UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea magistrale in
Informatica

IL TITOLO DELLA TESI

Relatore: Relatore 1
Correlatore: Correlatore 1

Tesi di Laurea di:
Lorenzo D'Alessandro
Matr. Nr. 939416

ANNO ACCADEMICO 2020-2021

Dedica

Ringraziamenti

Questa sezione, facoltativa, contiene i ringraziamenti.

Indice

Capitolo 1

Introduzione

Introduzione...

1.1 I contenuti

Spiegazione problema...

1.2 Organizzazione della tesi

Organizzazione tesi...

Capitolo 2

Stato dell'arte

I recommender system sono algoritmi mirati a generare consigli significativi a una insieme di utenti per articoli o prodotti che potrebbero interessarli [1]. La definizione di oggetto è generica e include ad esempio film da guardare, libri da leggere, prodotti da comprare o qualsiasi altra cosa a seconda del contesto in cui sono implementati. Quando gli utenti interagiscono con il sistema generano dei feedback. Questi feedback possono essere di due tipi: espliciti o impliciti. I feedback espliciti sono valori numerici che un utente assegna ad un prodotto, i feedback impliciti riflettono indirettamente le opinioni di un utente osservando la cronologia degli acquisti, i link clickati, gli elementi visualizzati, etc. Basandosi sui feedback passati, i RS imparano un modello per prevedere quanto un utente può essere interessato a nuovi oggetti. Questi oggetti sono poi ordinati in base alla pertinenza prevista per l'utente. In ultimo, gli oggetti con il rank più alto vengono suggeriti all'utente. La relazione tra utenti e oggetti è rappresentata con una matrice R_M in cui sono memorizzati i rating passati degli utenti. La *ratings matrix* è definita come:

$$R_M : U \times I \to R$$

dove $U = \{u_1, \ldots, u_m\}$ rappresenta l'insieme degli utenti, $I = \{i_1, \ldots, i_n\}$ rappresenta l'insieme degli utenti, e $R = \{r_1, \ldots, r_k\}$ rappresenta l'insieme dei possibili ratings che un utente ha espresso riguardo a degli oggetti. Un valore mancante nella ratings matrix può avere due significati: l'utente non vuole esprimere un'opinione su un oggetto specifico, oppure l'utente non conoscendo ancora l'oggetto non può averlo valutato [2].

I recommender system si dividono principalmente in tre categorie:

- collaborative filtering
- content-based

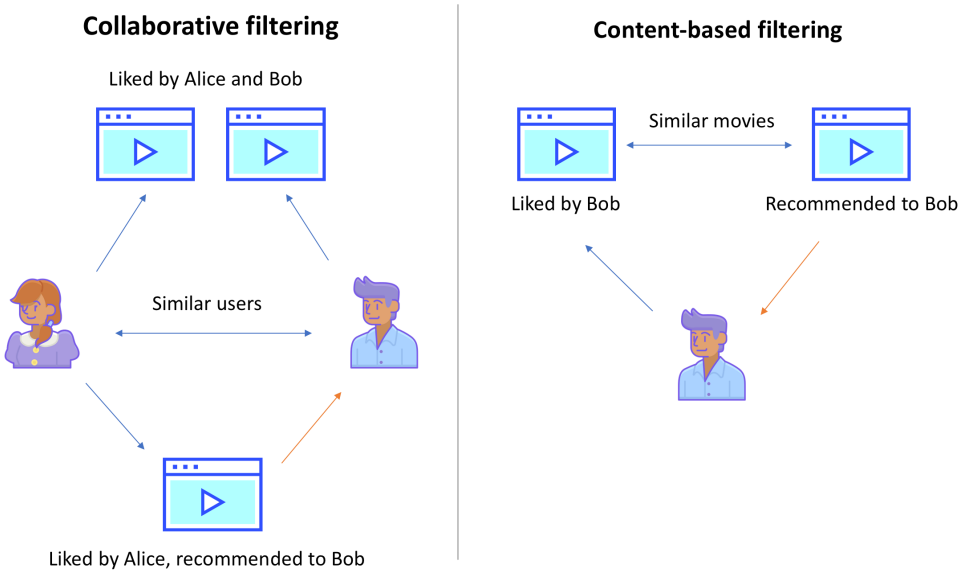

Figura 1: Collaborative filtering vs Content-based

- hybrid

In figura Figura 1 è schematizzato il modo diverso in cui operano i metodi collaborative filtering rispetto a quelli content-based.

