



Sistemi Complessi e Incerti

Approfondimento su Fuzzy Sets per Sistemi Incerti

Fuzzy Clustering per il Supporto alle Decisioni in
Sistemi di Raccomandazione

AA 2024/2025

di



Andrea Falbo

a.falbo7@campus.unimib.it

Università degli Studi di Milano-Bicocca

Indice

Capitolo 1

Introduzione

La teoria degli *insiemi fuzzy*, introdotta da Zadeh negli anni '60 [Goguen_1973], ha rappresentato una svolta nella modellazione dell'incertezza e dell'imprecisione nei sistemi informatici. La sua caratteristica distintiva risiede nella possibilità di esprimere l'appartenenza graduale di un elemento a un insieme, superando la rigidità binaria degli insiemi classici. Questo approccio ha trovato applicazioni in una vasta gamma di domini, tra cui il controllo automatico, l'analisi semantica, il data mining e, più recentemente, i *sistemi di raccomandazione*.

I *sistemi di raccomandazione* sono oggi componenti centrali delle piattaforme digitali, come Netflix, Amazon o Spotify, dove svolgono il compito di suggerire contenuti rilevanti agli utenti, basandosi sulle loro preferenze esplicite o implicite [554776b9726f44c582f01d870cefd26b]. Tuttavia, la natura ambigua e incerta delle preferenze umane rende complesso modellarle con tecniche tradizionali. In tale contesto, la logica fuzzy offre strumenti promettenti per rappresentare tali incertezze e preferenze in modo più flessibile e realistico [EKEL2006179].

L'*Obiettivo* del seguente approfondimento è esplorare l'utilizzo del **fuzzy clustering** nei sistemi di raccomandazione, con particolare attenzione all'ambito della raccomandazione personalizzata di contenuti audiovisivi. In particolare, si propone un'estensione metodologica di quanto svolto in [KOOHI2016134], dove si applica l'algoritmo *Fuzzy C-Means (FCM)* per raggruppare gli utenti in cluster sovrapposti, modellando così la possibilità che un utente appartenga simultaneamente a più gruppi di preferenza.

A partire da questo contributo, il progetto realizzato si propone di esplorare l'impatto di diverse tecniche di normalizzazione, parametri di clustering e rumore sulle performance del sistema, fornendo un framework riproducibile per valutare il clustering fuzzy nel filtraggio collaborativo tramite metriche quantitative e analisi visive.

La seguente relazione è strutturata come segue:

- **Capitolo ??** presenta i fondamenti teorici relativi alla logica fuzzy, al fuzzy clustering e ai sistemi di raccomandazione;
- **Capitolo ??** descrive l'analisi esplorativa del dataset MovieLens 100k;
- **Capitolo ??** descrive l'implementazione del sistema di raccomandazione fuzzy;
- **Capitolo ??** presenta i risultati degli esperimenti condotti;
- **Capitolo ??** conclude il lavoro e propone linee di ricerca future.

Capitolo 2

Fondamenti Teorici

2.1 Parte I: Logica Fuzzy e Clustering

2.1.1 Insiemi Fuzzy

Un insieme fuzzy [Goguen_1973] è una generalizzazione dell'insieme classico in cui la nozione di appartenenza è graduata e rappresentata da una funzione $\mu_A(x) : X \rightarrow [0, 1]$, denominata come *funzione di appartenenza*. Il valore restituito indica il grado con cui un elemento x appartiene all'insieme fuzzy A .

Interpretazione di un insieme fuzzy

L'idea centrale dei sistemi fuzzy non è solo usare valori tra 0 e 1, ma fornire un'interpretazione semantica adeguata alla funzione di appartenenza. Secondo [DUBOIS1997141], tali interpretazioni possono rappresentare similarità, preferenze o incertezza:

- **Similarità:** L'utilizzo del grado di appartenenza come similarità sottintende l'idea che ci siano elementi tipici di un determinato insieme, altri che sicuramente non vi appartengono e una gradualità di sfumature per i rimanenti.
- **Incerteza:** La funzione di appartenenza viene utilizzata per dotare di un grado l'incertezza relativa ad un qualche fatto.
- **Preferenza:** Il grado di appartenenza viene utilizzato per esprimere preferenze graduali rispetto a determinati criteri. Questa interpretazione trova il suo principale utilizzo in teoria delle decisioni e ha favorito lo sviluppo di un'ampia letteratura, in particolare sulle modalità di aggregare diversi insiemi fuzzy.

Estensione e operazioni fuzzy

Le operazioni insiemistiche di unione, intersezione e complemento vengono estese agli insiemi fuzzy con il metodo pointwise, ovvero fissando un elemento ed utilizzando solo il suo grado di appartenenza agli insiemi fuzzy coinvolti:

- **Intersezione:** $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$
- **Unione:** $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$
- **Complemento:** $\mu_{\neg A}(x) = 1 - \mu_A(x)$

Tali operatori possono essere generalizzati tramite t-norme e t-conorme, per ottenere maggiore flessibilità computazionale.

Variabili Linguistiche

L'approccio linguistico fuzzy [890332] è utile per modellare preferenze incerte o vaghe, utilizzando variabili linguistiche [ZADEH1975199], i cui termini sono rappresentati tramite funzioni di appartenenza fuzzy.

Granular Computing

Una disciplina emersa grazie agli insiemi fuzzy è la granular computing [doi:10.1142/9789814261302], cioè la disciplina che si occupa di rappresentare e processare l'informazione sotto forma di qualche tipo di aggregato, generalmente chiamato granulo di informazione, risultato di un processo di astrazione o estrazione di conoscenza dai dati.

Fuzzy Clustering

L'obiettivo del *clustering* è raggruppare elementi simili tra loro o simili ad uno o più elementi centrali, in modo che i diversi gruppi siano il più possibile omogenei al loro interno e allo stesso tempo diversi tra loro. È quindi uno dei metodi utilizzati nella granular computing per definire i granuli.

Quel che si ottiene nella variante fuzzy prende il nome di *Fuzzy Clustering* [dunn1973fuzzy], dove i diversi gruppi non sono separati tra loro ma possono essere sovrapposti, e dunque un oggetto può appartenere a gruppi diversi, con un determinato grado di appartenenza. L'algoritmo più noto è il **Fuzzy C-Means (FCM)** [BEZDEK1984191], che generalizza il classico K-Means minimizzando la funzione:

$$J_m = \sum_{i=1}^N \sum_{j=1}^C u_{ij}^m \cdot \|x_i - c_j\|^2$$

dove u_{ij} è il grado di appartenenza dell'elemento x_i al centro c_j , e $m > 1$ è il parametro di fuzzificazione. L'algoritmo aggiorna iterativamente i centroidi e le appartenenze per ottenere una partizione fuzzy dei dati.

Il parametro di fuzzificazione m controlla la "morbidezza" della partizione: per $m \rightarrow 1$ il clustering tende a diventare più rigido, simile al K-Means classico, mentre per valori crescenti di m l'appartenenza degli oggetti ai cluster diventa più uniforme. Valori tipici di m sono compresi tra 1.5 e 2.5 [BEZDEK1984191].

2.2 Parte II: Sistemi di Raccomandazione

2.2.1 Sistemi di raccomandazione

I *sistemi di raccomandazione* sono stati introdotti nel 1992 da Goldberg[10.1145/138859.138867]. Secondo [burke2002hybrid], un sistema di raccomandazione è definito come "qualsiasi sistema che produce raccomandazioni personalizzate in output o ha l'effetto di guidare l'utente in modo personalizzato verso oggetti interessanti o utili in un ampio spazio di opzioni possibili".

Classificazione

Una delle classificazioni dei sistemi di raccomandazione più popolari è quella proposta da Bobadilla et al. [bobadilla2013recommender], che distingue:

- **Filtraggio demografico:** utilizza attributi dell'utente (es. età, sesso, localizzazione) per identificare preferenze simili tra utenti con caratteristiche comuni [bobadilla2013recommender, zhao2014we].
- **Filtraggio collaborativo:** sfrutta unicamente le valutazioni espresse dagli utenti [adomavicius2005toward]. I suggerimenti sono generati confrontando le valutazioni di utenti simili.
- **Raccomandazione basata sul contenuto:** utilizza descrizioni degli oggetti e un profilo dell'utente per raccomandare elementi simili a quelli già apprezzati [de2015semantics, lops2011content].
- **Approcci ibridi:** combinano più paradigmi, come filtraggio collaborativo e demografico [vozalis2007using], o collaborativo e basato sul contenuto [balabanovic1997fab]. Burke [burke2002hybrid] ha identificato sei tecniche principali di ibridazione.

Filtraggio Collaborativo

Il *filtraggio collaborativo* (CF) è una delle tecniche più diffuse nei sistemi di raccomandazione e si basa sull'idea che utenti che hanno espresso giudizi simili in passato tenderanno a condividere gusti simili anche in futuro [adomavicius2005toward].

Gli algoritmi CF si dividono generalmente in due macro-categorie [LU20121]:

- **Approcci Memory-Based:** utilizzano l'intero database di valutazioni utente-oggetto per generare raccomandazioni in tempo reale. Le raccomandazioni vengono prodotte direttamente confrontando la similarità tra utenti o tra oggetti, senza costruire un modello esplicito. Questi approcci si distinguono ulteriormente in:
 - **User-Based:** confrontano le valutazioni di un utente target con quelle di altri utenti, individuando i “vicini” più simili. Le preferenze degli utenti simili vengono poi aggregate per prevedere le valutazioni dell'utente target.
 - **Item-Based:** analizzano la similarità tra gli oggetti (es. film, prodotti) sulla base delle valutazioni espresse dagli utenti, ipotizzando che un utente apprezzerà oggetti simili a quelli già valutati positivamente.
- **Approcci Model-Based:** costruiscono un modello predittivo a partire dai dati storici, spesso impiegando tecniche di machine learning come clustering, regressione o decomposizione matriciale. I modelli ottenuti operano su una rappresentazione ridotta dei dati, e sono in grado di affrontare problemi come la scalabilità e la sparsità della matrice delle valutazioni.

Approccio User-Based

Nel CF *user-based*, si parte da una matrice utente-oggetto $n \times m$ contenente le valutazioni espresse da n utenti su m oggetti. Quando un nuovo utente entra nel sistema, si identificano utenti “simili” (i *vicini*) e si usano le loro valutazioni per predire le preferenze dell'utente target.

La similarità tra utenti viene solitamente calcolata con misure classiche come:

- **Pearson correlation coefficient**;
- **Cosine similarity**;
- **Jaccard index**, ecc.

Il coefficiente di Pearson tra due utenti a e b è definito da:

$$\text{sim}(a, b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}}$$

dove P è l'insieme degli item valutati da entrambi, e $r_{a,p}$ è la valutazione dell'utente a sull'oggetto p .

Una volta individuati i vicini, la previsione della valutazione dell'utente target su un oggetto p può essere calcolata con una media pesata:

$$\text{pred}(a, p) = \bar{r}_a + \frac{\sum_{b \in N} \text{sim}(a, b)(r_{b,p} - \bar{r}_b)}{\sum_{b \in N} |\text{sim}(a, b)|}$$

oppure con una semplice media delle valutazioni:

$$\text{pred}(a, p) = \frac{1}{n} \sum_{i=1}^n r_{i,p}$$

Infine, gli oggetti con le valutazioni predette più alte vengono raccomandati all'utente.

