	0.415 ± 0.016	0.415 ± 0.017
Winequality-red	0.391 ± 0.004	0.388 ± 0.0	0.389 ± 0.002	0.39 ± 0.002
Winequality-white	0.363 ± 0.002	0.362 ± 0.0	0.363 ± 0.001	0.363 ± 0.001
Yeast	0.278 ± 0.0	0.278 ± 0.001	0.279 ± 0.001	0.28 ± 0.001
Optdigits	0.816 ± 0.0	0.816 ± 0.0	0.816 ± 0.0	0.816 ± 0.0

Dai risultati ottenuti, ed osservando i valori di errore, è possibile affermare che esiste una sorta di proporzionalità inversa tra il numero di features e le performance ottenute da ogni algoritmo. Infatti all'aumentare delle features sembra che la performance vada diminuendo, e che quindi il risultato di clustering non sia ottimale. Tuttavia è opportuno considerare che nella maggior parte dei casi si è ottenuto un risultato tendente più verso lo 0 che verso l'1. Infatti è compreso tra 0.28 e 0.45, quindi il risultato di clustering è più che accettabile per quasi tutti i dataset. Nonostante gli algoritmi abbiano avuto tempi di

esecuzione differenti tra loro è doveroso sottolineare che **l'errore ottenuto è pressochè identico** nonostante ogni algoritmo abbia una logica applicativa diversa dall'altro e che quindi, in ottica di un'altra applicazione, **converrebbe utilizzare gli algoritmi coi tempi di esecuzione più bassi**.

6.4 Grafici

In conferma dei risultati appena mostrati e in conferma delle considerazioni dette fin'ora sono stati allegati dei grafici che mostrano il tempo di esecuzione, l'errore di ogni algoritmo ed infine il grafico che mostra la complessità del dataset in funzione ai tempi di esecuzione.