2.1 Collaborative filtering recommender system

Collaborative filtering (CF) è considerata la tecnica di raccomandazione più popolare e ampiamente utilizzata nei RS. Il presupposto alla base di CF è che le persone con preferenze simili valuteranno gli stessi oggetti con ratings simili. CF quindi sfrutta le informazioni sul comportamento passato o le opinioni di una comunità di utenti esistente per prevedere quali elementi potranno piacere o saranno interessanti per l'utente corrente del sistema [3]. Gli approcci CF puri non sfruttano né richiedono alcuna conoscenza degli oggetti stessi ma solo dei feedback degli utenti.

La classe di algoritmi più famosa è quella di Matrix Factorization, ma recentemente sono stati sviluppati diversi approcci basati sul deep learning [4].

2.1.1 Vantaggi e svantaggi

Vantaggi:

- *Nessuna conoscenza del dominio necessaria:* il modello può funzionare in domini in cui non c'è molto contenuto associato agli oggetti e in cui il contenuto è difficile da analizzare per un sistema informatico (come opinioni e ideali) [5].

- *Serendipity:* il modello ha la capacità di fornire consigli fortuiti, il che significa che può consigliare elementi pertinenti per l'utente anche senza che il contenuto si trovi nel profilo dell'utente permettendo all'utente di scoprire nuovi interessi [5] [6].

Svantaggi:

- *Problema del cold-start:* Si riferisce alla situazione in cui il sistema di raccomandazione non ha abbastanza informazioni su un utente od un oggetto per poter fare previsioni rilevanti. Un nuovo oggetto inserito nel RS di solito non ha voti, ed è quindi improbabile che venga raccomandato. Un oggetto non consigliato passa inosservato a gran parte della community. Il problema è presente anche per i nuovi utenti: gli utenti che hanno espresso nessuna o poche valutazioni non ricevono raccomandazioni affidabili [7].

- *Problema di data sparsity:* questo è il problema che si verifica a causa della mancanza di informazioni sufficienti, cioè quando solo pochi rispetto al numero totale di oggetti disponibili in un database sono valutati dagli utenti. Ciò porta ad una rating matrix sparsa, e raccomandazioni poco efficaci [5].

- *Scalabilità:* Questo è un altro problema associato agli algoritmi di raccomandazione perchè il tempo di computazione cresce linearmente con il numero di utenti e oggetti. Un sistema di raccomandazione efficiente con un dataset limitato potrebbe non esserlo quando il volume è incrementato [5].

2.1.2 Matrix factorization

Gli algoritmi basati su matrix factorization(MF) caratterizzano sia utenti che oggetti mediante dei vettori di fattori estratti dai pattern sui ratings. Una corrispondenza alta tra i fattori di un utente e un oggetto porta ad una raccomandazione. Questi metodi sono diventati popolari negli ultimi anni perchè combinano scalabilità e accuratezza.