2.3 Parte III: Clustering nei Sistemi di Raccomandazione

2.3.1 Clustering nei Sistemi di Raccomandazione

Le misure di similarità, sebbene diffuse, presentano limiti rilevanti:

- possono essere computazionalmente onerose su dataset ampi;
- risultano inefficaci in presenza di dati sparsi (*sparsity problem*);
- non catturano relazioni strutturali latenti tra utenti.

Per superare tali limiti, si ricorre spesso a tecniche di **clustering**, che organizzano gli utenti in gruppi omogenei. A ogni cluster possono essere associati contenuti preferiti, da raccomandare agli utenti che vi appartengono.

Approcci di Clustering

Tra le tecniche di clustering più utilizzate in ambito CF troviamo:

- **K-Means**: produce partizioni disgiunte; semplice ed efficiente, ma assume cluster sferici e richiede k fissato a priori;
- **SOM (Self-Organizing Maps)**: rete neurale non supervisionata che proietta i dati in uno spazio ridotto conservando relazioni topologiche;
- **Fuzzy Clustering (es. FCM)**: consente appartenenze multiple; utile per modellare utenti con gusti sfaccettati.

L'uso del clustering nei sistemi di raccomandazione consente di:

- ridurre la dimensionalità del problema (scalabilità);
- mitigare il problema della sparsità;
- selezionare i vicini tra gli utenti dello stesso cluster, migliorando la coerenza del filtraggio.

Capitolo 3

Analisi Esplorativa del Dataset

In questo capitolo viene presentata l'analisi esplorativa dei dati (EDA) condotta sul dataset MovieLens 100k, con l'obiettivo di comprendere la struttura, le caratteristiche e le criticità dei dati utilizzati per la sperimentazione dei sistemi di raccomandazione. L'analisi è stata svolta tramite lo script Python `eda_movielens100k.py`, che genera automaticamente tutti i plot e le statistiche descritte di seguito.

3.1 Descrizione del Dataset

Il dataset MovieLens 100k contiene 100.000 valutazioni espresse da 943 utenti su 1682 film. Ogni valutazione è un intero da 1 a 5. Sono inoltre disponibili informazioni anagrafiche sugli utenti (età, genere, occupazione) e sui film (titolo, data di uscita, generi associati).

3.1.1 Filtraggio e Preprocessing

Per ottenere una base dati densa e affidabile, durante l'analisi esplorativa sono stati applicati filtri sui dati grezzi:

- Sono stati mantenuti solo gli utenti con almeno 150 valutazioni.
- Sono stati mantenuti solo i film con almeno 150 valutazioni.

Questo ha ridotto il dataset a 24669 valutazioni, 230 utenti e 203 film, rendendo più affidabile l'analisi dei cluster.

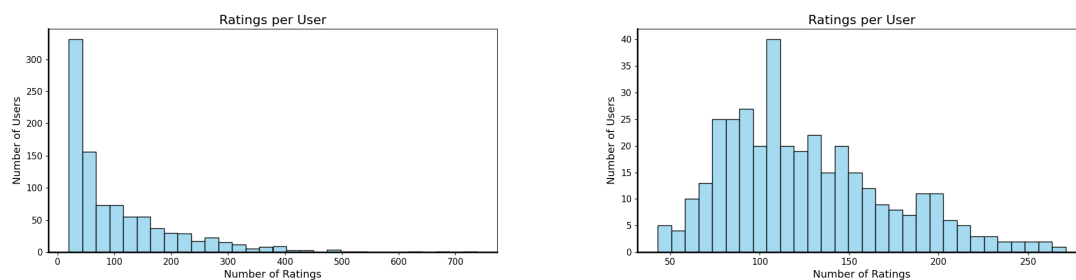


Figura 3.1: Distribuzione del numero di valutazioni per utente, prima e dopo il filtraggio.

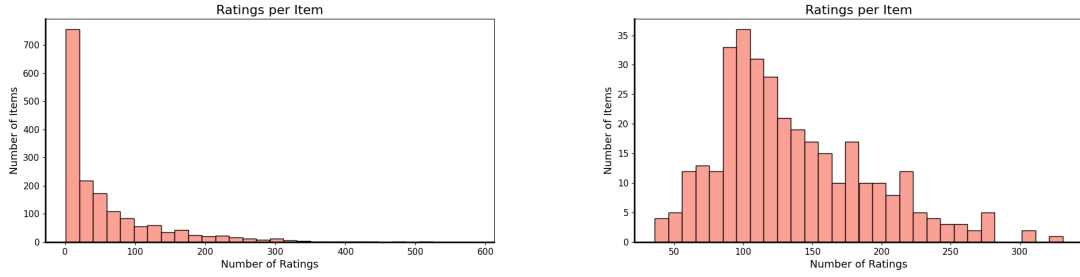


Figura 3.2: Distribuzione del numero di valutazioni per film, prima e dopo il filtraggio.

3.2 Distribuzione delle Valutazioni

La distribuzione delle valutazioni mostra una forte asimmetria verso i valori alti. Questo bias positivo può influenzare le metriche di errore e la qualità delle raccomandazioni.

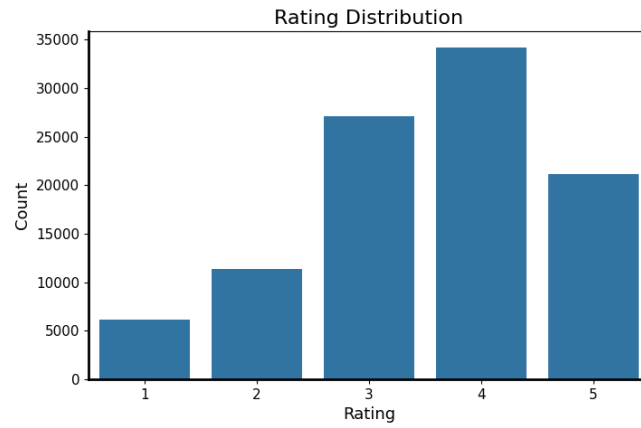


Figura 3.3: Distribuzione delle valutazioni (prima del filtraggio).

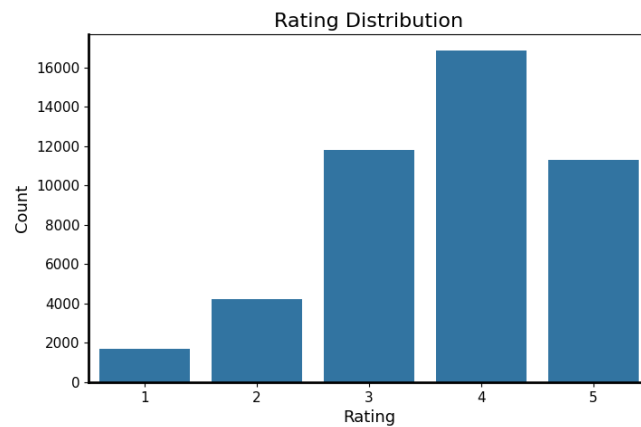


Figura 3.4: Distribuzione delle valutazioni (dopo il filtraggio).

3.2.1 Valutazioni per utente e per film

I boxplot delle valutazioni per utente e per film evidenziano la presenza di outlier e la variabilità tra utenti e tra film.

Si può osservare come il filtraggio ha eliminato i valori più alti, lasciando solo i valori più centrali, e quindi i valori più rappresentativi.

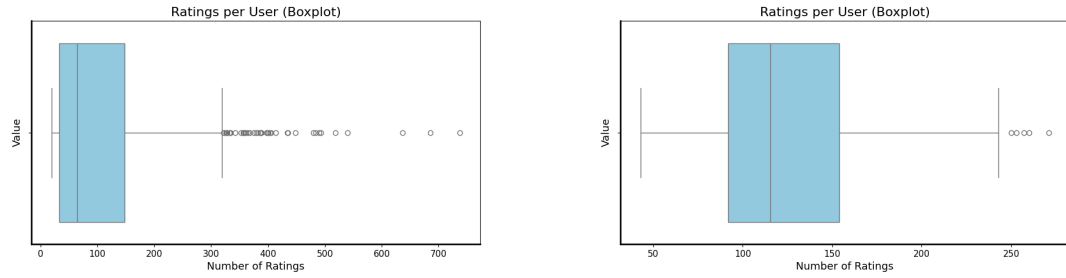


Figura 3.5: Boxplot del numero di valutazioni per utente, prima e dopo il filtraggio.

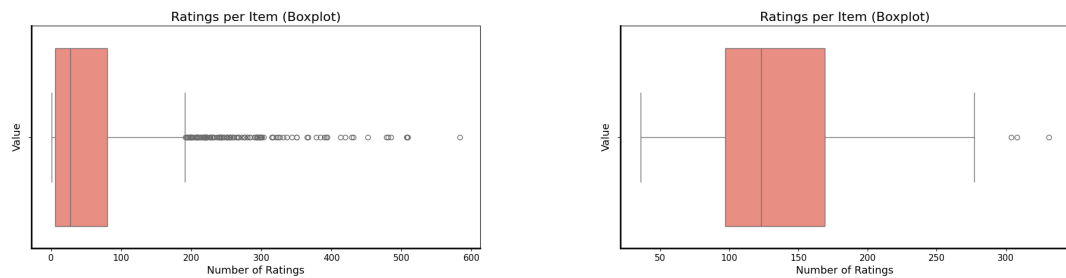


Figura 3.6: Boxplot del numero di valutazioni per film, prima e dopo il filtraggio.

3.3 Sparsità della Matrice Utente-Item

La matrice delle valutazioni è estremamente sparsa: anche dopo il filtraggio, la maggior parte delle possibili coppie utente-film non ha una valutazione. Questo può comportare problemi nell'analisi dei dati e nella generazione di raccomandazioni.

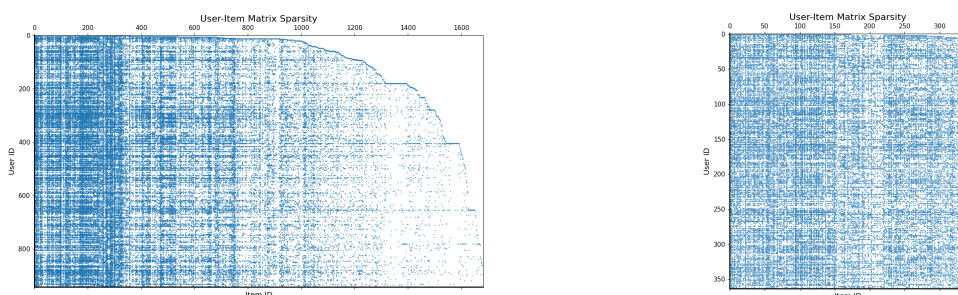


Figura 3.7: Visualizzazione della matrice utente-item, prima e dopo il filtraggio.

3.3.1 Correlazioni tra Utenti e tra Film

Sono state calcolate le matrici di correlazione (Pearson) tra utenti e tra film, su un campione di 50 utenti/film per leggibilità. Come si può osservare, non sembra esserci una correlazione tra utenti e film evidente.