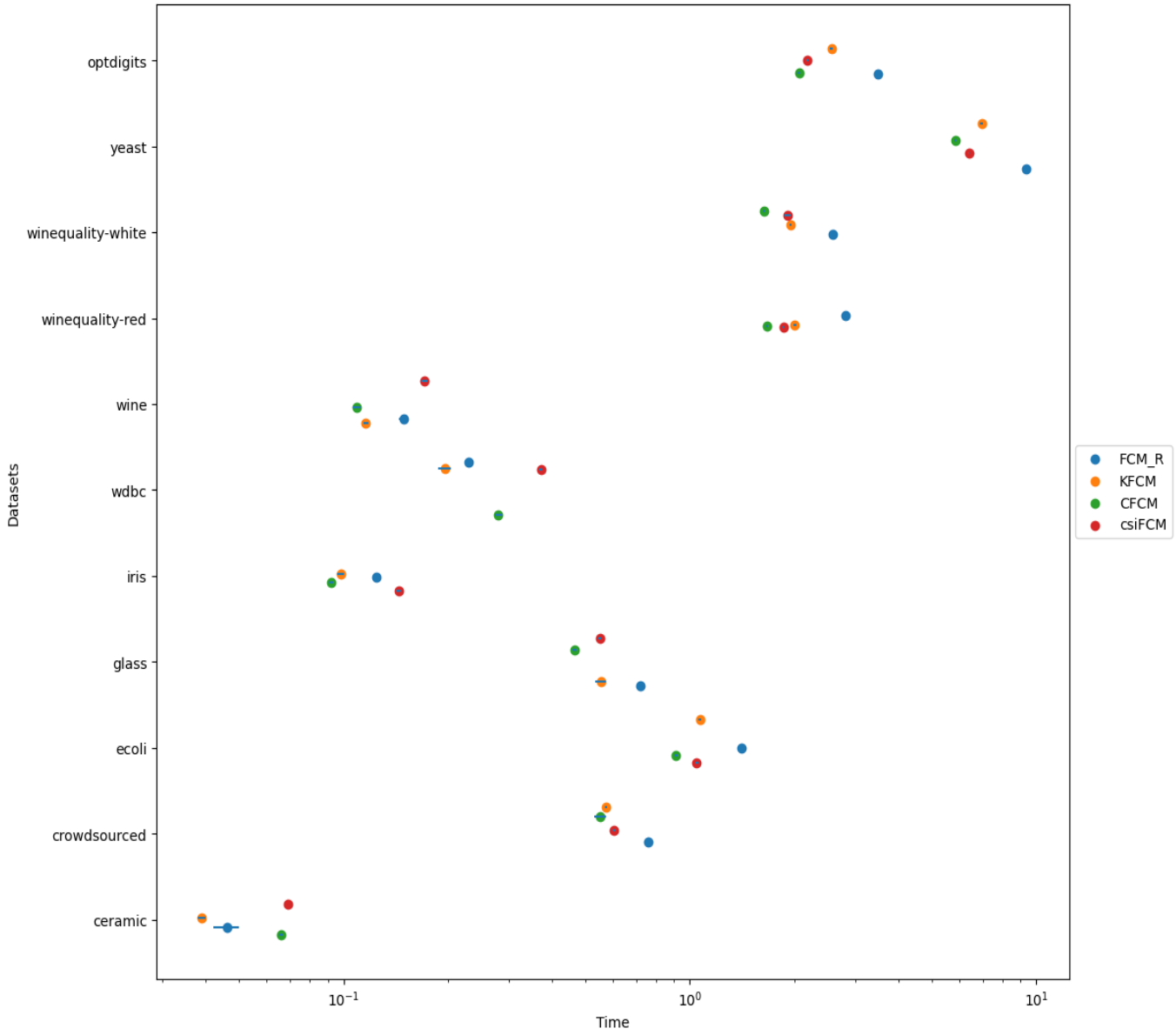

Figura 6.1: Grafico di media e deviazione standard dei tempi di esecuzione

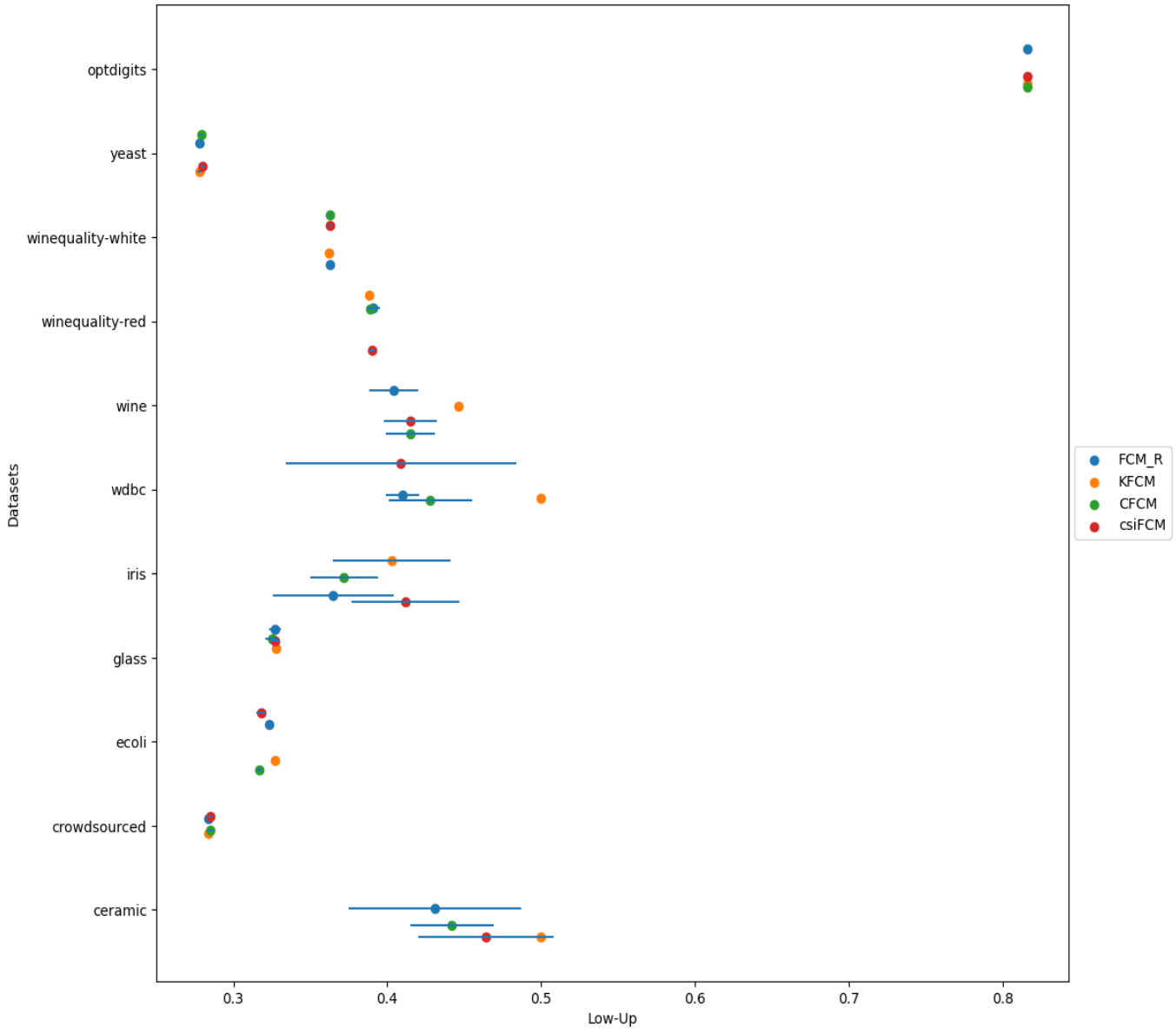

Figura 6.2: Grafico di media e deviazione standard dell'errore ottenuto

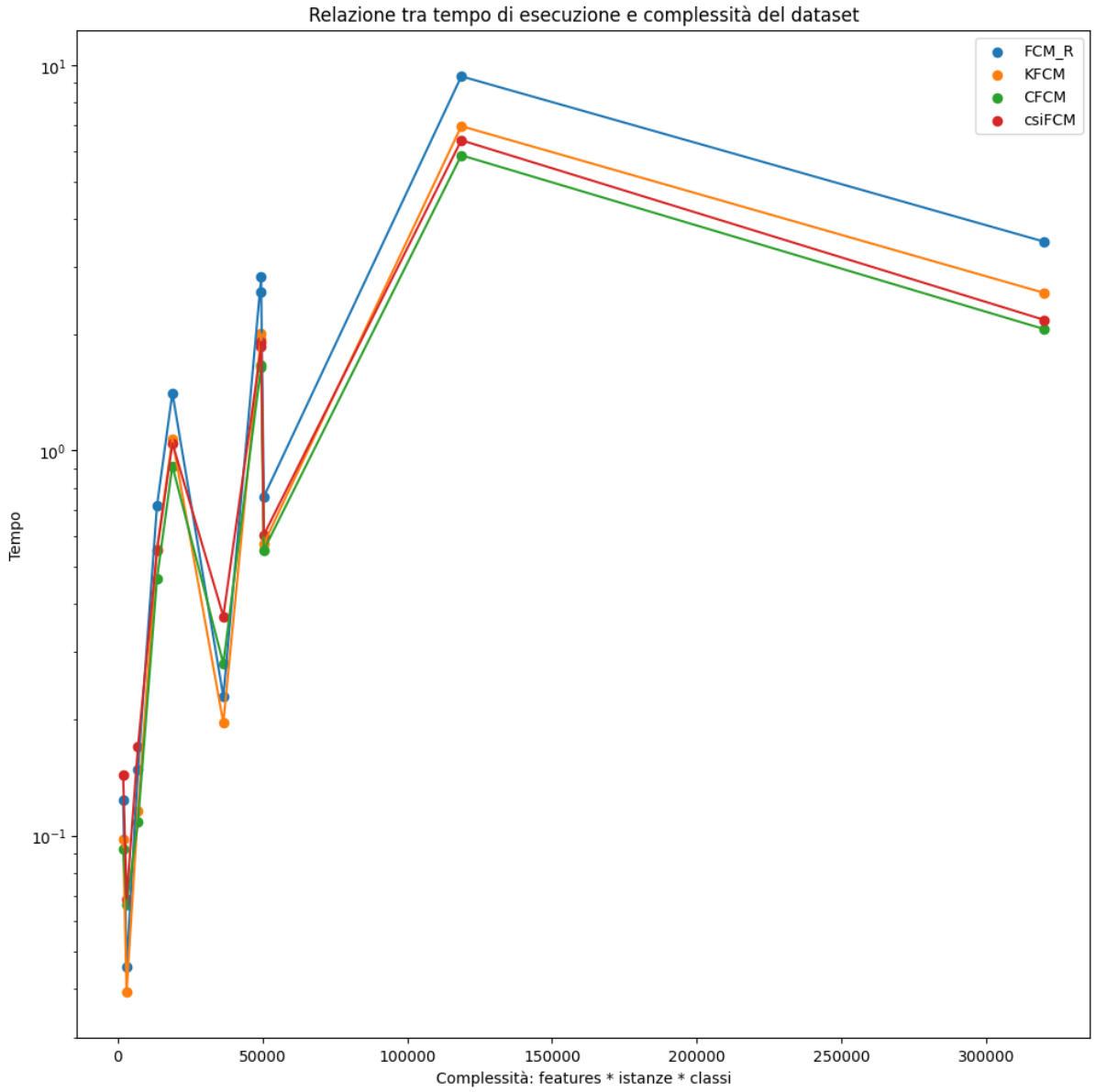

Figura 6.3: Grafico della complessità del dataset legato al tempo di esecuzione

Come ci si aspettava è possibile affermare che **i tempi di esecuzione degli algoritmi aumentano** vertiginosamente all'**aumentare della complessità** del dataset preso in esame. Nonostante vi siano degli intervalli di crescita e decrescita questi sono dovuti al fatto che in alcuni casi si ha un numero di features maggiore di quello delle istanze o viceversa. Per esempio, il penultimo gruppo di punti corrisponde al dataset Yeast (1483 x 8 x 10) mentre l'ultimo gruppo corrisponde al dataset Optical Recognition (500 x 64 x 10). Tuttavia il dataset che ha impiegato più tempo è Yeast proprio perchè ha quasi il triplo del numero di istanze di Optical Recognition. Sebbene i dataset abbiano comunque complessità diverse tra loro, il tempo di esecuzione non è poi così distaccato l'uno dall'altro; quindi l'elemento **più influente sui tempi di esecuzione** di ogni algoritmo è proprio il **numero di features**.