Più formalmente, i modelli basati su matrix factorization mappano utenti e oggetti in uno spazio di fattori latenti di dimensionalità f, tale che le interazioni tra utenti e oggetti sono modellate come prodotti in quello spazio. Di conseguenza, ogni oggetto i è associato con un vettore $q_i \in \mathbb{R}^f$, e ogni utente u con un vettore $p_u \in \mathbb{R}^f$. Per un dato oggetto i, gli elementi di q_i indicano la misura in cui l'oggetto

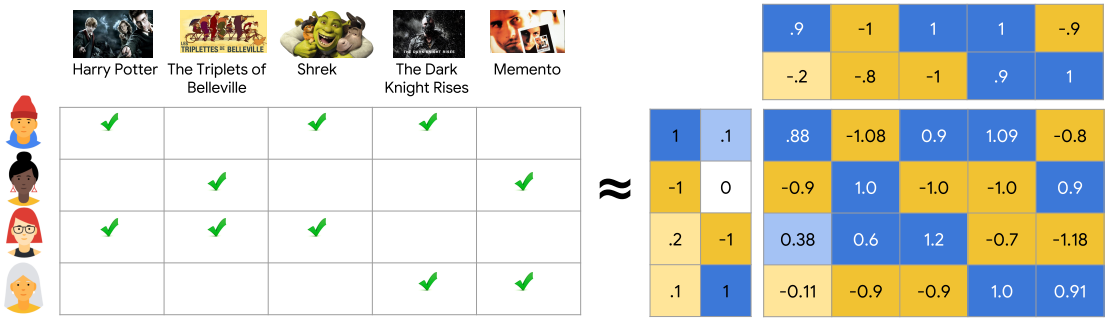

Figura 2: Approssimazione matrice dei ratings con matrix factorization

possiede quei fattori, positivi o negativi. Per un dato utente u, gli elementi di p_u indicato l'entità dell'interesse che l'utente ha per gli oggetti che hanno un valore alto sui fattori corrispondenti, positivi o negativi. Il prodotto scalare $q_i^T p_u$ indica l'interesse dell'utente u per le caratteristiche dell'oggetto i [8]. Quindi il rating r_{ui} può essere approssimato come

$$r_{ui} = q_i^T p_u \tag{1}$$

Il problema principale è calcolare il mapping di ogni oggetto e utente su dei vettori $q_i, p_u \in \mathbb{R}^f$. Una volta che il recommender system ha completato il mapping, può facilmente stimare il rating che un utente darà a qualsiasi oggetto utilizzando l'equazione 1.

In Figura 2 è mostrato come la matrice dei ratings $A \in \mathbb{R}^{m \times n}$, con m numero di utenti e n numero di oggetti, viene decomposta in due matrici di dimensionalità minore:

- Una matrice di embedding per gli utenti $P \in \mathbb{R}^{m \times f}$, in cui la riga i è l'embedding dell'utente i
- Una matrice di embedding per gli oggetti $Q \in \mathbb{R}^{n \times f}$, in cui la colonna j è l'embedding dell'utente j

Gli embedding di P e Q sono imparati in modo tale che il prodotto PQ^T sia una buona approssimazione della matrice dei ratings A

2.2 Content-based recommender system

Nei recommender system content-based (CB), gli attributi descrittivi degli oggetti sono usati per produrre raccomandazioni. Il termine "content" indica queste

descrizioni. Nei metodi content-based, i ratings degli utenti sono combinati con le informazioni disponibili per gli oggetti, per poi essere usati come training data per creare un modello di classificazione o regressione specifico per l'utente. Questo modello specifico dell'utente viene utilizzato per prevedere se alla persona corrispondente piacerà un articolo per il quale la sua valutazione è ancora sconosciuta [9].

Mentre nei CF la similarità tra due oggetti (o due utenti) è calcolato come la correlazione o la similarità tra i ratings forniti dagli altri utenti, i recommender system content-based sono progettati per consigliare oggetti simili a quelli che l'utente ha preferito in passato. Non considerando gli altri utenti, la lista di raccomandazioni può essere generata anche se c'è un solo utente.

2.2.1 Vantaggi e svantaggi

Vantaggi:

- *Consigliare nuovi oggetti:* Questi modelli hanno la capacità di consigliare nuovi oggetti anche se non ci sono valutazioni fornite dagli utenti a differenza dei modelli collaborative filtering [5].
- ???

