Amazon reviews recommender systems

Elena Bensi[1], Sofia Ferrari[2], and Anna Gotti[3]

[1] Corso di laurea magistrale in Scienze Statistiche, matricola 2018893
elena.bensi@studenti.unipd.it
[2] Corso di laurea magistrale in Scienze Statistiche, matricola 2026885
sofia.ferrari@studenti.unipd.it
[3] Corso di laurea magistrale in Scienze Statistiche, matricola 2023128
anna.gotti@studenti.unipd.it

Sommario La relazione si pone l'obiettivo di costruire sistemi di raccomandazione con approccio *collaborative filtering item-based* per dati provenienti dal sito Amazon.com. In particolare si tenterà di implementare diverse misure di distanza applicate agli *item*, combinandole al metodo di *clustering k-means*

Keywords: Matrice di utilità · Collaborative filtering · Previsioni · RMSE · Similarity measures · k-means

1 Introduzione

In questo progetto verrà affrontato il problema delle raccomandazioni per dati provenienti dal sito Amazon.com, i cui dati sono caratterizzati da un alto numero di *user* rispetto al numero di *item*.

Il problema delle raccomandazioni può essere definito prevedendo la risposta di un *user* ad un nuovo *item*, basandosi sulle informazioni che *user* e *item* portano con loro, consigliando agli *user* nel sistema nuovi ed originali articoli per i quali il *rating* è predetto con valore elevato. In particolare l'approccio scelto è un ***collaborative filtering* (*CF*) *item based***, in cui l'idea chiave è che un *user* $\boldsymbol{u}$ valuti due *item* $\boldsymbol{i}$ e $\boldsymbol{j}$ in modo simile, se altri *user* hanno dato valutazioni simili a questi due articoli.

Punto forte dell'approccio *CF* è che articoli per i quali non si ha alcuna informazione o per i quali è difficile ottenerla, possono essere raccomandati ad *user* attraverso il solo feedback di altri *user*. Perciò il *rating user-item* salvato in memoria può essere direttamente usato per predire *rating* di nuovi *item*. La variante *item based* invece si basa sul fatto che un piccolo numero di *user* altamente simili è preferibile ad un grande numero di *user* le cui misure di similarità non sono affidabili. Per tale ragione, quando il numero di *user* è molto maggiore del numero di *item*, come in nel caso di dati presi da Amazon.com, un approccio di tipo *item-based* produce raccomandazioni più accurate. Anche in termini di efficienza la raccomandazione *item-based* risulta migliore quando il numero di *user* supera il numero di *item*, in quanto richiede molta meno memoria e tempo per calcolare i pesi di similarità.

Per il calcolo della similarità possono essere prese in considerazione varie misure. La scelta della misura è cruciale visto che, nelle stesse condizioni informative, diverse misure possono condurre a differenti risultati. In generale la scelta deve essere fatta sulla base del contesto di lavoro e delle proprietà dei dati. Per tipo di distanza scelta verrà calcolata la previsione del *rating*, e sulla base di questo verranno valutate dal punto di vista dell'accuratezza.

Inoltre sarà implementato il metodo di clustering *k-means*, che raggruppa gli *item* sulla base di distanze da centroidi rappresentativi dei gruppi. Tali gruppi potrebbero essere espressione di pattern non altrimenti individuabili. La scelta di implementare algoritmi di clusterizzazione all'interno di sistemi di raccomandazione segue dalla necessità di una maggiore efficienza computazionale, in quanto con essi verrà ridotto il numero di confronti da calcolare, limitandosi esclusivamente alle entità interne ai *cluster*.

L'obiettivo della relazione è quello di valutare come le combinazioni che vedono coinvolte diverse distanze e l'uso o meno del *clustering,* varino sia in termini di efficienza che di accuratezza previsiva, la prima valutata in termini di tempo di calcolo mentre la seconda in termini di *RMSE*.

2 Base di partenza

Per le procedure descritte nel seguente paragrafo si è fatto riferimento al cap. 4 del libro [**1**] e al cap. 2 del libro [**2**].

I punti centrali dell'implementazione di un sistema di raccomandazione devono tenere in considerazione:

- la normalizzazione dei rating e trattamento dei dati mancanti;
- il calcolo dei pesi di similarità;
- la selezione degli item più simili.