Di seguito, invece, sono stati creati dei plot per mostrare la **similarità** o le differenze sostanziali tra un algoritmo e l'altro. Come avevamo visto poco fa, infatti, i valori di errore risultavano essere molto simili tra loro e nella maggior parte dei casi anche poco distanti. Statisticamente, per quanto riguarda l'errore, gli algoritmi sono molto simili tra loro e non c'è un reale motivo per preferire l'utilizzo di uno piuttosto che dell'altro. Per quanto riguarda i tempi di esecuzione è mostrata la classifica che conferma i tempi riportati nella tabella precedente e che sottolinea, a coppie di 2, la similarità dei tempi di esecuzione tra un algoritmo e il suo successivo in classifica.

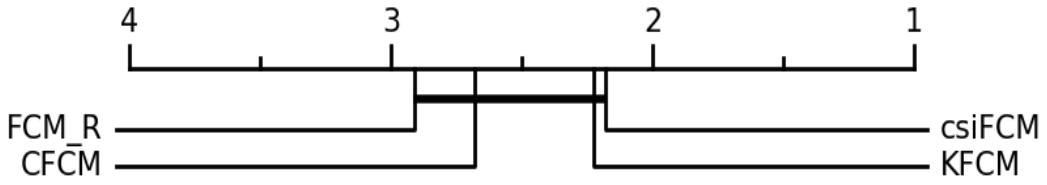

Figura 6.4: Grafico di similirità del valore di errore

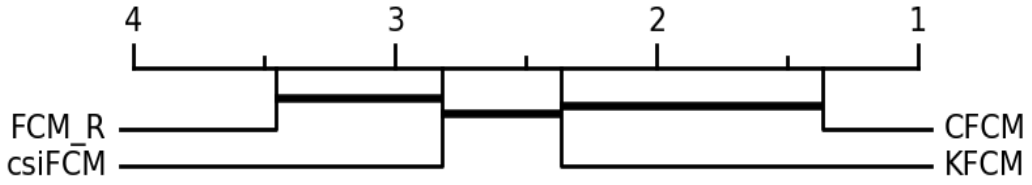

Figura 6.5: Grafico di similarità dei tempi di esecuzione

Capitolo 7

Conclusione

Questa sezione esprime le mie considerazioni personali sulle esperienza di tirocinio e i possibili sviluppi futuri.

7.1 Bilancio dell'esperienza

Il bilancio dell' esperienza è stato positivo e molto interessante, nonostante non vi sia stata la possibilità di lavorare direttamente in sede quotidianamente. Gli incontri svolti col tutor durante la fase di studio e sviluppo sono stati fondamentali per apprendere come implementare al meglio ogni algoritmo. Ho avuto la possibilità di mettere mano a tecnologie che sono considerate standard assoluti nel loro settore della Data Science e dell'AI. Basti pensare che, secondo un'intervista di Gitnux[13] l'80% degli sviluppatori utilizza Python e secondo un sondaggio condotto nel 2020 da Kaggle[16], una piattaforma online popolare tra i data scientist, ha rivelato che oltre il 75% dei professionisti della Data Science utilizza Python come linguaggio principale per il proprio lavoro, di cui:

- il 74% utilizza Jupyter Notebook
- il 33% utilizza Visual Studio Code
- il 32% utilizza Pycharm

Inoltre l'82% degli sviluppatori nel mondo utilizza la libreria Scikit-learn per sviluppare nell'ambito della Data Science e dell'AI.