Svantaggi:

- *Features degli oggetti:* la precisione del modello dipende dall'insieme delle features che descrivono gli oggetti. Identificare le features più rilevanti non è semplice e dipende molto dall'applicazione specifica [2].
- *Content overspecialization:* Dato che i metodi CB si affidano solo alle caratteristiche degli oggetti già valutati dall'utente corrente, egli riceverà solo raccomandazioni simili ad altri oggetti già definiti nel suo profilo [5].
- *Dimensione training set:* In modo da consentire a un RS content-based di imparare le preferenze di un utente, egli dovrà valutare un numero sufficiente di oggetti. In caso contrario, il RS non avrà sufficienti informazioni per imparare un modello accurato e fallirà nel raccomandare oggetti a utenti con pochi o nessun ratings.

2.3 Hybrid recommender system

I modelli ibridi combinano tipi diversi di sistemi di raccomandazione per formare dei modelli in grado di superare le debolezze dei modelli singoli. In [9] sono descritti tre modi per creare recommender system ibridi:

1. *Ensemble design:* Con questo metodo i risultati degli algoritmi base sono combinati in un output singolo più robusto. Il principio fondamentale è molto simile ai metodi di ensemble usati in molte applicazioni di data mining come clustering, classificazione e analisi degli outlier. Gli ensemble design possono essere formalizzati nel modo seguente. Sia R^k una matrice $m \times n$ contenente le predizioni di m utenti per n oggetti dell'algoritmo k-esimo, con $k \in \{1, \ldots, q\}$. Pertanto, un totale di q algoritmi diversi sono usati per ottenere queste predizioni. L'elemento (u, j)-esimo di R^k contiene il rating predetto per l'utente u sull'oggetto j dall'algoritmo k-esimo. Gli elementi della matrice originale R sono replicati in ogni R^k, e solo gli elementi non presenti in R variano nei differenti R^k a causa dei diversi risultati degli algoritmi. Il risultato finale è ottenuto combinando le predizioni $R^1, \ldots, R^q$ in un singolo output. La combinazione può essere fatta in vari modi, ad esempio calcolando la media pesata delle varie predizioni. Le caratteristiche comuni di questi algoritmi sono usare sistemi di raccomandazione già esistenti e produrre uno score/ranking unico.

2. *Monolithic design:* In questo caso, viene creato un algoritmo di raccomandazione integrato utilizzando vari tipi di dati. A volte non esiste una chiara distinzione tra le varie parti (es. content-based e collaborative filtering) dell'algoritmo. In altri casi, potrebbe essere necessario modificare gli algoritmi algoritmi di raccomandazione esistenti per essere usati all'interno dell'approccio generale, anche quando c'è una chiara distinzione tra gli algoritmi utilizzati. Pertanto, questo approccio tende a integrare più strettamente le varie fonti di dati e non è possibile visualizzare facilmente i singoli componenti come black-boxes separate.

3. *Mixed systems:* Come per gli ensembles, questi sistemi usano diversi algoritmi di raccomandazione come black-boxes, ma gli oggetti raccomandati dai vari sistemi sono presentati insieme senza essere combinati.

2.4 Context-aware recommender system

I recommender system context-aware (CARS) producono le loro raccomandazioni utilizzando informazioni addizionali che definiscono la situazione di un utente. Queste informazioni addizionali sono chiamate *contesto*.
Alcuni esempi di contesto sono:

1. *Data e ora:* Dalle informazioni di data e ora è possibile estrarre diverse features contestuali come il momento della giornata, il giorno della settimana, week-end, vacanze, stagioni ed altro ancora. Una raccomandazione

potrebbe essere rilevante la mattina ma non il pomeriggio, e viceversa. Le raccomandazioni sui vestiti invernali o estivi possono essere molto diverse.