Una volta costruita la matrice di utilità, ovvero la matrice contenente i rating, indicizzata per riga con gli *user* e per colonna con gli *item*, apparirà evidente il problema dell'elevato numero di dati mancanti. La sparsità della matrice verrà trattata tramite normalizzazione dei rating e centramento in media, metodo la cui idea è quella di determinare se un rating è negativo oppure positivo rispetto alla media dei rating. In particolare, utilizzando un approccio di tipo *item-based*, il centramento avverrà intorno alla media per *item*, ponendo pari a 0 , quindi pari alla media, i rating mancanti, ovvero *item* per i quali un *user* non ha dato una valutazione. Per i restanti *rating*, preso il rating non centrato, r_{ui}, relativo all'u-esimo *user* per l'i-esimo *item*, una sua versione trasformata e centrata $h(r_{ui})$ si ottiene sottraendo a r_{ui} la media $\bar{r}_i$ relativa ai rating dati all'*item* i da un insieme di *user*, U_i:

$$h(r_{ui}) = r_{ui} - \bar{r}_i.$$

Tale tecnica di normalizzazione è in questo progetto utilizzata per il sistema di raccomandazione *item-based*, dove la **predizione del rating** r_{ui} relativo all'

i-esimo *item* per l'u-esimo *user*, è scritta come:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij}(r_{uj} - \bar{r_j})}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}, \tag{1}$$

dove $\bar{r}_i$ rappresenta la media per l'i-esimo *item*, w_{ij} il peso assunto dal j-esimo *item* nella previsione dell'i-esimo *item*, $\mathcal{N}_u(i)$ l'insieme di *item* valutati da u, che abbiano almeno un *user* che abbia valutato anche i. La presenza di $\bar{r}_i$ è conseguenza della precedente operazione di centratura intorno alla media dei dati.

Il peso w_{ij} verrà calcolatato attraverso varie di misure di similarità, le quali verranno confrontate tra di loro sulla base di una misura di accuratezza calcolata sui rating predetti. In particolare questo peso evidenzia come, nel calcolo della predizione finale, si dia più importanza agli item più simili.

Nel momento in cui si decide di introdurre il metodo di *clustering*, si considererà una sotto-partizione della matrice di utilità. Accadrà quindi che, nella formula (1) per il calcolo del rating predetto, $\mathcal{N}_u(i)$, precedentemente definito, verrà sostituiro da $\mathcal{N}_u(i)'$, ovvero l'insieme di *item* valutati da u appartenenti allo stesso cluster di i, che abbiano almeno un *user* in comune con i. La predizione del rating si otterrà in questo caso come:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)'} w_{ij}(r_{uj} - \bar{r_j})}{\sum_{j \in \mathcal{N}_u(i)'} |w_{ij}|}. \tag{2}$$

2.1 Misura dell'accuratezza delle previsioni dei rating

Per valutare le performance delle varie versioni del sistema di raccomandazione che di vedranno implementate, si è diviso il dataset in due parti. Il *training set* R_{train}, che contiene l'80% delle osservazioni e utilizzato per allenare i dati per fare previsioni, e *test set* R_{test}, contenente il 20% delle osservazioni, che verrà utilizzato per un confronto con le previsioni basate sul *training*.

Per conoscere la qualità dei risultati ottenuti in questo progetto, si utilizza una misura di accuratezza per valutare le previsioni fatte sui *rating*, il **Root Mean Squared Error (RMSE)**, definito come

$$RMSE = \sqrt{\frac{1}{|R_{test}|} \sum_{(u,i) \in R_{test}} (\hat{r}_{ui} - r_{ui})^2}, \tag{3}$$

dove $\hat{r}_{ui}$ rappresenta la previsione del *rating* r_{ui} e $|R_{test}|$ la cardinalità di R_{test} . Tale misura viene calcolata soltanto per i *rating* che gli *user* presenti in R_{test} hanno già dato a certi *item*, in quanto non è possibile misurare l'accuratezza delle previsioni per *rating* non ancora disponibili.

3 Problema affrontato

In riferimento a questo progetto, il sistema di raccomandazione sviluppato può essere suddiviso in due corpi principali. Il primo riguardante la *creazione e pulizia*

della matrice di utilità e il secondo il *calcolo delle previsioni*. In questo paragrafo vengono descritte le funzioni che compongono il primo corpo che, in quanto legato all'aspetto dell'elaborazione dei dati, vedrà anche proposta un'analisi esplorativa per metterne in luce la natura.

I moduli maggiormente utilizzati per la definizione delle funzioni, costruiti sulla base del linguaggio `Python`, sono `pandas` [**8**] e `numpy` [**9**]. Il primo è stato utilizzato eslcusivamente per lavorare sulla struttura dei dati mentre il secondo è stato scelto per eseguire operazioni di tipo matriciali e vettoriali.