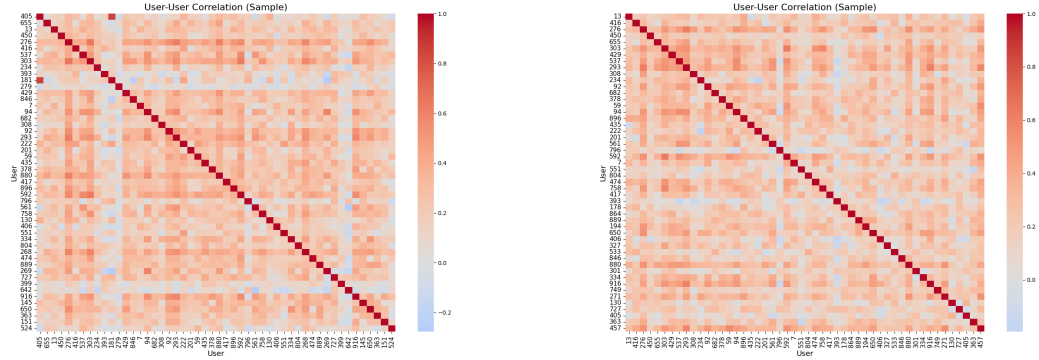


Figura 3.8: Heatmap delle correlazioni tra utenti, prima e dopo il filtraggio.

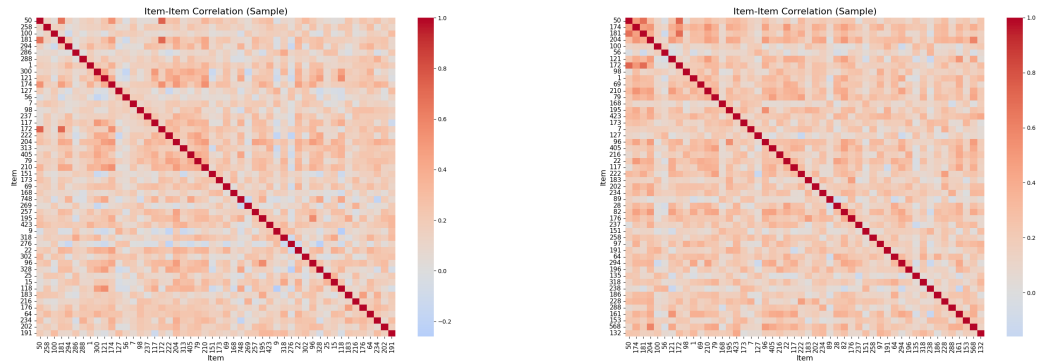


Figura 3.9: Heatmap delle correlazioni tra film, prima e dopo il filtraggio.

3.3.2 Analisi dei Generi

L'analisi dei generi mostra una forte predominanza di alcuni generi (drama, comedy, action), mentre altri sono molto meno rappresentati. Questo può influenzare la varietà delle raccomandazioni.

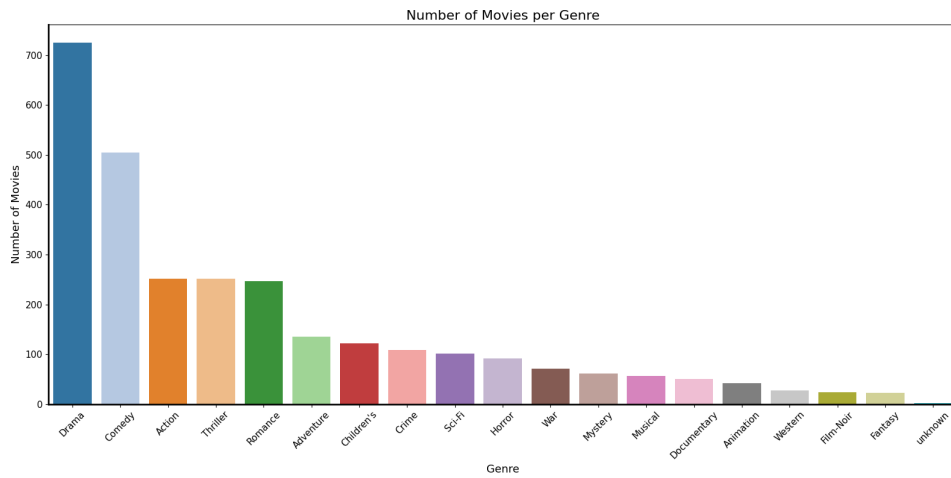


Figura 3.10: Numero di film per genere (prima del filtraggio).

3.4 MovieLens 1M

Sebbene il sistema realizzato supporta l'utilizzo del dataset MovieLens 1M, è stato scelto di utilizzare il dataset MovieLens 100k per la sperimentazione, in quanto più compatto e più facile da gestire. L'analisi di quest'ultimo può essere considerata come un'estensione futura di questo lavoro.

Capitolo 4

Implementazione del Sistema

Nel capitolo seguente vengono riportati alcuni snippet di codice selezionati, utili per illustrare le parti più rilevanti dell'implementazione. Per consultare il codice completo si rimanda alla repository indicata nell'Appendice ??.

4.1 Architettura del Sistema

Il sistema di raccomandazione fuzzy è stato implementato seguendo un'architettura modulare e scalabile, progettata per supportare esperimenti sistematici e riproducibili. L'architettura è organizzata in quattro livelli principali:

- **Livello di Configurazione:** Gestione centralizzata dei parametri sperimentali
- **Livello di Orchestrazione:** Coordinamento del flusso di esecuzione degli esperimenti
- **Livello di Elaborazione:** Componenti per clustering, valutazione e visualizzazione
- **Livello di Utilità:** Funzioni di supporto per caricamento, preprocessing e normalizzazione

4.1.1 Pipeline Sperimentale

Il flusso completo di un esperimento è:

1. **Caricamento Configurazione:** Lettura parametri da JSON
2. **Preparazione Dati:** Caricamento, filtro, split train/test
3. **Loop Parametri:** Per ogni combinazione di:
 - Normalizzazione
 - Numero di cluster
 - Parametro di fuzziness
 - Metodo di clustering
 - Strategia di defuzzificazione
 - Selezione vicini
4. **Clustering:** Applicazione algoritmo selezionato
5. **Predizione:** Calcolo rating predetti

6. **Valutazione:** Calcolo RMSE e MAE
7. **Visualizzazione:** Generazione grafici
8. **Salvataggio:** Risultati e configurazioni

4.2 Esecuzione del Sistema

Il sistema utilizza un file di configurazione JSON centralizzato (`config/config.json`) che controlla tutti gli aspetti dell'esperimento:

Listing 4.1: Esempio di configurazione

```
1 {  
2     "dataset_name": "ml-100k",  
3     "normalizations": ["simple_centering", "zscore_per_user",  
4                         "minmax_per_user", "no_normalization"],  
5     "cluster_values": [4, 6, 8],  
6     "m_values": [1.5, 2.0, 2.5],  
7     "clustering_methods": ["fcm", "kmeans"],  
8     "defuzzification_methods": ["maximum", "cog"],  
9     "neighbor_selection_methods": ["none", "pearson"],  
10    "min_user_ratings": 150,  
11    "min_item_ratings": 150,  
12    "test_size": 0.2,  
13    "max_iter": 3000,  
14    "error": 1e-06  
15 }
```

Questa configurazione consente di sviluppare le analisi in modo rapido e modulare, poiché tutti i parametri rilevanti sono centralizzati in un unico file di configurazione. In questo modo, è possibile modificare facilmente le impostazioni degli esperimenti senza intervenire direttamente sul codice, riducendo il rischio di errori e migliorando l'efficienza del processo di sviluppo. Sebbene quando si utilizza K-Means variare il parametro di fuzziness e metodo di defuzzificazione non abbiano impatto, è risultato più veloce mantenerli nel file di configurazione ed eseguirli piuttosto che gestire il caso specifico.

4.2.1 Orchestrazione degli Esperimenti

La classe `Runner` rappresenta il componente centrale di orchestrazione, in quanto è la classe che si occupa di:

- Caricare la configurazione da eseguire
- Creazione la directory in cui salvare i risultati
- Eseguire tutte le combinazioni dei parametri
- Salvare i risultati ottenuti e la configurazione utilizzata

```
1 class Runner:  
2     def __init__(self, config_path):  
3         self.config = ConfigManager(config_path).load()  
4         timestamp_format = self.config.get("run_timestamp_format",
```

```

5                                     "run_%Y_%m_%d_%H_%M_%S")
6     run_timestamp = datetime.now().strftime(timestamp_format)
7     base_output_dir =
8         os.path.join(self.config.get("output_dir", "output"),
9                         run_timestamp)
10
11     self.config["images_dir"] = os.path.join(base_output_dir,
12                                                self.config.get(
13                                                    "images_subdir",
14                                                    "images"))
15     self.config["results_dir"] = os.path.join(base_output_dir,
16                                                self.config.get(
17                                                    "results_subdir",
18                                                    "results"))
19
20     self.data_manager = DataManager(self.config)
21     self.result_manager = ResultManager(self.config)
22
23     def run(self):
24         R_train, R_test_aligned =
25             self.data_manager.load_and_preprocess()
26
27         for norm_name in normalizations:
28             for c in cluster_values:
29                 for m in m_values:
30                     for clustering_method in clustering_methods:
31                         for defuzz_method in defuzz_methods:
32                             for neighbor_method in neighbor_methods:
33                                 experiment = Experiment(...)
34                                 metrics = experiment.run()
35                                 results[norm_name][str(c)]
36                                     [str(m)][clustering_method]
37                                     [defuzz_method][neighbor_method]
38                                     = metrics
39
40         self.result_manager.save_results(results)

```

Listing 4.2: estratti della classe Runner

4.2.2 Gestione Dati

Il `DataManager`, una volta chiamato dalla classe `Runner`, si occupa di due mansioni principali:

1. Caricare e Preprocessare i dati, tramite il metodo `load_and_preprocess`
2. Normalizzare i dati, tramite il metodo `normalize`

Il primo metodo si occupa di:

- Scegliere quale dataset utilizzare, in base al nome specificato nel file di configurazione

- Chiamare il filtro per rimuovere utenti con meno di `min_user_rating` rating e item con meno di `min_item_rating` rating,
- Eseguire lo split di train size e test size in base al parametro specificato nel file di configurazione
- Allineare il test set affinché le matrici di train e test abbiano stessa dimensione

```

1 def load_and_preprocess(self):
2     if self.config.get('dataset_name') == 'ml-1m':
3         _, R = load_data_1m()
4     else:
5         _, R = load_data_100k()
6
7     R_dense = filter_dense(R,
8                           self.config['min_user_ratings'],
9                           self.config['min_item_ratings'])
10
11     R_train, R_test = split_train_test_per_user(
12         R_dense,
13         test_size=self.config['test_size'],
14         random_state=self.config['random_state']
15     )
16
17     R_test_aligned = R_test.reindex(columns=R_train.columns,
18                                     fill_value=np.nan)
19     return R_train, R_test_aligned

```

Listing 4.3: Metodo `load_and_preprocess` del `DataManager`

Il secondo metodo si occupa di:

- Utilizzare la funzione di normalizzazione corretta
- Calcolare la media globale dei rating nel dataset di training
- Controlla se la media dell'utente è valida
- Se l'utente ha media valida la usa, altrimenti usa la media globale

```

1 def normalize(self, R_train, R_test_aligned, norm_func=None):
2     if norm_func is not None:
3         R_train_norm = norm_func(R_train)
4         R_test_norm = norm_func(R_test_aligned)
5     else:
6         R_train_norm = R_train.astype(float)
7         R_test_norm = R_test_aligned.astype(float)
8
9     global_mean = R_train_norm.stack().mean()
10    R_train_filled = R_train_norm.apply(lambda row:
11        row.fillna(row.mean() if not np.isnan(row.mean()) else
12        global_mean), axis=1)
11    R_test_filled = R_test_norm.apply(lambda row:
12        row.fillna(row.mean() if not np.isnan(row.mean()) else
13        global_mean), axis=1)