2. *Posizione:* Con la crescente popolarità del GPS disponibile ormai su qualunque telefono, le raccomandazioni sensibili alla posizione dell'utente hanno guadagnato importanza. Per esempio, un viaggiatore potrebbe desiderare raccomandazioni su ristoranti vicini alla propria posizione. Questo può essere fatto aggiungendo la posizione come contesto nel recommender system.

3. *Informazioni sociali:* Il contesto sociale è spesso importante per un sistema di raccomandazione. Le informazioni su amici, tag e cerchie sociali di un utente possono avere un impatto sul processo di raccomandazione. Per esempio una persona potrebbe scegliere di guardare un film diverso a seconda che lo guardi con i suoi genitori o con i suoi amici.

Il contesto può essere ottenuto in vari modi [10] che includono:

1. *Esplicitamente* ponendo domande dirette alle persone rilevanti o richiedendo le informazioni con altri mezzi. Per esempio, un sito web potrebbe ottenere informazioni contestuali chiedendo agli utenti di compilare un form.

2. *Implicitamente* dai dati o dall'ambiente, come il cambio di posizione rilevato da una compagnia di telefonia mobile. In questo caso non è necessario fare nulla in termini di interazione con l'utente o altre fonti di informazioni contestuali perché l'informazione contestuale è acceduta direttamente e i dati sono estratti da essa.

3. *Per inferenza* usando metodi statistici o di data mining.

2.4.1 Approccio multidimensionale

Il problema tradizionale di raccomandazione può essere visto come apprendere una funzione che associa le coppie utente-oggetto ai ratings. La funzione corrispondete f_R è definita come:

$$f_R : U \times I \rightarrow rating$$

Quindi la rating function mappa da uno spazio bidimensionale di utenti e oggetti ai ratings. I CARS generalizzano questo approccio utilizzando un approccio multidimensionale in cui la ratings function può essere vista come un mapping da una matrice n-dimensionale all'insieme dei ratings [2].

$$f_R : D_1 \times D_2 \cdots \times D_n \rightarrow rating$$

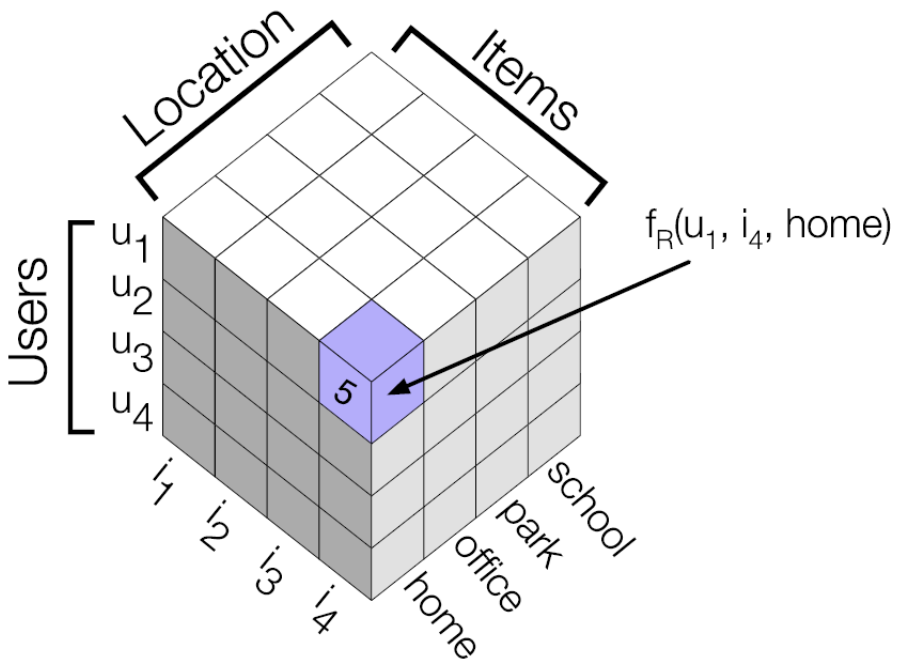

Figura 3: Esempio di un cubo multidimensionale per $User \times Item \times Location$