Si è scelto di avvalersi della libreria `pandas` in quanto gli algoritmi che seguono sfruttano ampiamente l'indicizzazione degli oggetti coinvolti nei vari passi. In particolare, la struttura `pandas.DataFrame` consente di operare in maniera intuitiva e semplice sui dati a nostra disposizione. Non è invece stato fatto uso di `pandas` al di fuori delle operazioni di indicizzazione. Infatti, per le restanti operazioni matriciali si è fatto ricorso alle funzioni presenti nella libreria `numpy`, dopo aver opportunamente converitito il `pandas.DataFrame` in un oggetto `numpy.array`. La struttura `pandas.DataFrame` ha consentito di sviluppare una peculiarità non indifferente del codice: esso può essere utilizzato con qualunque *data set* contentente le poche variabili necessarie per l'analisi, apportando minime modifiche al codice.

Aspetti quali il tempo di esecuzione o l'ordine di complessità verranno trattati nel paragrafo 5.

3.1 Analisi esplorativa dei dati

Il *data set* utilizzato per questo progetto è `DigitalMusic`, scaricato da [**7**]. Questo *data set* contiene i *rating* lasciati dagli utenti del sito su articoli appartenenti alla categoria "musica digitale". Ogni riga del *data set* rappresenta un record, e sono note informazioni rigardanti lo *user* (`reviewerID`), l'*item* (`asin`), il *rating* (`overall`) e il momento in cui il *rating* è stato assegnato (`reviewTime`). Il dataset contiene 5541 *user* e 3568 *item*. Il numero totale di *record* è pari a 64706. Si è deciso di considerare solo quegli *item* a cui sono stati asseganti almeno 11 *rating*. Sono stati esclusi allora 12708, pari al 5.1% dei *record* iniziali, relativi agli item che non soddisfano la condizione. Dopo questa prima pulizia si hanno a disposizione 5401 *user* e 1715 *item*.

Successivamente si procede con la costruzione del *training set* e del *test set*. Il primo contiene 40569 *record* mentre il secondo 11429. Le rispettive matrici di utilità hanno una percentuale di sparsità pari a 99,56% per *training set* e 99,81% per *test set*.

É stata effettuatua anche un'ulteriore operazione di pulizia, necessaria per eliminare dalle matrici di utilità gli *item* che, idealmente, non hanno ricevuto alcun *rating* ed escludere gli *user* che, sempre idealmente, non hanno lasciato *rating*. Con riferimento alla *training set* la percentuale di *item* e *user* persi è dell'1,87% e 0,13%, rispettivamente. Si passa quindi da 1715 a 1683 per *item* e da 5401 a 5394 per gli *user*. Per la *test set* si ha una perdita del 13,76% per gli*item* e 5,26% per gli *user*. In questi caso si passa da 1715 a 1479 pr gli *item* e da 3575 a 3387 per gli *user*. Infine, per fare previsioni con i dati della *training*

set su quelli della *test set* è stato necessario estrarre i solo *item* e *user* comuni ai due *set*. Si è giunti ad avere 1466 *item* e 3284 *user* finali

3.2 Divisione del data set

Nelle seguente sezione si illustra la soluzione individuata per effettuare la divisione del *data set* in *training set* e *test set*.

Algoritmo 1 Divisione dei dati in *training set* e *test set*

Input: Dataframe dei dati
Output: *test set*, *training set*, numero di *rating* esclusi alla fine della ripartizione
1: Impostare la frazione `testratio` e inizializzare il contatore di *rating* esclusi `canc`
2: **for** ogni *item i* **do**
3: Individuare il numero `n` di *rating* dati all'*item i*
4: **if** `n` ≤ 10 **then**
5: Aggiornare `canc` di un valore pari a `n`
6: **end if**
7: Calcolare il numero di *rating* dell'*item i* da inserire nel *test set*, ottenuto moltiplicando `testratio` per `n`
8: Ordinare temporalmente i *rating* dell'*item i*
9: Formare i dataframe *training set* e *test set*
10: **end for**

In questo caso è possibile sfruttare anche l'informazione riguardo al momento in cui i *rating* sono stati dati, perciò il *training set* conterrà valutazioni "vecchie" mentre il *test set* sarà composta da valutazioni "recenti". L'idea è quella che rating assegnati in passato possano aiutare nelle raccomandazioni di nuovi *item* per un utente.