```

```
return R_train_filled, R_test_filled
```

Listing 4.4: Metodo normaliza del DataManager

4.2.3 Normalizzazione

Il sistema implementa quattro strategie di normalizzazione nel modulo `normalizer.py`:

1. **Simple Centering**: sottrae la media delle valutazioni di ogni utente da tutte le sue valutazioni, centrando dunque i dati attorno allo zero.

$$r'_{ui} = r_{ui} - \bar{r}_u$$

dove \bar{r}_u è la media delle valutazioni dell'utente u .

```
1 def simple_centering(R):
2     user_means = R.mean(axis=1)
3     return R.subtract(user_means, axis=0).astype(float)
```

Listing 4.5: Normalizzazione: Simple Centering

2. **Z-score per utente**: calcola media e deviazione standard delle valutazioni di ogni utente. Ad ogni valutazione si sottrae la media come in simple centering, ma si divide anche per la deviazione standard, normalizzando non solo la scala ma anche la variabilità delle valutazioni.

$$r'_{ui} = \frac{r_{ui} - \bar{r}_u}{\sigma_u}$$

dove \bar{r}_u è la media e σ_u la deviazione standard delle valutazioni dell'utente u .

```
1 def zscore_per_user(R):
2     R_norm = R.copy()
3     for user in R_norm.index:
4         user_ratings = R_norm.loc[user].dropna()
5         if len(user_ratings) > 1 and user_ratings.std() > 0:
6             mean_val = user_ratings.mean()
7             std_val = user_ratings.std()
8             R_norm.loc[user] = (R_norm.loc[user] - mean_val) /
9                 std_val
10    return R_norm.fillna(0).astype(float)
```

Listing 4.6: Normalizzazione: Z-Score

3. **Min-Max per utente**: trovando il valore minimo e massimo delle valutazioni di ogni utente e poi applicando la formula $(\text{valore} - \min) / (\max - \min)$, scala le valutazioni di ogni utente nell'intervallo $[0,1]$.

$$r'_{ui} = \frac{r_{ui} - \min_u}{\max_u - \min_u}$$

dove \min_u e \max_u sono rispettivamente il valore minimo e massimo delle valutazioni dell'utente u .

```

1 def minmax_per_user(R):
2     R_norm = R.copy()
3     for user in R_norm.index:
4         user_ratings = R_norm.loc[user].dropna()
5         if len(user_ratings) > 1:
6             min_val = user_ratings.min()
7             max_val = user_ratings.max()
8             if max_val > min_val:
9                 R_norm.loc[user] = (R_norm.loc[user] - min_val)
10                / (max_val - min_val)
11     return R_norm.astype(float).fillna(0)

```

Listing 4.7: Normalizzazione: Min-Max

4. **Nessuna normalizzazione:** mantenimento dei valori originali.

4.2.4 Algoritmi di Clustering

Il sistema, all'interno della classe `Clusterer` implementa due algoritmi di clustering per la segmentazione degli utenti:

Fuzzy C-Means (FCM)

L'algoritmo Fuzzy C-Means è stato scelto come metodo principale per la sua capacità di gestire l'incertezza nelle assegnazioni degli utenti ai cluster. A differenza del K-Means tradizionale, FCM assegna a ogni utente un grado di membership (valore tra 0 e 1) per ogni cluster, riflettendo la natura sfumata delle preferenze degli utenti. Questo approccio è particolarmente adatto ai sistemi di raccomandazione dove gli utenti possono avere gusti ibridi o appartenere parzialmente a più categorie.

L'obiettivo dell'algoritmo Fuzzy C-Means è minimizzare la seguente funzione di costo:

$$J_m = \sum_{u=1}^N \sum_{c=1}^C u_{cu}^m \cdot \|x_u - v_c\|^2$$

dove:

- $m > 1$ è il parametro di fuzzificazione
- u_{cu} è la membership dell'utente u al cluster c
- x_u è il vettore di caratteristiche dell'utente u
- v_c è il centroide del cluster c
- $\|\cdot\|$ indica la norma euclidea

L'implementazione del Fuzzy C-Means utilizza la libreria `skfuzzy`:

```

1 def fcm_cluster(self, X, n_clusters, m, error, max_iter, seed):
2     cntr, u, _, _, _, _ = fuzz.cmeans(
3         data=X.T,                # Trasposizione per skfuzzy
4         c=n_clusters,            # Numero di cluster

```

```

5         m=m,                # Parametro di fuzziness
6         error=error,        # Criterio di convergenza
7         maxiter=max_iter,    # Iterazioni massime
8         seed=seed            # Seed per riproducibilit 
9     )
10     return cntr, u

```

Listing 4.8: Implementazione FCM

K-Means

L'algoritmo K-Means   stato incluso come baseline per confronto, implementando clustering hard dove ogni utente appartiene esclusivamente a un singolo cluster. Questo approccio tradizionale serve come punto di riferimento per valutare i benefici dell'approccio fuzzy, permettendo di quantificare l'impatto della gestione dell'incertezza nelle assegnazioni dei cluster.

La funzione obiettivo del K-Means da minimizzare  :

$$J = \sum_{u=1}^N \|x_u - v_{c(u)}\|^2$$

dove $c(u)$   il cluster assegnato all'utente u .

Di seguito viene fornita l'implementazione del K-Means standard:

```

1 def kmeans_cluster(self, X, n_clusters, seed):
2     kmeans = sklearn.cluster.KMeans(n_clusters=n_clusters,
3                                     random_state=seed)
4     labels = kmeans.fit_predict(X)
5     cntr = kmeans.cluster_centers_
6
7     u = np.zeros((n_clusters, X.shape[0]))
8     u[labels, np.arange(X.shape[0])] = 1
9     return cntr, u

```

Listing 4.9: Implementazione K-Means

Predizione Rating

La predizione dei rating   necessaria nei sistemi di raccomandazione fuzzy in quanto, dopo aver ottenuto i cluster degli utenti attraverso gli algoritmi di clustering, il sistema utilizza i centroidi dei cluster e le membership degli utenti per generare le predizioni dei rating mancanti.

Il processo di predizione segue questi passaggi:

1. **Calcolo dei Centroidi:** I centroidi rappresentano il profilo medio di preferenze per ogni cluster
2. **Utilizzo delle Membership:** Le membership fuzzy determinano quanto ogni utente contribuisce a ciascun cluster

3. **Predizione Pesata:** Le predizioni finali sono calcolate come media pesata dei centroidi, dove i pesi sono le membership

La predizione di un rating avviene combinando le appartenenze fuzzy dell'utente ai cluster e i centroidi dei cluster relativi agli item. La formula generale è:

$$\hat{r}_{ui} = \sum_{c=1}^C u_{cu} \cdot v_{ci}$$

dove:

- \hat{r}_{ui} è il rating predetto per l'utente u sull'item i
- u_{cu} è il grado di appartenenza dell'utente u al cluster c
- v_{ci} è il valore associato all'item i nel centroide del cluster c
- C è il numero totale di cluster

```
1 def predict(self, cntr, membership):
2     n_clusters, n_users = membership.shape
3     n_items = cntr.shape[1]
4     pred = np.zeros((n_users, n_items))
5
6     for c in range(n_clusters):
7         weights = membership[c, :]
8         pred += np.outer(weights, cntr[c, :])
9
10    return pred
```

Listing 4.10: Algoritmo di predizione della classe Clusterer

Infine, la classe `Clusterer` calcola la membership di utenti mai visti usando i centroidi addestrati, tramite il metodo `predict_test`.

```
1 def predict_test(self, R_test_scaled, cntr, m, error, max_iter):
2     u_test, _, _, _, _ = fuzz.cmeans_predict(
3         R_test_scaled.T, cntr, m, error=error, maxiter=max_iter)
4     return u_test
```

Listing 4.11: Calcolo della Membership in fase di test

4.2.5 Strategie di Defuzzificazione

Le strategie di defuzzificazione convertono le membership fuzzy, ovvero valori continui tra 0 e 1, in decisioni discrete o valori crisp, cioè in un valore preciso e definito. Questo viene effettuato per poter facilitare l'interpretazione dei dati e permettere di prendere decisioni finali, come determinare a quale cluster appartiene ciascun utente.

Il sistema implementa due strategie principali di defuzzificazione:

1. **Metodo del Massimo:** Assegna ogni utente al cluster con membership massima, approccio più conservativo che privilegia la certezza.

```

1 def defuzzify_maximum(membership):
2     return np.argmax(membership, axis=0)

```

Listing 4.12: Defuzzificazione per massimo

2. **Center of Gravity (COG):** Calcola un indice continuo come media pesata delle membership, mantenendo la natura fuzzy.

```

1 def defuzzify_cog(membership):
2     cluster_indices =
3         np.arange(membership.shape[0]).reshape(-1, 1)
4     cog = np.sum(membership * cluster_indices, axis=0) /
5         np.sum(membership, axis=0)
6     return cog

```

Listing 4.13: Defuzzificazione COG

4.2.6 Selezione dei Vicini

La selezione dei vicini è una strategia opzionale che permette di filtrare gli utenti candidati per il calcolo delle raccomandazioni basandosi su criteri di similarità.

Il sistema supporta due modalità di selezione:

1. **Nessuna Selezione:** Considera tutti gli utenti nel cluster, approccio più inclusivo ma potenzialmente rumoroso.
2. **Correlazione di Pearson:** Filtra gli utenti basandosi sulla correlazione delle loro valutazioni, approccio più selettivo che privilegia la similarità.

```

1 def select_pearson_neighbors(user_vector, candidate_matrix,
2     threshold=0.5):
3     indices = []
4     pearson_values = []
5
6     for idx, candidate in enumerate(candidate_matrix):
7         mask = ~np.isnan(user_vector) & ~np.isnan(candidate)
8         if np.sum(mask) < 2:
9             continue
10
11         r_tuple = pearsonr(user_vector[mask], candidate[mask])
12         r = r_tuple[0]
13
14         if not isinstance(r, (float, int)) or np.isnan(r):
15             continue
16
17         if r > threshold:
18             indices.append(idx)
19             pearson_values.append(r)
20
21     return np.array(indices), np.array(pearson_values)

```

Listing 4.14: Selezione vicini Pearson

4.3 Valutazione e Visualizzazione delle Performance

Una volta implementata la pipeline esecutiva del programma, si è resa necessaria una fase di valutazione delle performance ottenibili.