In questo caso, il risultato è un cubo n-dimensionale al posto che una matrice bidimensionale. Le diverse dimensioni sono denotate come $D_1 \dots D_n$. Due di queste dimensioni saranno sempre utenti e oggetti, le altre D_i dimensioni corrispondono alle features del contesto [9]. In Figura 3 è mostrato un esempio di un cubo tridimensionale che memorizza i ratings per $User \times Items \times Location$, in cui $f_R(u_1, i_4, home) = 5$ significa che l'utente u_1 ha valutato con un punteggio pari a 5 l'oggetto i_4 mentre era a casa.

Il contesto può essere applicato nelle varie fasi del processo di raccomandazione. Come rappresentato in Figura 4 si possono identificare tre paradigmi principali:

1. *Contextual pre-filtering:* In questo paradigma, le informazioni riguardo il contesto attuale sono utilizzate per selezionare o costruire l'insieme dei dati rilevanti (la matrice dei ratings). Poi i ratings mancanti dai dati selezionati possono essere predetti utilizzando qualsiasi sistema di raccomandazione 2D tradizionale.

2. *Contextual post-filtering:* In questo paradigma, le informazioni contestuali sono inizialmente ignorate e i ratings sono predetti utilizzando qualsiasi sistema di raccomandazione 2D traidizionale. Poi, l'insieme di raccomandazioni non rilevanti nel contesto c sono filtrate, e la lista di raccomandazioni è sistemata in base a c.

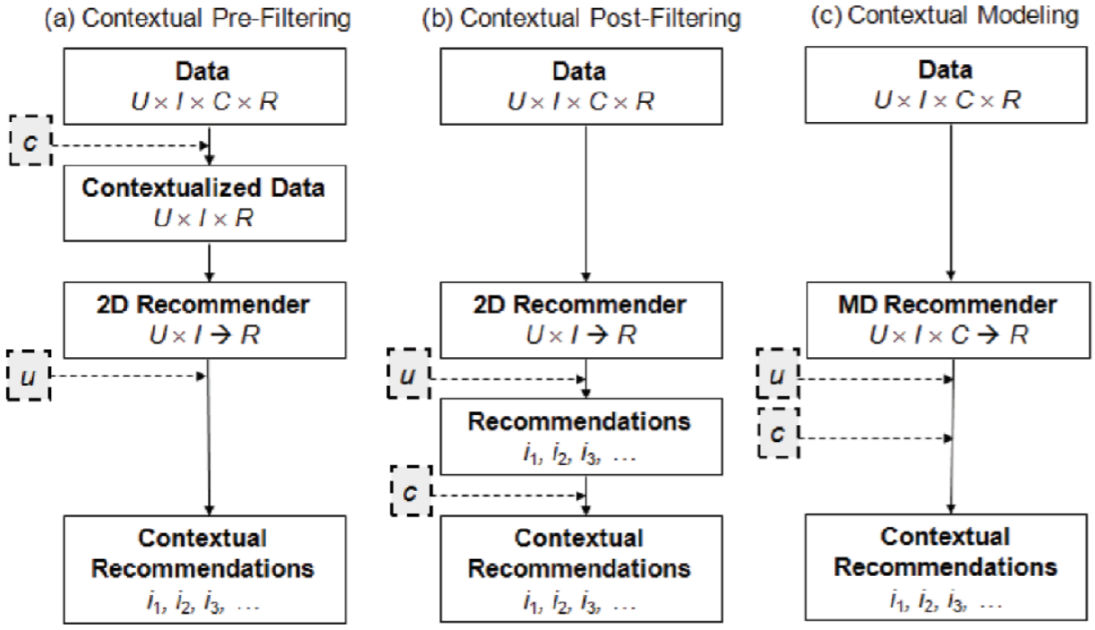

Figura 4: Paradigmi per incorporare il contesto nei sistemi di raccomandazione

3. *Contextual modeling:* Mentre gli approcci di contextual pre-filtering e post filterning fanno uso di una funzione di raccomandazione 2D, gli approcci di contextual modeling danno luogo a funzioni di raccomandazione veramente multidimensionali che rappresentano modelli predittivi o euristiche che incorporano informazioni contestuali in aggiunta ai dati di utenti e oggetti.