3.3 Creazione della matrice di utilità

Nelle seguente sezione si delinea l'implementazione delle funzioni destinate a creare e pulire le matrici di utilità. La matrice di utilità definisce le coppie *user-item* e i relativi *rating*. La seguente funzione crea la matrice di utilità a partire dal dataframe passato in input

Algoritmo 2 Creazione della matrice di utilità con `NaNs`

Input: Dataframe dei dati
Output: Matrice di utilità, indici degli *user*, indici degli *item*
1: Ricavare gli indici degli *user*
2: **for** ogni riga *i* del dataframe **do**
3: Estrarre *i*-simo *user*, *i*-simo *item* e *i*-simo *ratings* dalle liste prima create.
4: Salvare i *rating* in posizione *(i,j)* della matrice di utilità
5: **end for**
6: Ricavare gli indici degli *item*

La matrice costruita con tale funzione avrà dimensione $n \times m$, dove n è il numero di *user* e m è il numero di *item*. La caratteristica principale di questi oggetti è l'elevata sparsità, ovvero la presenza di un gran numero di valori mancanti, indicati da `Nan`, dovuta alle mancanti valutazioni di alcuni*item* da parte degli *user*.

Il successivo algoritmo offre un modo per poter trattare questo problema:

Algoritmo 3 Centratura della matrice di utilità

Input: Matrice di utilità, indici degli *user*, indici degli *item*
Output: Matrice di utilità centrata e senza `NaNs`
1: Mascherare i valori `NaNs`.
2: Sostituire i valori mascherati con il valore medio calcolato per colonna.
3: Rimuovere ad ogni *rating* il valore medio calcolato per colonna.

La funzione restituisce una matrice $n \times m$ dove i *rating* mancanti vengono posti pari al valore 0. Un'altra soluzione che è stata implementata pone i dati mancanti di default pari a 0, lasciando invariati i valori dei rating assegnati. La differenza sta nell'interpretazione che il valore 0 assume. Nel primo caso il valore 0 rappresenta il valor medio dei *rating* per *item*, dato che ai *rating* presenti è stata sottratta la media per *item*. Nel secondo caso esso è un valore di default che può portare ad una maggiore distorsione. Infatti, la prima soluzione al problema dei valori mancanti risulta metodologicamente più corretta, in quanto il valor medio è meno distante dai *rating* rispetto invece al valore 0 del secondo algoritmo, soggetto ad avere una distanza maggiore, fino a 5 punti nel caso di valutazione massima dell'item. Nel progetto si cercherà di prediligere la matrice di utilità centrata per il calcolo delle distanze, quindi per la matrice di utilità derivata dal *training set*. La matrice di utilità con valori nulli posti di default pari a 0 verrà invece presa in considerazione per il calcolo dell'RMSE, sulla *test set*, e per la creazione di gruppi con il metodo *k-means*.

Algoritmo 4 Pulizia della matrice di utilità da righe e colonne con elementi tutti pari a zero

Input: Matrice di utilità
Output: Matrice di utilità pulita, indici degli *item*
1: Eliminare dal dataframe le colonne che hanno tutti valori zero.
2: A partire dal passo 1. eliminare quelle righe che hanno tutti i valori uguali a zero.
3: Ricavare gli indici degli *item*.

L'**Algoritmo 4**, che riassume la funzione di pulizia, vede implementata l'eliminazione delle colonne della matrice di utilità centrata dove ogni valore è pari a 0, ovvero dove agli *item* non è stato assegnato nemmeno un *rating* oppure più verosimilmente (viste le operazioni di *pre processing* implementate) tutti i

rating assegnati avevano valore pari alla media per *item*. Inoltre vengono eliminate le righe fatte solo di 0 che, idealmente, rappresentano user che non hanno dato nemmeno un *rating* oppure che hanno sempre asseganto *rating* pari alla media per *item*. Tale operazione è stata necessaria ai fini del calcolo di alcune distanze che, in presenza di vettori di item pari a 0, non sarebbe stato possibile implementate. Inoltre si è pensato che eliminare queste righe e colonne non comportasse una perdita significativa in termini di informatività in quanto, in fase previsiva, i valori pari a 0 della matrice di utilità saranno trattati tutti come valori mancanti. non verrà quindi tenuto conto della natura dello 0, se dovuto all'assegnazione di un rating "nella media" oppure alla presenza di un valore `Nan`.

4 Metodi proposti

Vengono ora presentate le misure di similarità oggetto di studio in questa relazione e si definisce l'algoritmo di *clustering* utilizzato per raggruppare gli *item*. Per le procedure descritte nel seguente paragrafo si è fatto riferimento all'articolo [**3**] e ai cap. 3, 9 del libro [**5**], per le misure di similarità, mentre, al cap. 7 del libro [**4**] e del libro [**5**], per l'algoritmo di *clustering*.