Le metriche utilizzate a tal fine sono:

- **RMSE (Root Mean Square Error)**: Penalizza maggiormente gli errori grandi, utile per identificare predizioni molto inaccurate
- **MAE (Mean Absolute Error)**: Fornisce una misura più robusta degli errori, meno sensibile agli outlier

La valutazione viene eseguita sia sui dati di test che su quelli di training per analizzare la capacità del modello di generalizzare su dati non visti e per monitorare eventuali fenomeni di overfitting.

```
1 def evaluate(self, y_true, y_pred):
2     y_true = np.array(y_true, dtype=np.float64)
3     y_pred = np.array(y_pred, dtype=np.float64)
4
5     mask = ~np.isnan(y_true) & ~np.isnan(y_pred)
6
7     mse = mean_squared_error(y_true[mask], y_pred[mask])
8     mae = mean_absolute_error(y_true[mask], y_pred[mask])
9
10    return np.sqrt(mse), mae
```

Listing 4.15: Calcolo metriche di valutazione

Il sistema include un modulo di visualizzazione completo (`Plotter.py`) che genera:

1. **Cluster PCA**: Visualizzazione 2D dei cluster utente
2. **Istogrammi di Membership**: Distribuzione dei valori di membership massimi
3. **Heatmap di Membership**: Matrice di membership per utenti incerti
4. **Confronti tra Normalizzazioni**: Plot riassuntivo con PCA, Max Membership e Data Distribution

Infine, vista la grande mole di plot ottenuti, il modulo `aggregate_plotter.py` genera visualizzazioni comparative come:

- **Barplot Top-N**: Migliori configurazioni per ogni metrica
- **Heatmap**: Performance per cluster e normalizzazione
- **Boxplot**: Distribuzione delle metriche per metodo di clustering
- **Summary**: File di testo con i top-n migliori risultati per ogni combinazione di metrica (RMSE, MAE) e fase (train, test)

Confronto con Letteratura

L'analisi svolta in [KOOHI2016134], su cui tale lavoro si basa, propone un sistema di raccomandazione collaborativo user-based basato su clustering, confrontando l'efficacia di

tre tecniche: K-means, Self-Organizing Map (SOM) e Fuzzy C-means (FCM). L'analisi si concentra sull'accuratezza della raccomandazione in termini di accuracy, precision e recall, utilizzando la metrica di similarità di Pearson e media delle valutazioni come tecniche di predizione due metodi di defuzzificazione: massimo e centro di gravità. Il dataset utilizzato è MovieLens 100k, con valutazione basata su una procedura di *5-fold cross-validation*.

Il sistema proposto in questo lavoro si differenzia sotto alcuni aspetti. Osservando che SOM offre prestazioni inferiori rispetto a FCM, è stato deciso di ometterlo e limitarsi a un algoritmo per hard clustering e uno per fuzzy clustering. Stesso discorso vale per la tecnica di predizione delle valutazioni mediante media. Infine, non è stato implementato la valutazione tramite fold. In aggiunta, invece, vengono implementati 3 metodi di normalizzazione, gestione dei vincoli minimi su utenti e item, controllo del parametro di fuzzificazione m , introduzione di rumore artificiale e definizione personalizzata della suddivisione train/test.

La Tabella ?? riassume le principali differenze tra i due approcci.

Tabella 4.1: Confronto dei parametri sperimentali tra questo lavoro e Koohi & Kiani (2016)

Parametro / Funzionalità	Koohi & Kiani (2016)	Questo lavoro
Algoritmi di clustering	K-means, SOM, FCM	K-means, FCM
Numero di cluster	Sì (da 3 a 15)	Sì (valore configurabile)
Parametro m	No	Sì (configurabile)
Metodi di predizione	Average, Pearson	Pearson
Metriche di valutazione	Accuracy, Precision, Recall	RMSE, MAE, Precision, Recall, Accuracy
Metodi di defuzzificazione	Max, Center of Gravity	Max, Center of Gravity
Tecniche di normalizzazione	No	3 (centering, min-max, z-score)
Filtro su utenti/item minimi	No	Sì
Introduzione di rumore nei dati	No	Sì (valore configurabile)
Train/Test	5-fold	Dimensione split configurabile
Selezione della similarità	Pearson, Media	Pearson

Capitolo 5

Risultati Sperimentali

Il sistema di raccomandazione fuzzy è stato valutato attraverso tre esperimenti progressivi, ciascuno progettato per esplorare aspetti specifici dell'algoritmo e ottimizzare le performance. La progressione degli esperimenti segue un approccio iterativo: dall'esplorazione iniziale dello spazio dei parametri, all'ottimizzazione focalizzata del Fuzzy C-Means, fino al confronto approfondito con il clustering hard tradizionale.

I risultati ottenuti, oltre a essere riportati nel seguente capitolo, sono osservabili nelle cartelle di output del repository. Nel presente capitolo vengono riportati principalmente i plot di comparison (boxplot, heatmap, violinplot, lineplot nclusters) per ciascuna metrica e configurazione e alcuni plot utili per le run migliori, in quanto più sintetici e utili per il confronto tra algoritmi e parametri. Per un'analisi più approfondita e dettagliata, sono disponibili ulteriori visualizzazioni (ad esempio, fuzzy clusters, membership heatmap, membership histogram, ecc.) all'interno delle cartelle di output del repository.

Gli esperimenti sono stati condotti sul dataset MovieLens 100k, filtrato per includere solo utenti e item con almeno 150 rating, risultando in un dataset di circa 300 utenti e 900 film. Questa scelta garantisce una densità di rating sufficiente per valutare l'efficacia del clustering, pur mantenendo la complessità necessaria per testare le capacità del sistema fuzzy.

5.1 Metriche

5.1.1 Metriche di Errore Predittivo

Le metriche RMSE e MAE valutano l'accuratezza predittiva del modello nel prevedere i rating esatti degli utenti. Queste metriche misurano quanto il modello sia preciso nel predire il rating che un utente darebbe a un film specifico.

RMSE (Root Mean Square Error)

La metrica RMSE misura l'errore quadratico medio tra i rating predetti e quelli reali. Un valore più basso indica una migliore capacità predittiva. Il RMSE penalizza maggiormente gli errori grandi, rendendolo utile per identificare predizioni molto inaccurate.

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,i) \in T} (r_{ui} - \hat{r}_{ui})^2}$$

dove:

- T è l'insieme delle coppie utente-item nel test set

- r_{ui} è il rating reale dell'utente u sull'item i
- \hat{r}_{ui} è il rating predetto

MAE (Mean Absolute Error)

La metrica MAE misura l'errore assoluto medio tra i rating predetti e quelli reali. Il MAE è meno sensibile agli outlier rispetto al RMSE e fornisce una misura più robusta degli errori.

$$MAE = \frac{1}{|T|} \sum_{(u,i) \in T} |r_{ui} - \hat{r}_{ui}|$$

dove le variabili sono definite come per il RMSE.

5.1.2 Metriche di Qualità delle Raccomandazioni

Le metriche Precision, Recall, Accuracy e F1-score valutano la qualità effettiva delle raccomandazioni generate dal sistema. A differenza delle metriche di errore che misurano la precisione predittiva dei rating, queste metriche valutano quanto le raccomandazioni siano utili e pertinenti per l'utente nella pratica.

Precision

La metrica Precision misura la frazione di item raccomandati nei top-N che sono effettivamente piaciuti all'utente. Un valore più alto indica che il sistema raccomanda principalmente item che l'utente apprezza.

La metrica Precision si calcola come:

$$Precision = \frac{|\text{Relevant} \cap \text{Recommended}|}{|\text{Recommended}|}$$

dove:

- **Recommended** è l'insieme degli item raccomandati
- **Relevant** è l'insieme degli item che l'utente ha valutato positivamente (rating \geq soglia)

Recall

La metrica Recall misura la frazione di item apprezzati dall'utente che sono stati effettivamente raccomandati nei top-N. Un valore più alto indica che il sistema riesce a catturare la maggior parte delle preferenze dell'utente.

La metrica Recall si calcola come:

$$Recall = \frac{|\text{Relevant} \cap \text{Recommended}|}{|\text{Relevant}|}$$

dove:

- **Relevant** è l'insieme degli item che l'utente ha effettivamente apprezzato
- **Recommended** è l'insieme degli item suggeriti dal sistema

Accuracy

La metrica Accuracy misura la frazione di raccomandazioni corrette nei top-N. Questa metrica fornisce una misura diretta dell'accuratezza delle raccomandazioni generate.

La metrica Accuracy si calcola come:

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

dove:

- *TP* (True Positive): item rilevanti correttamente raccomandati
- *TN* (True Negative): item irrilevanti correttamente non raccomandati
- *FP* (False Positive): item irrilevanti raccomandati
- *FN* (False Negative): item rilevanti non raccomandati

F1-Score

La metrica F1-Score calcola la media armonica tra Precision e Recall, fornendo una misura bilanciata che considera sia la qualità che la copertura delle raccomandazioni.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Fornisce un compromesso tra qualità (Precision) e copertura (Recall) delle raccomandazioni.

Metrica	Formula
Precision	$\frac{TP}{TP + FP}$
Recall	$\frac{TP}{TP + FN}$
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$
F1-Score	$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$

Tabella 5.1: Metriche di valutazione delle raccomandazioni

5.2 Run Sample

La run "sample" rappresenta un primo approccio alla pipeline di clustering realizzata. Lo scopo principale di questa run è stato quello di verificare che entrambi gli algoritmi di clustering fossero in grado di produrre risultati di qualità. Di conseguenza, è stata mantenuta la normalizzazione utilizzata, così come il metodo di defuzzificazione e di

selezione dei vicini. Sono stati invece 3 valori di cluster, sapendo che i risultati dal paper utilizzato mostravano come 3 fosse il numero che fornisse i risultati migliori. Sono stati poi testati due valori di m: 1.5 e 2.0.

Listing 5.1: Configurazione aggiornata Run Sample

```

1 {
2   "dataset_name": "ml-100k",
3   "normalizations": ["simple_centering"],
4   "min_user_ratings": 100,
5   "min_item_ratings": 100,
6   "cluster_values": [2, 3, 4],
7   "m_values": [1.5, 2.0],
8   "noise_std": 0.05,
9   "test_size": 0.2,
10  "random_state": 42,
11  "seed": 31,
12  "max_iter": 3000,
13  "error": 1e-06,
14  "output_dir": "output",
15  "show_plots": false,
16  "images_subdir": "images",
17  "results_subdir": "results",
18  "run_timestamp_format": "run_sample",
19  "clustering_methods": ["kmeans", "fcm"],
20  "defuzzification_methods": ["maximum"],
21  "neighbor_selection_methods": ["pearson"],
22  "summary_only": false,
23  "top_n": 5,
24  "top_n_evaluation": {
25    "n_recommendations": 10,
26    "rating_threshold": 4.0
27  }
28 }
```

5.2.1 Risultati in Train

In questa sezione vengono presentati i risultati ottenuti in fase di addestramento per tutte le metriche considerate. Per ciascuna metrica, si riportano le migliori configurazioni e i relativi plot.