7.2 Obiettivi raggiunti

Al termine dello stage sono riuscito a raggiungere la maggior parte degli obiettivi che mi ero inizialmente prefissato. Ho potuto toccare con mano cosa vuol dire studiare e applicarsi nel tradurre una logica in un vero e proprio algoritmo analizzando tutte le dinamiche e le funzionalità che ciascuno di essi doveva svolgere. Inoltre, ho visto tutte le dinamiche di scelta e preparazione dei dataset per la fase di analisi e sviluppo. Sono riuscito ad effettuare una analisi completa di tutti gli algoritmi andando a valutarne i tempi di esecuzione ed il loro errore in funzione al dataset che essi stavano esaminando. Posso, dunque, unicamente dire di essere particolarmente grato per l'opportunità, di essere soddisfatto del mio lavoro e anche felice di aver ottenuto dei buoni risultati.

7.3 Sviluppi futuri

In quest'ultima sezione della tesi vengono proposti alcuni utili aggiornamenti che potrebbero migliorare le performance di clustering degli algoritmi.

7.3.1 Altre varianti ibride

Vista la somiglianza dei risultati ottenuta dagli algoritmi implementati sarebbe interessante vedere se le versioni che fondono entrambe le logiche creando degli algoritmi "ibridi" riescano effettivamente a migliorare la qualità del clustering. In particolare si potrebbe pensare di sviluppare **KFCM-σ (Robust Kernel Fuzzy C-Mean)** che mescola la logica di KFCM, applicando la funzione di kernel migliorando la separabilità dei cluster, a quella di FCM-σ, che è meno sensibile ai valori anomali grazie all'uso della funzione robusta. Un altro possibile algoritmo "ibrido" interessante da implementare, secondo me, potrebbe essere **CKFCM (Credibilistic Kernelized Fuzzy C-Means)** in cui avremmo un'analisi spaziale del q-vicinato dove però viene applicata la funzione di kernel per migliorare la separabilità. In questo modo avremmo sia i benefici che porta un'analisi per densità che i benefici portati dall'applicare una funzione di ordine superiore.

7.3.2 Altri algoritmi

Un'altra possibilità sarebbe quella di implementare altri algoritmi che abbiano delle logiche completamente differenti adottando magari una logica di densità spaziale come **ISFCM (Improved Spatial Fuzzy C-Means Clustering)** incorporando informazioni spaziali per migliorare la qualità del clustering. Oppure potrebbe avere un impatto simile l'applicare una logica intuizionistica come in **IFCM (Intuitionistic Fuzzy C-Means)** in cui ogni punto dati è assegnato non solo a un cluster con un certo grado di appartenenza, ma anche a un cluster secondario con un grado di non appartenenza e ad anche un grado di incertezza/ambiguità. A differenza di FCM, infatti, lo scopo è proprio quello di minimizzare l'incertezza. Un altro algoritmo interessante che potrebbe avere una logica di densità è **DPIFCM (Density Based Probabilistic Intuitionistic Fuzzy C-Means)** che, stimando la densità per ogni regione dello spazio dei dati e utilizzando essa per calcolare i valori di probabilità dei centroidi dei cluster, potrebbe in qualche modo migliorare l'efficienza del clustering. Infine sarebbe interessante vedere se, applicando la logica Fuzzy di Tipo 2, aumenterebbero le performance. Infatti l'algoritmo **T2FCM (Type-2 Fuzzy C-Means)**, utilizzando insiemi Fuzzy di Tipo 2, avrà la funzione di appartenenza di ciascun elemento che dipende non solo dall'elemento stesso, ma anche dal contesto in cui si trova. Ciò consente di rappresentare in modo più preciso l'incertezza e l'ambiguità nei dati. Tuttavia, una mia idea di possibile soluzione, è quella di pensare a un tipo di implementazione che vada a comparare la bontà del clustering confrontando diversi modelli in cui si hanno tutte le possibili combinazioni di features da prendere per l'analisi (ignorandone totalmente alcune) in modo tale da trovare il miglior modello che si adatti al training set con l'obiettivo di evitare l'overfitting, massimizzando il più possibile lo score.