4.1 Similarity Measures

Le misure di similarità che possono essere utilizzate in sistemi di raccomandazioni *CF* sono varie e la scelta della misura più adatta è decisiva in quanto, a parità di input, misure diverse posso resistuire risultati diversi.

Diventa di interesse indagare le proprietà delle principali misure di similarità e vedere come varia la bontà della procedura tra le diverse funzioni scelte.

In particolare, il calcolo delle misure di similarità si baserà sulla matrice di utilità centrata sulla media per item, nel caso della ***Centered Cosine Similarity*** e dell'***Euclidean Similarity***. Per la ***Jaccard*** è indifferente che la matrice sia centrata o meno perchè, come si vedrà, il *rating* dato all'*item* risulta ininfluente nel calcolo della misura. Infine, la ***Jaccard for bags similarity*** farà uso della matrice di utilità non centrata poichè il valore del rating originario è in questo caso necessario per il calcolo di una versione corretta della *Jaccard Similarity*. Queste misure di similarità, inoltre, verranno utilizzate per determinare il valore del peso del j-esimo *item* nella previsione dell'i-esimo *item*, ovvero definiscono w_{ij}, con la differenza dell'*Euclidean Similarity* dove si è considerato opportuno porre il peso w_{ij} pari al reciproco del valore della distanza ottenuta.

Jaccard Similarity (JS) La *JS* misura la similarità tra due *item* **i** e **j** come il rapporto tra la caridinalità dell'intersezione di x e y e la cardinalità dell'unione di x e y, dove x e y sono i vettori di *user* che hanno assegnato un *rating* agli *item* **i** e **j**, rispettivamente. Essa è quindi definita come:

$$SIM(x, y) = \frac{|x \cap y|}{|x \cup y|}.$$

Questa misura di similarità ha un range compreso tra 0 e 1, dove 0 indica la dissimalirità massima tra i due *item*, ovvero si verifica che nessun *user* ha valutato entrambi gli *item*, rendendo conseguentemente impossibile il confronto, e dove 1 rappresenta la similarità massima, ottenuta quando x e y hanno ricevuto valutazioni dagli stessi *user*.

Il limite principale della *JS* consiste nel fatto che essa non tiene in considerazione il valore del *rating* ma solo il numero di *user* che hanno assegnato un *rating* ad entrambi gli *item*. Ciò ha conseguenze in fase di previsione e valutazione della precisione.

Jaccard Similarity for bags (JSB) Per superare il limite delle *JS* è possibile utilizzare la variante *JSB* che, a differenza della *JS*, tiene in considerazione il valore del *rating* dato da un *user* a un certo *item*. Infatti in questo caso x e y sono i vettori che contengono tante volte l'*user* che ha assegnato un *rating* a **i** e a **j** quanto è il valore del rating assegnato.

La *JSB* sarà quindi definita come rapporto tra la somma del numero minimo di volte che gli *user* appaiono nei due vettori x e y e la somma del numero di volte che gli *user* appaiono nei due vettori.

Questa misura di similarità ha un range compreso tra 0 e $\frac{1}{2}$, dove 0 indica la massima dissimilarità tra due *item*, mentre $\frac{1}{2}$ indica la perfetta similarità tra due *item*.

Centered Cosine Similarity (CCS) Come suggerisce il nome, la *CCS* è equivalente alla *Cosine Similarity* applicata ai dati centrati, ovvero essa viene applicata ai rating a cui è stata sottratta la loro media per *item*.

La *CCS* tra due *item* **i** e **j** è così definita:

$$SIM_{cos}(i,j) = \frac{\sum_{u \in U_{i,j}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U_{i,j}} (r_{ui} - \bar{r}_i)^2 (r_{uj} - \bar{r}_j)^2}},$$

dove $U_{i,j}$ è l'insieme degli *user* che hanno assegnato un *rating* sia a **i** che a **j**, r_{ui} è il *rating* dato dall'u-simo *user* all' i-simo *item* e $\bar{r}_i$ è il *rating* medio dell'$i-$simo *item*.

La misura di similarità considerata ha un range compreso tra -1 e 1. Questa assume valore pari a 1 se i due *item* sono stati valutati allo stesso modo, mentre è pari a -1 quando i due *item* sono stati valutati in maniera opposta. La similarità $SIM_{cos}(i,j)$ sarà pari a zero se nemmeno uno *user* ha valutato entrambi gli *item*.