Tabella 5.2: Top 5 Configurazioni per Train RMSE - Run Sample

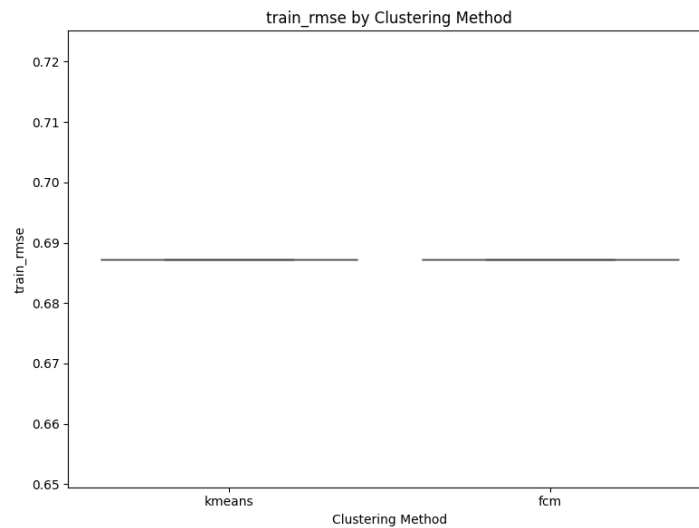
Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Train RMSE
1	simple_centering	2	1.5	fcm	maximum	pearson	0.687303
2	simple_centering	3	1.5	fcm	maximum	pearson	0.687303
3	simple_centering	4	2.0	kmeans	maximum	pearson	0.687303
4	simple_centering	2	2.0	kmeans	maximum	pearson	0.687303
5	simple_centering	3	2.0	kmeans	maximum	pearson	0.687303

Tabella 5.3: Top 5 Configurazioni per Train MAE - Run Sample

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Train MAE
1	simple_centering	2	1.5	fcm	maximum	pearson	0.506783
2	simple_centering	3	1.5	fcm	maximum	pearson	0.506783
3	simple_centering	4	2.0	kmeans	maximum	pearson	0.506783
4	simple_centering	2	2.0	kmeans	maximum	pearson	0.506783
5	simple_centering	3	2.0	kmeans	maximum	pearson	0.506783

RMSE e MAE Possiamo osservare come i valori ottenuti per RMSE e MAE durante la fase di train siano molto bassi, con valori vicini a 0.5. I valori risultano uguali nonostante variano alcuni parametri in quanto, probabilmente, i dati sono molto compatti e i test effettuati non riescono ancora ad identificare eventuali pattern presenti.

I boxplot e le heatmap mostrano che le differenze tra FCM e K-Means sono contenute, con una leggera prevalenza di K-Means in alcune configurazioni.

**Figura 5.1:** Boxplot Train RMSE: confronto tra FCM e K-Means

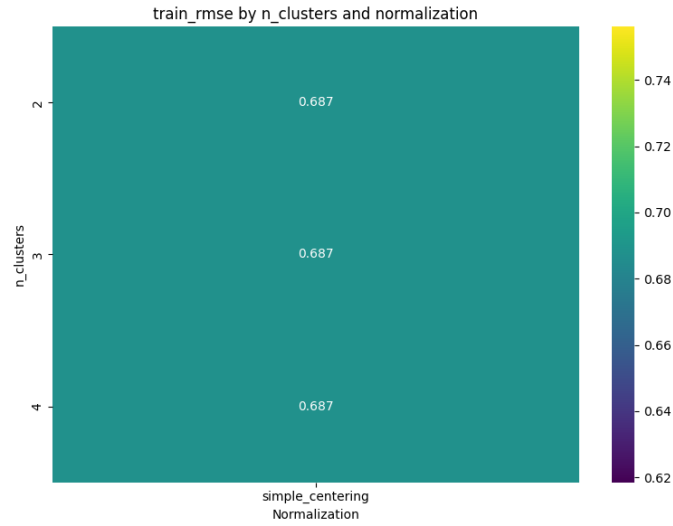


Figura 5.2: Heatmap Train RMSE: distribuzione dei risultati rispetto a cluster e normalizzazione.

Precision, Recall, Accuracy, F1-score Le metriche di qualità delle raccomandazioni valutano la capacità del sistema di suggerire item rilevanti agli utenti. Le migliori configurazioni per ciascuna metrica sono riportate di seguito.

Tabella 5.4: Top 5 Configurazioni per Train Precision - Run Sample

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Precision
...

Analoghe tabelle sono disponibili per Recall, Accuracy e F1-score.

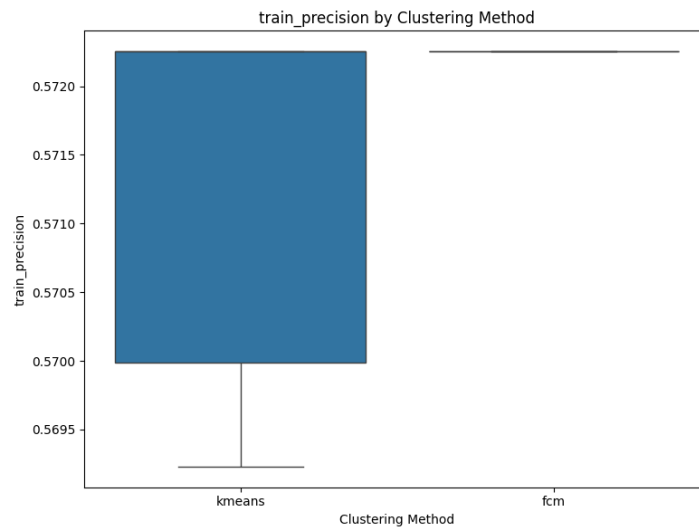


Figura 5.3: Boxplot Train Precision: confronto tra FCM e K-Means

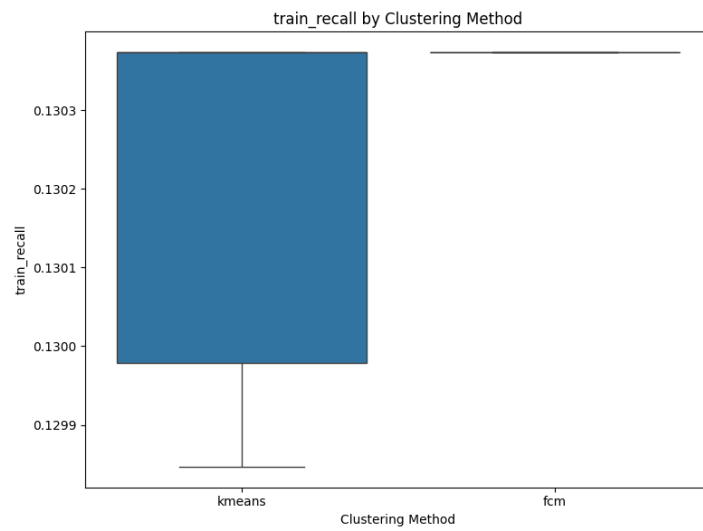


Figura 5.4: Boxplot Train Recall: confronto tra FCM e K-Means

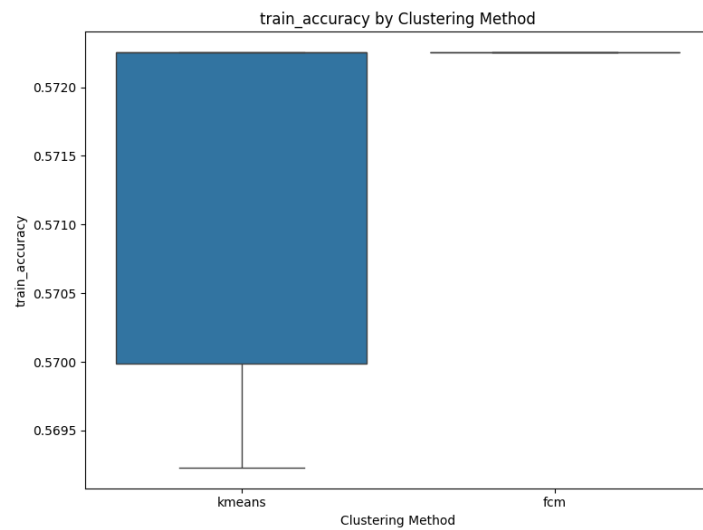


Figura 5.5: Boxplot Train Accuracy: confronto tra FCM e K-Means

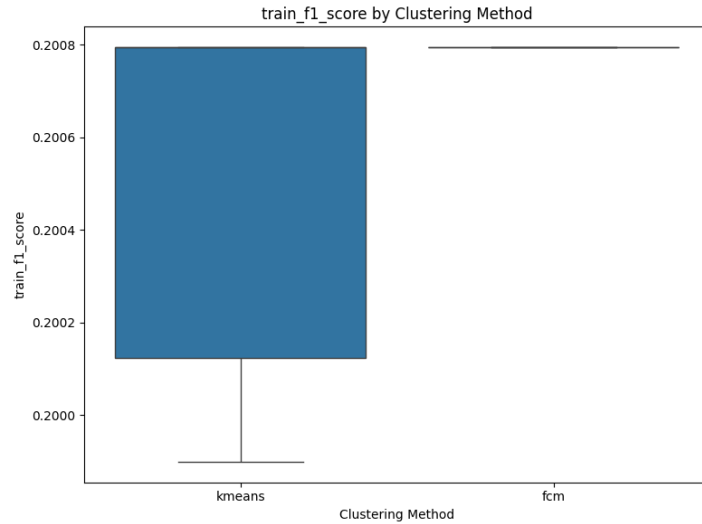


Figura 5.6: Boxplot Train F1-score: confronto tra FCM e K-Means

L'analisi dei risultati mostra che, in fase di train, le metriche di qualità delle raccomandazioni sono generalmente elevate per entrambe le famiglie di algoritmi, con leggere variazioni a seconda della configurazione. K-Means tende a ottenere valori di precision e accuracy leggermente superiori, mentre FCM mostra una maggiore stabilità su recall e F1-score.

5.2.2 Risultati in Test

In questa sezione vengono presentati i risultati ottenuti in fase di test, seguendo la stessa struttura della sezione precedente.

RMSE e MAE Le migliori configurazioni per RMSE e MAE in test sono riportate nelle tabelle seguenti.

Tabella 5.5: Top 5 Configurazioni per Test RMSE - Run Sample

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Test RMSE
...

Tabella 5.6: Top 5 Configurazioni per Test MAE - Run Sample

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Test MAE
...

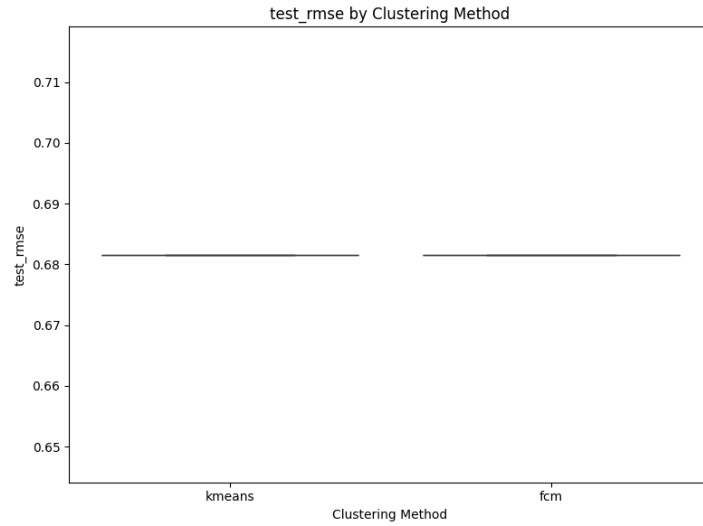


Figura 5.7: Boxplot Test RMSE: confronto tra FCM e K-Means



Figura 5.8: Heatmap Test RMSE: distribuzione dei risultati rispetto a cluster e normalizzazione.

Precision, Recall, Accuracy, F1-score Le metriche di qualità delle raccomandazioni in test sono riportate nelle tabelle seguenti.

Tabella 5.7: Top 5 Configurazioni per Test Precision - Run Sample

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Precision
...