Ringraziamenti

Vorrei riservare questo spazio finale della mia tesi di laurea ai ringraziamenti verso tutti coloro che hanno contribuito, con il loro instancabile supporto, alla realizzazione della stessa. Per prima cosa, vorrei ringraziare il mio relatore D.Ciucci e il mio correlatore A.Campagner, per i loro preziosi consigli e per la loro disponibilità. Grazie per avermi fornito spunti fondamentali nella stesura di questo lavoro e per avermi indirizzato nei momenti di indecisione. Inoltre vorrei ringraziare infinitamente i miei genitori, la mia famiglia e la mia ragazza, che mi hanno sempre motivato a dare il meglio e che mi hanno sempre supportato nei momenti di gioia e nei momenti più difficili di questo percorso universitario. Infine, dedico questa tesi a me stesso, al mio caro papà, ai miei sacrifici e alla mia tenacia che mi hanno permesso di arrivare fin qui.

Bibliografia

[1] S. Askari. «Fuzzy C-Means clustering algorithm for data with unequal cluster sizes and contaminated with noise and outliers: Review and development». In: *Expert Systems with Applications* 165 (2021).

[2] Apprendimento automatico. ultima visita: aprile 2023. URL: https://it.wikipedia.org/wiki/Apprendimento_automatico.

[3] Vito Fabrizio Brugnola. 2019. URL: https://www.prometheus-studio.it/prometheus_blog_wp/2019/08/15/cosa-e-la-fuzzy-logic-o-logica-sfumata/.

[4] A. Campagner, D. Ciucci e T. Denœux. *A Distributional Approach for Soft Clustering Comparison and Evaluation*. Vol. 13506 LNAI. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2022.

[5] A. Campagner, D. Ciucci e T. Denœux. «A general framework for evaluating and comparing soft clusterings». In: *Information Sciences* 623 (2023).

[6] Clustering. ultima visita: aprile 2023. URL: https://it.wikipedia.org/wiki/Clustering.

[7] *Clustering | Introduction, Different Methods, and Applications (Updated 2023)*. ultima visita: aprile 2023. URL: https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/#What_Is_Clustering?.

[8] Fuzzy Clustering. ultima visita: aprile 2023. URL: https://en.wikipedia.org/wiki/Fuzzy_clustering.

[9] Visual Studio Code. ultima visita: giugno 2023. URL: https://it.wikipedia.org/wiki/Visual_Studio_Code.

[10] Compilatore. ultima visita: maggio 2023. URL: https://it.wikipedia.org/wiki/Compilatore.

[11] *Density-based clustering in data minin*. ultima visita: aprile 2023. URL: https://www.javatpoint.com/density-based-clustering-in-data-mining.

[12] Logica Fuzzy. ultima visita: aprile 2023. URL: https://it.wikipedia.org/wiki/Logica_fuzzy.

[13] Gitnux. *Linguaggi di programmazione: elenco definitivo, statistiche e tendenze sul mondo della programmazione*. ultima visita: giugno 2023. URL: https://blog.gitnux.com/it/statistiche-sui-linguaggi-di-programmazione/.

[14] A. Gosain e S. Dahiya. «Performance Analysis of Various Fuzzy Clustering Algorithms: A Review». In: *Procedia Computer Science*. Vol. 79. 2016.

[15] Interprete. ultima visita: maggio 2023. URL: https://it.wikipedia.org/wiki/Interprete_(informatica).

[16] Kaggle. *State of Data Science and Machine Learning 2020*. ultima visita: giugno 2023. URL: https://www.kaggle.com/kaggle-survey-2020.

[17] Python. ultima visita: maggio 2023. URL: https://it.wikipedia.org/wiki/Python.

[18] Indice di Rand. ultima visita: luglio 2023. URL: https://en.wikipedia.org/wiki/Rand_index.

[19] UCI Machine Learning Repository. ultima visita: maggio 2023. URL: https://archive.ics.uci.edu/ml/datasets.php.

[20] A. K. Varshney, P. K. Muhuri e Q. M. D. Lohani. «Density-based IFCM along with its interval valued and probabilistic extensions, and a review of intuitionistic fuzzy clustering methods». In: *Artificial Intelligence Review* 56.4 (2023).