Questa misura di similarità presenta limiti quando sono presenti *rating* mancanti in quanto diventa impossibile applicare la definzione. Sarà necessario imputare i *rating* mancanti prima di procedere col calcolo della similarità.

Euclidean distance (ED) La *ED* è, tra le misure finora presentate, la più semplice da implementare. Considerati due *item* **i** e **j** essa si definisce come:

$$d(i,j) = \sqrt{\sum_{u \in U_{i,j}} (r_{ui} - r_{uj})^2},$$

dove r_{ui} e r_{uj} sono i vettori dei *rating* dati da uno stesso *user* su due diversi *item*.

A differenza delle misure precedentemente presentate, la *ED* misura la distanza tra *item*, e quindi si avrà una diversa interpretazione dei valori. Infatti, più la distanza $d(i,j)$ si avvicina a 0, più due *item* si assomigliano mentre, più, $d(i,j)$ è grande, più gli *item* sono dissimili tra loro. Per definizione $d(i,j) = 0$ quando i due *item* sono uguali.

4.2 Analisi dei cluster

Come precedentemente sottolineato, parte essenziale per poter ottenere le previsioni è il calcolo delle distanze tra gli *item*. Nel caso di approccio *CF item-based*, questo passaggio può diventare computazionalmente dispendioso quando il numero di *item* risulta elevato.

Verranno allora aggiunti ai metodi tradizionali algoritmi di *clustering* con lo scopo di riunire nello stesso gruppo *item* simili. La definizione di tali raggruppamenti consentirà di ridurre il numero di misure di similarità da calcolare e quindi i tempi di calcolo. Infatti si lavorerà su porzioni della matrice di utilità, definite dal modo in cui sono stati ripartiti gli *item* nei vari *cluster*. A differenza del tempo di esecuzione, non sappiamo quale effetto potrebbe avere il partizionamento degli *item* sull'accuratezza delle previsioni. Tale risultato dipende dalla buona riuscita dell'operazione di *clustering*. Infatti, se i raggruppamenti individuati colgono realmente caratteristiche simili e non visibili tra gli *item*, le previsioni potrebbero risultare più accurate perchè cirscoscritte ad elementi "vicini" in termini di *feature*. Contrariamente, se i raggruppamenti non dovessero cogliere fattori latenti che caratterizzano gli *item*, è possibile che l'accuratezza diminuisca in quanto l'operazione di *clustering* porterebbe all'esclusione di elementi che potrebbero avere un peso rilevante nel calcolo della previsione.

k-means Tale algoritmo definisce una tecnica di *clustering* partizionale per cui, fissato a priori il numero di *cluster*, i dati sono organizzati in gruppi di modo che i punti interni a un *cluster* siano più simili tra loro rispetto ai restanti, e non vi siano sovrapposizioni tra gruppi. Questo approccio presenta numerosi vantaggi, tra i tanti citiamo la ridotta complessità d'implementazione, la velocità di calcolo e la buona qualità dei risultati finali. Di seguito s'illustra l'idea dell'algoritmo.

Si dispone di n dati, in generale di tipo vettoriale, e si è interessati a raggrupparli in k gruppi. L'implementazione prevede tre passi principali:

- **passo 1:** Scegliere in maniera casuale i centroidi per i k *clusters*.

- **passo 2:** Calcolare la *ED* di ogni punto dai k centroidi e assegnare ogni punto all'opportuno *cluster* in base alla distanza minima.
- **passo 3:** Calcolare i nuovi centroidi come la media dei punti interni ai *cluster*.

I passi 2 e 3 verranno iterati finché i centroidi tra due iterazioni successive sono uguali o la differenza è infintesimale.

Come si è già detto, questa metodologia prevede che il numero di *cluster* venga prestabilito. Nel caso in questione si assume che il numero di *cluster* venga fissato pari a $k = 2$, in quanto da analisi fatte sui dati, si è constatato che valori di k superiori a quello scelto portino alla creazione di gruppi con numerosità troppo esigue, addiritura contenenti un solo elemento, rendendo inapplicabili i metodi scelti.

In fase di *clustering*, inoltre, si è deciso di non utilizzare la matrice di utilità centrata in quanto, in questo caso, conduce alla situazione sopra descritta, ovvero ripresenta il problema dei gruppi a numerosità ridotte. Come già accennato nel paragrafo 3, si è ritenuto opportuno lavorare sulla matrice di utilità non centrata, che non produce situazioni simili a questa.