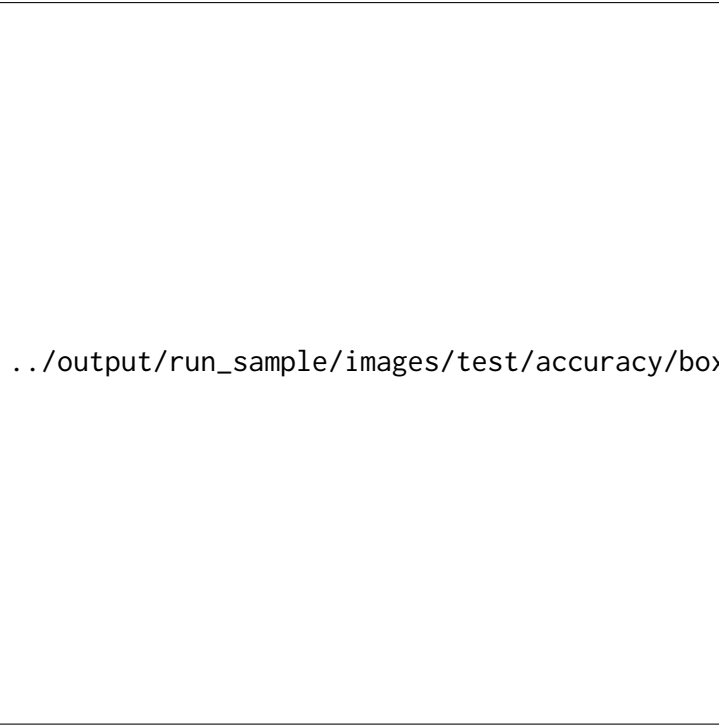
Analoghe tabelle sono disponibili per Recall, Accuracy e F1-score.



Figura 5.9: Boxplot Test Precision: confronto tra FCM e K-Means




Figura 5.10: Boxplot Test Recall: confronto tra FCM e K-Means



../output/run_sample/images/test/accuracy/boxplot_test_accuracy.png

Figura 5.11: Boxplot Test Accuracy: confronto tra FCM e K-Means



../output/run_sample/images/test/f1_score/boxplot_test_f1_score.png

Figura 5.12: Boxplot Test F1-score: confronto tra FCM e K-Means

In fase di test, le metriche di qualità delle raccomandazioni risultano generalmente inferiori rispetto al train, come atteso, ma confermano la tendenza osservata: K-Means mostra valori di precision e accuracy leggermente superiori, mentre FCM mantiene una maggiore stabilità su recall e F1-score.

5.2.3 Conclusioni Run Sample

L'analisi della run sample aggiornata mostra che, oltre alle metriche di errore predittivo, le metriche di qualità delle raccomandazioni forniscono una visione più completa delle prestazioni dei modelli. K-Means tende a ottenere valori di precision e accuracy leggermente migliori, mentre FCM si distingue per una maggiore stabilità su recall e F1-score. Tuttavia, le differenze tra i due algoritmi rimangono contenute, suggerendo che la scelta tra FCM e K-Means debba essere guidata anche da considerazioni di interpretabilità e di struttura dei dati.

5.3 Run FCM

La run "FCM" è stata progettata per esplorare in modo sistematico il comportamento dell'algoritmo Fuzzy C-Means, variando in modo più fine il numero di cluster e il grado di fuzziness, e mantenendo costanti le altre impostazioni. L'obiettivo era ottimizzare la qualità delle soluzioni fuzzy per poter valutare se l'approccio fuzzy a tale dominio fosse efficace.

Listing 5.2: Configurazione Run FCM Deep Dive

```
1 {
2   "dataset_name": "ml-100k",
3   "normalizations": ["no_normalization", "simple_centering", "
4     minmax_per_user", "zscore_per_user"],
5   "min_user_ratings": 150,
6   "min_item_ratings": 150,
7   "cluster_values": [6, 8, 10, 12],
8   "m_values": [1.2, 1.5, 1.8, 2.0, 2.2, 2.5],
9   "noise_std": 0.05,
10  "test_size": 0.2,
11  "random_state": 42,
12  "seed": 31,
13  "max_iter": 3000,
14  "error": 1e-06,
15  "output_dir": "output",
16  "show_plots": false,
17  "images_subdir": "images",
18  "results_subdir": "results",
19  "run_timestamp_format": "run_%Y_%m_%d_%H_%M_%S",
20  "clustering_methods": ["fcm"],
21  "defuzzification_methods": ["maximum", "cog"],
22  "neighbor_selection_methods": ["none", "pearson"],
23  "summary_only": false,
24  "top_n": 5
25 }
```

5.3.1 Train RMSE

Tabella 5.8: Top 5 Configurazioni per Train RMSE - Run FCM

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Train RMSE	Avg Max Membership	Avg Entropy	Elapsed Sec
1	simple_centering	10	1.5	fcm	maximum	pearson	0.686397	0.100000	2.302585	15.221505
2	simple_centering	12	2.5	fcm	cog	pearson	0.686788	0.083333	2.484907	13.964393
3	simple_centering	12	2.5	fcm	maximum	pearson	0.686992	0.083333	2.484907	11.301697
4	simple_centering	8	2.2	fcm	cog	pearson	0.687305	0.125000	2.079442	15.716032
5	simple_centering	8	1.8	fcm	cog	pearson	0.687305	0.125000	2.079442	15.827245

L'analisi approfondita di FCM mostra che la configurazione ottimale (simple_centering, 10 cluster, m=1.5) raggiunge un Train RMSE di 0.6864, inferiore rispetto alla run sample di 0.02 circa. Le prime posizioni sono tutte caratterizzate da valori di avg max membership molto bassi (tra 0.08 e 0.16) e avg entropy elevata (fino a 2.48), a testimonianza di una forte sovrapposizione tra cluster.

In figura ?? si può osservare l'istogramma delle membership per la configurazione migliore. Si osserva come i valori di appartenenza siano distribuiti in maniera asimmetrica: i valori di membership massimi sono concentrati principalmente tra 0 e 2, mentre valori più alti sono molto rari, suggerendo che pochi elementi hanno un'appartenenza molto alta a un cluster.



Figura 5.13: Boxplot Train RMSE: confronto tra configurazioni FCM.

5.3.2 Test RMSE

Tabella 5.9: Top 5 Configurazioni per Test RMSE - Run FCM

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Test RMSE	Avg Max Membership	Avg Entropy	Elapsed Sec
1	simple_centering	12	2.5	fcm	maximum	pearson	0.577756	0.083333	2.484907	11.301697
2	zscore_per_user	6	1.8	fcm	maximum	pearson	0.582730	0.166667	1.791759	13.483854
3	zscore_per_user	6	2.0	fcm	maximum	pearson	0.584205	0.166667	1.791759	12.751047
4	zscore_per_user	6	2.2	fcm	maximum	pearson	0.584205	0.166667	1.791759	12.773057
5	simple_centering	12	2.5	fcm	cog	pearson	0.584598	0.083333	2.484907	13.964393

Anche per il Test RMSE, la configurazione migliore (simple_centering, 12 cluster, m=2.5) ottiene un errore di 0.5778, leggermente inferiore rispetto alla run sample. A differenza di quanto accadeva nella run sample, simple_centering torna a essere la migliore normalizzazione insieme a minmax_per_user, mentre zscore_per_user leggermente inferiore.



Figura 5.14: Heatmap Test RMSE: distribuzione dei risultati rispetto a cluster e normalizzazione.

5.3.3 Train e Test MAE

I risultati per la metrica MAE ricalcano quelli per la metrica RMSE, non offrendo spunti interessanti da analizzare.

5.3.4 Conclusioni Run FCM

I risultati della run FCM evidenziano un leggero miglioramento rispetto alla fase sample in termini di errore. Tuttavia, è necessario confermare l'ipotesi suscitata durante la run

di sample: FCM, sebbene risultati leggermente migliori rispetto a K-Means, non sembra essere in grado di distinguere tra i vari cluster. La prossima run, che si occuperà di K-Means, vuole considerare se, eseguendo un'analisi più approfondita, si possa ottenere un miglioramento rispetto a FCM.

5.4 Run K-Means

La run "K-Means" è stata dedicata all'analisi sistematica dell'algoritmo di hard clustering, testando un ampio range di valori di cluster. I parametri di fuzziness e defuzzification, sebbene presenti nel config, non sono realmente utilizzati e quindi fungono solo da placeholder.

Listing 5.3: Configurazione Run K-Means

```

1 {
2   "dataset_name": "ml-100k",
3   "normalizations": ["no_normalization", "simple_centering", "
4     minmax_per_user", "minmax_per_item"],
5   "min_user_ratings": 150,
6   "min_item_ratings": 150,
7   "cluster_values": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
8     16],
9   "m_values": [2.0],
10  "noise_std": 0.05,
11  "test_size": 0.2,
12  "random_state": 42,
13  "seed": 31,
14  "max_iter": 3000,
15  "error": 1e-06,
16  "output_dir": "output",
17  "show_plots": false,
18  "images_subdir": "images",
19  "results_subdir": "results",
20  "run_timestamp_format": "run_%Y_%m_%d_%H_%M_%S",
21  "clustering_methods": ["kmeans"],
22  "defuzzification_methods": ["maximum"],
23  "neighbor_selection_methods": ["none", "pearson"],
24  "summary_only": false,
25  "top_n": 5
26 }
```

5.4.1 Train RMSE

Tabella 5.10: Top 5 Configurazioni per Train RMSE - Run K-means

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Train RMSE	Elapsed Sec
1	simple_centering	6	2.0	kmeans	maximum	pearson	0.668238	8.051133
2	simple_centering	3	2.0	kmeans	maximum	pearson	0.689817	10.802989
3	simple_centering	8	2.0	kmeans	maximum	pearson	0.696528	8.678104
4	simple_centering	5	2.0	kmeans	maximum	pearson	0.699205	8.521540
5	zscore_per_user	13	2.0	kmeans	maximum	pearson	0.708584	2.345174

Per quanto riguarda K-Means, la configurazione migliore (simple_centering, 6 cluster) ottiene un Train RMSE di 0.6682, migliore rispetto a FCM di 0.02 circa. La tecnica di normalizzazione migliore sembra essere simple_centering. I tempi di esecuzione, non richiedendo fuzzificazione e defuzzificazione, sono molto più rapidi rispetto a FCM. Un'altra osservazione interessante riguarda il numero di cluster: nelle prime 5 posizioni si ottengono risultati simili passando da 3 a 13 cluster, suggerendo come i valori siano troppo omogenei e quindi poco separabili in maniera significativa.

In figura ?? si può confermare quanto affermato in precedenza: i cluster sono molto omogenei e quindi poco separabili in maniera significativa. Si osservano elementi appartenenti a cluster diversi, ma con posizioni nello spazio molto vicine.



Figura 5.15: Clusters per la configurazione migliore in Train RMSE - Run K-Means.

In figura ?? si può osservare come il metodo di normalizzazione migliore sia simple_centering, seguito da zscore_per_user e minmax_per_user. Si osserva anche come i metodi di normalizzazione offrono la loro migliore prestazione con numeri di cluster diversi: simple_centering è migliore con pochi cluster, mentre zscore_per_user e minmax_per_user con un numero più elevato di cluster.

../output/run_kmeans/images/comparison/train_rmse/heatmap_train_rmse.png

Figura 5.16: Heatmap Train RMSE: distribuzione dei risultati rispetto a cluster e normalizzazione.