5 Esperimenti

Come precedentemente annunciato, in questo paragrafo ci si occuperà di descrivere gli algoritmi contenuti nelle funzioni atte al *calcolo delle previsioni*. Una volta ottenute tutte le previsioni necessarie ad un confronto con i dati del *test set*, si vedrà utilizzato l'*RMSE* per valutare l'accuratezza delle misure di distanza implementate, sia nel caso di *clustering* che senza raggruppamenti.

5.1 Calcolo delle previsioni

Di seguito verranno illustrati gli algoritmi costruiti per ricavare la previsione di un *rating* mancante.

Algoritmo 5 Calcolo della previsione del *rating*

Input: Item i, user u, Matrice delle distanze, Matrice di utilità, misura di similarità
Output: Previsione p per l'u-esimo user sull'i-esimo item

```
1: Calcolare la similarità tra i e tutti gli item valutati da u e salvarle in un vettore.
2: if il vettore delle similarità è vuoto then
3:     p= media dell'item i.
4: else
5:     Calcolare la previsione secondo la definizione
6: end if
```

La definizione a cui si fa riferimento nel **punto 3** dell'**Algoritmo 5** sono quelle riportate nel paragrafo 2, riassunta dalla formula (1) per la predizione di *rating* senza *cluster*, mentre dalla formula (2) in caso di suddivisione in *cluster*.

La previsione viene adattata a tutte le misure di similarità definite nel paragrafo 4, in particolare al punto 4.1, semplicemente specificando la misura di similarità utilizzata nel calcolo delle distanze nei parametri di input.

Questo algoritmo può essere utilizzato sia nel caso in cui la previsione venga calcolata sul *data set* integrale sia quando si lavora su un sotto gruppo definito dall'algoritmo di *clustering*. Bisogna fare però un distinzione fondamentale tra le due situazioni: nel primo caso `Matrice di utilità` sarà la matrice di utilità centrata, e conseguentemente la previsione dovrà essere riscalata per la media, per riportala sullo spazio iniziale dei dati, mentre nel secondo caso `Matrice di utilità` dovrà essere la matrice di utilità non centrata, rendendo non necessaria il traslamento della previsione. É necessario ribadire che le ragioni che hanno voluto l'uso di questa seconda matrice nel caso di *clustering* sono state guidate dai dati e non dalla metodologia.

Algoritmo 6 Costruzione della matrice delle previsioni

Input: Item i, user u, Matrice1, Matrice2, m righe *train*,New$_{test}$, misura di similarità
Output: Matrice delle predizioni

```
Calcolare delle matrici di distanza.
for ogni utente u e per ogni item i do
    if rating coppia u-i è uguale a 0 then
        Non calcolare la previsione.
    else
        Calcolare la previsione del rating tramite l'Algoritmo 5
    end if
end for
```

Questo algoritmo consente di calcolare la "finta" matrice di utilità del *test set* che verrà poi confrontata con la "vera" matrice di utilità in fase di verifica della bontà della procedura. È fondamentale che questa venga costruita solo per gli *item* e gli *user* comuni a *test set* e *training set* perché altrimenti non sarebbe possibile effettuare i confronti.

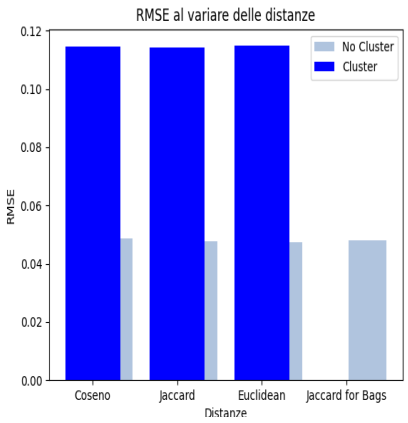
I parametri `Matrice1` e `Matrice2` sono necessari quando la misura di similarità scelta è la *JSB*, in quanto la matrice di utilità per calcolare le similarità tra *item* è diversa da quella usata per calcolare le previsioni. `Matrice1` sarà la matrice di utilità non centrata del *training set* mentre `Matrice2` sarà la matrice di utilità centrata. Nel caso di misure diverse dalla *JSB*, `Matrice1` e `Matrice2` saranno entrambe la matrice di utilità centrata del *training set*.

5.2 Accuratezza ed efficienza degli algoritmi

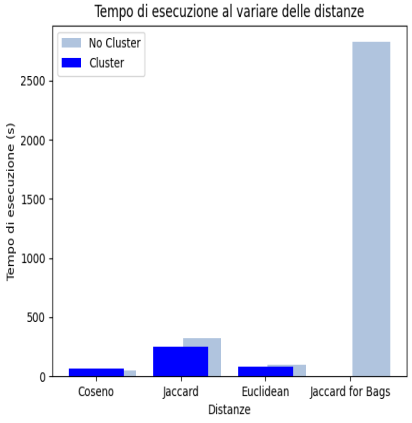
Si vedono qui i risultati delle **misure di accuratezza**, calcolate con l'*RMSE*, utilizzato con l'obiettivo di osservare le variazioni che le diverse distanze e metodi di *clustering* apportano alle previsioni. I valori assunti dall'RMSE in assenza di algoritmi di raggruppamento sono: 0.04876 (*CCS*), 0.04772 (*JS*), 0.04795 (*JSB*)

e 0.04749 (*ED*), che presenta quindi l'*RMSE* più basso tra tutte le misure. Con il *clustering* si ottengono invece i seguenti risultati per l'*RMSE*: 0.11444 (*CCS*), 0.11428 (*JS*) e 0.11478 (*ED*). Si è deciso di non implementare la misura di similarità *Jaccard for Bags* nell'utilizzo dei *cluster* in quanto, come si vede qualche riga sotto, già altamente dispendiosa in termini di costo computazionale senza *clustering*. In generale, i valori ottenuti evidenziano una maggiore accuratezza del metodo implementato senza l'utilizzo di *cluster* e una differenza minima tra le distanze scelte. È possibile notare come i valori ottenuti clusterizzando gli *item* siano il doppio rispetto a quelli ottenuti senza l'utilizzo di tale tecnica, mettendo in luce la maggiore accuratezza del metodo che non vede coinvolto il *clustering*. Il **Grafico 1** riporta tali risultati.

Altro aspetto rilevante per la metodologia presentata è l'**efficienza** del processo, valutata in termini di tempo di esecuzione degli algoritmi implementati per le misure di similarità, senza e con il raggruppamento degli *item*. Tale analisi è stata condotta utilizzando la funzione `time()` del modulo `time`, che restituisce il numero di secondi trascorsi dall'inizio dell'epoca (per i sistemi `Unix` sono le ore 0:00 del primo gennaio 1970) sotto forma di valore a virgola mobile. Il tempo di esecuzione è stato calcolato come la differenza tra il tempo misurato alla fine e all'inizio dell'algoritmo per ciascuna combinazione di metodi implementata. i tempi in secondi, ottenuti rispettivamente senza l'utilizzo di *cluster* sono: 51.924s (*CCS*), 322.158s (*JS*), 2826.765s (*JSB*) e 94.066s(*ED*). L'utilizzo dell'algoritmo di *clustering* produce i seguenti risultati in termini di tempo in secondi: 67.137s (*CCS*), 252.485s (*JS*) e 85.476s (*ED*). Si noti come la situazione migliori notevolmente per la (*JS*). Alla luce delle considerazioni fatte nella Sezione 4.2, l'utilizzo dell'algoritmo di *clustering* porta, come ci si può attendere, ad una notevole riduzione del tempo di esecuzione delle previsioni per le diverse distanze. Il **Grafico 2** riporta tali risultati.



(a) Grafico 1.



(b) Grafico 2.

Riferimenti bibliografici

1. Ricci, Francesco, Rokach, Lior, Shapira, Bracha, and Kantor, Paul B.. Recommender Systems Handbook (1st. ed.). Springer-Verlag, Berlin, Heidelberg, (2010).
2. Aggarwal, Charu C. Recommender systems. - The Textbook (1st. ed.). Springer International Publishing, (2016).
3. Sondur, Mr Sridhar Dilip, Mr Amit P. Chigadani, and Shantharam Nayak. Similarity measures for recommender systems: a comparative study. Journal for Research 2.3 (2016).
4. Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar. Introduction to data mining. Pearson Education India, (2016).
5. Leskovec, Jure and Rajaraman, Anand and Ullman, Jeffrey David. Mining of Massive Datasets. Cambridge University Press, USA, (2016).
6. Aho, Alfred V. and Ullman, Jeffrey D. Foundations of Computer Science. W. H. Freeman Co., USA, (1994).
7. Amazon Review Data (2018): `https://nijianmo.github.io/amazon/index.html`
8. Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010). `http://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf`
9. Jones, E., Oliphant, T., Peterson, P., et al.: NumPy Reference Guide release 1.14.5. `https://docs.scipy.org/doc/numpy-1.14.5/numpy-ref-1.14.5.pdf`