5.4.2 Test RMSE

Tabella 5.11: Top 5 Configurazioni per Test RMSE - Run K-means

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Test RMSE	Elapsed Sec
1	simple_centering	16	2.0	kmeans	maximum	pearson	0.585079	6.995507
2	simple_centering	14	2.0	kmeans	maximum	pearson	0.590401	6.780639
3	simple_centering	15	2.0	kmeans	maximum	pearson	0.591287	6.479710
4	zscore_per_user	2	2.0	kmeans	maximum	pearson	0.591508	12.853024
5	simple_centering	10	2.0	kmeans	maximum	pearson	0.593909	9.259707

Nel Test RMSE, la configurazione ottimale (simple_centering, 16 cluster) raggiunge 0.5851, mostrando quindi un leggero peggioramento rispetto a FCM, dove la migliore run era inferiore di 0.01 circa. Anche come per la fase di train, i risultati sono molto simili passando da 2 a 16 cluster.

Nelle figure ?? e ?? si può osservare come 16 cluster stiano creando overfitting e non ci sia alcuna rappresentazione significativa di cosa un cluster rappresenti.

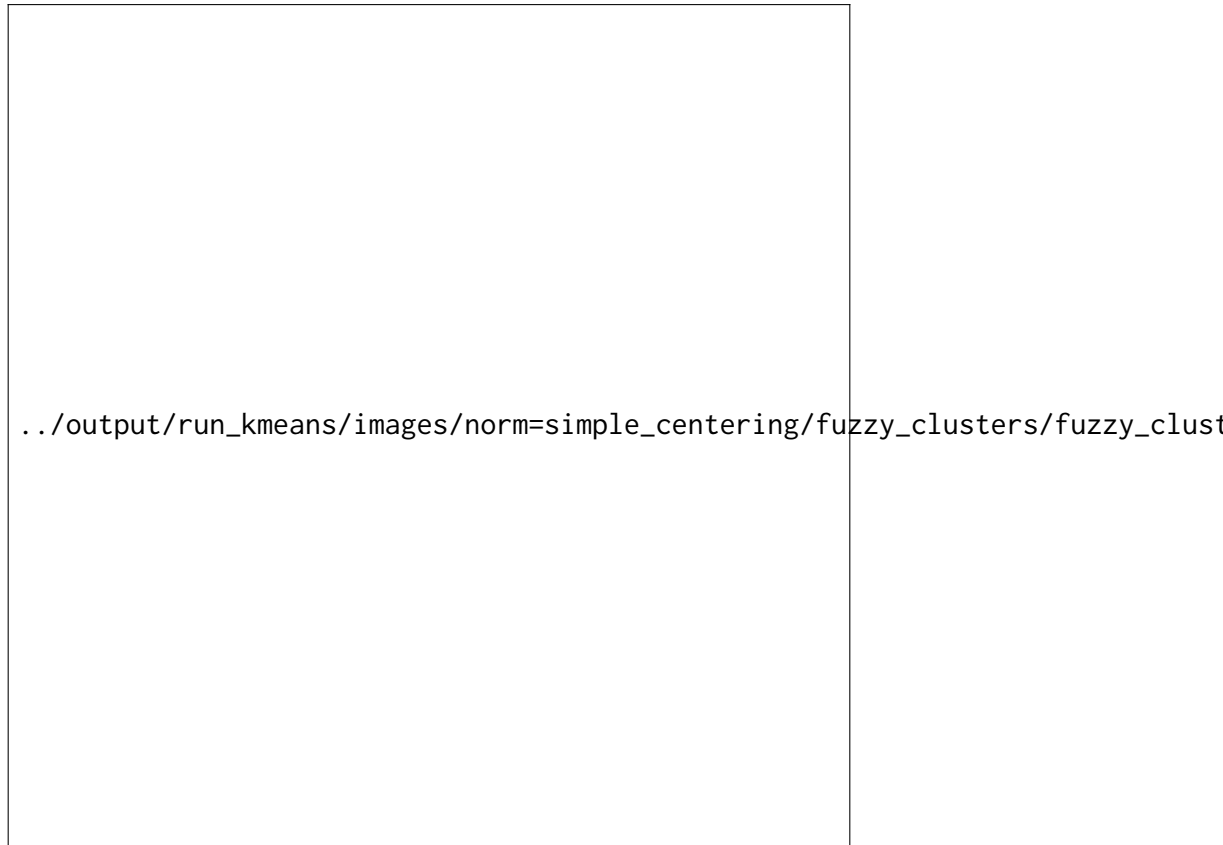


Figura 5.17: Clusters per la configurazione migliore in Test RMSE - Run K-Means.



Figura 5.18: Clusters per la configurazione migliore in Test RMSE - Run K-Means.

5.4.3 Train MAE

Tabella 5.12: Top 5 Configurazioni per Train MAE - Run K-means

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Train MAE	Elapsed Sec
1	simple_centering	6	2.0	kmeans	maximum	pearson	0.520091	8.051133
2	simple_centering	5	2.0	kmeans	maximum	pearson	0.542668	8.521540
3	simple_centering	8	2.0	kmeans	maximum	pearson	0.543493	8.678104
4	simple_centering	3	2.0	kmeans	maximum	pearson	0.546920	10.802989
5	simple_centering	2	2.0	kmeans	maximum	pearson	0.560178	19.746683

Per la metrica MAE, si ottiene un leggero miglioramento rispetto a FCM, dove la migliore run era superiore di 0.02 circa. Tutte le prime posizioni sono occupate dalla tecnica di normalizzazione `simple_centering` e da numeri di cluster molto bassi. In figura ?? si può osservare come il numero di cluster stia effettivamente peggiorando i risultati, sebbene non in maniera significativa.

5.4.4 Test MAE

Tabella 5.13: Top 5 Configurazioni per Test MAE - Run K-means

Rank	Normalization	Clusters	m	Method	Defuzz	Neighbor	Test MAE	Elapsed Sec
1	simple_centering	16	2.0	kmeans	maximum	pearson	0.461418	6.995507
2	simple_centering	14	2.0	kmeans	maximum	pearson	0.463065	6.780639
3	simple_centering	13	2.0	kmeans	maximum	pearson	0.463293	6.880829
4	simple_centering	15	2.0	kmeans	maximum	pearson	0.464252	6.479710
5	simple_centering	10	2.0	kmeans	maximum	pearson	0.465046	9.259707

Infine, per il Test MAE, la configurazione migliore (`simple_centering`, 16 cluster) ottiene 0.4614, confermando un trend: `kmeans` ottiene 0.02 di miglioramento su entrambe le metriche in fase di train, ma un peggioramento di 0.01 in fase di test, rispetto a FCM.

5.4.5 Conclusioni Run K-Means

L'analisi esplorativa di K-Means ha mostrato come anche tramite la tecnica di hard clustering si possano ottenere risultati simili a FCM. I tempi di esecuzione sono molto rapidi rispetto a FCM, data la mancanza delle fasi di fuzzification e defuzzificazione. La mancanza di una struttura fuzzy limita la capacità di cogliere sfumature nelle preferenze utente, e dunque annulla il vantaggio che il fuzzy clustering potrebbe offrire rispetto alla controparte hard. cioè la possibilità di appartenenza a più cluster simultaneamente.

Capitolo 6

Conclusioni e Sviluppi Futuri

6.1 Conclusioni

Il lavoro svolto ha permesso di esplorare in profondità l'applicazione del *fuzzy clustering* nei *sistemi di raccomandazione*, con particolare attenzione al confronto tra l'algoritmo *Fuzzy C-Means (FCM)* e il più tradizionale *K-Means*. Attraverso la progettazione e l'implementazione di un framework sperimentale flessibile, è stato possibile testare numerose configurazioni, strategie di normalizzazione e parametri, conducendo esperimenti sistematici sul dataset *MovieLens 100k*.

Dall'analisi dei risultati sperimentali, emerge un quadro chiaro: sebbene FCM risulti migliore in termini di errore in fase di test, il vantaggio rispetto a K-Means si rivela marginale. In particolare, le differenze tra i due algoritmi si attestano su pochi centesimi di punto nelle metriche di valutazione, e spesso K-Means risulta addirittura preferibile in termini di rapidità computazionale, non necessitando delle fasi di fuzzificazione e defuzzificazione.

Un aspetto particolarmente rilevante, emerso dall'analisi delle heatmap delle membership e dei fuzzy clusters, è la difficoltà di FCM nel distinguere cluster realmente significativi all'interno del dataset. I valori medi di appartenenza ai cluster risultano distribuiti in modo uniforme, e l'entropia elevata suggerisce una forte sovrapposizione tra i cluster stessi. Questo fenomeno è probabilmente riconducibile all'*omogeneità del dataset*, che limita la possibilità di apprendere strutture fuzzy realmente informative.

In sintesi, i risultati ottenuti suggeriscono che, almeno nel contesto e con i dati considerati, l'approccio fuzzy non apporta un miglioramento sostanziale rispetto al clustering hard. Tuttavia, il *framework* sviluppato si è dimostrato efficace nell'esplorare lo spazio delle configurazioni e potrà essere *facilmente esteso* a contesti e dataset differenti.

6.2 Sviluppi Futuri

Alla luce delle evidenze sperimentali raccolte, si delineano diversi possibili sviluppi futuri per approfondire e ampliare la ricerca:

- **Applicazione a dataset di dominio differente:** L'omogeneità del MovieLens 100k ha probabilmente limitato la capacità del fuzzy clustering di individuare strutture latenti. Un naturale sviluppo consiste nell'applicare il framework a dataset di natura diversa, ad esempio in ambito e-commerce, musica o news, dove la varietà delle preferenze utente potrebbe favorire la formazione di cluster fuzzy più significativi.

- **Analisi su dataset di dimensioni maggiori:** Un'estensione immediata riguarda l'utilizzo del dataset MovieLens 1M, che, grazie al maggior numero di utenti e item, potrebbe offrire una maggiore eterogeneità e quindi permettere al fuzzy clustering di esprimere appieno il proprio potenziale. Essendo il dataset già compatibile all'interno del sistema, un'analisi esplorativa (EDA) su questo dataset potrebbe fornire indicazioni preziose sull'eventuale presenza di strutture più complesse e sulla separabilità dei cluster.
- **Variazione di ulteriori parametri:** Gli esperimenti condotti si sono concentrati principalmente su normalizzazione, numero di cluster e grado di fuzziness. Un approfondimento futuro potrebbe riguardare la variazione sistematica di altri parametri, quali la quantità di rumore aggiunto, la dimensione del test set, il numero minimo di rating per utente e per item, al fine di valutare l'impatto di tali scelte sulle performance e sulla natura dei cluster individuati.
- **Sviluppo e confronto con altre tecniche:** Infine, il framework potrebbe essere esteso per includere e confrontare ulteriori tecniche di clustering fuzzy (ad esempio, Gustafson-Kessel, Possibilistic C-Means) o approcci ibridi che integrino informazioni di contenuto o relazionali, al fine di valutare se tali metodologie possano superare i limiti osservati con FCM.

Appendice A

Appendice A: Repository del Progetto

A.1 Link alla Repository

Il codice sorgente, i risultati ottenuti e i grafici utilizzati per la relazione sono disponibili pubblicamente al seguente indirizzo:

<https://github.com/LilQuacky/fuzzy-recommender-system>