2.5 Alternating least square

Alternating least square (ALS) [11] è un algoritmo di matrix factorization stato dell'arte per quanto riguarda i feedback impliciti. ALS è un processo di ottimizzazione iterativo in cui ad ogni iterazione si cerca di arrivare il più vicino possibile a una rappresentazione fattorizzata dei dati originali [12].

2.5.1 Feedback impliciti

I feedback impliciti hanno alcune importanti caratteristiche che li distinguono dai feedback espliciti e impediscono di usare direttamente algoritmi progettati con i feedback espliciti in mente [11]:

1. *Non ci sono feedback negativi.* Osservando il comportamento di un utente, è possibile inferire quali oggetti gli interessano e che quindi ha scelto di consumare. É difficile però capire in modo affidabile quali oggetti l'utente non apprezza. Per esempio, un utente che non ha guardato una serie tv potrebbe non averlo fatto perchè non interessato a quella serie, oppure perché non la conosceva. Questa asimmetria non esiste nei feedback espliciti in cui un utente si esprime su cosa gli piace e cosa non gli piace. Anche i dati mancanti sono un problema, nei feedback espliciti si conosce quali ratings non sono stati espressi dall'utente, nei feedback impliciti no.

2. *I feedback impliciti sono intrinsicamente rumorosi.* Tenendo traccia passivamente dei comportamenti degli utenti è difficile distinguere il caso in cui un utente ha consumato un oggetto perchè davvero interessato o per altri motivi.

3. Il valore numerico dei feedback espliciti indica la *preferenza*, mentre il valore numerico dei feedback impliciti indica la *confidenza*. I sistemi basati sui feedback espliciti permettono all'utente di impostare il loro livello di preferenza su un oggetto, ad esempio con un voto da 1 a 5. I feedback impliciti invece descrivono la frequenza di un'azione, ma un valore più alto non indica per forza una preferenza maggiore

2.5.2 Il modello

Per prima cosa vanno formalizzati i concetti di preferenza e confidenza. La preferenza di un utente u per un item i è indicata con un valore binario p_{ui}:

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

In pratica, se un utente u ha consumato un oggetto i ($r_{ui} > 0$), allora si ha un indicazione che u è interessato a i. D'altro canto, se u non ha mai consumato i la preferenza è uguale a 0. In generale, con l'aumento del valore di r_{ui} si ha un indicazione più forte che l'utente sia davvero interessato all'oggetto. Di conseguenza, si può indicare con c_{ui} la confidenza nell'osservare p_{ui}. Una possibile scelta è

$$c_{ui} = 1 + \alpha r_{ui}$$

In questo modo, si ha una confidenza minima su p_{ui} per ogni coppia utente-oggetto, ma osservando più preferenze positive la confidenza su $p_{ui} = 1$ aumenta. La velocità di incremento è controllata dalla costante α

Come spiegato nella sottosezione 2.1.2, l'obbiettivo è trovare un vettore $x_u \in R^f$ per ogni utente u, ed un vettore $y_i \in R^f$ per ogni oggetto i che fattorizzano le preferenze degli utenti. Le preferenze possono essere poi calcolate come $p_{ui} = x_u^T y_i$. I vettori latenti in ALS sono calcolati minimizzando la funzione obbiettivo:

$$\min_{x_*, y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u ||x_u||^2 + \sum_i ||y_i||^2 \right)$$

Il termine $\lambda \left(\sum_u ||x_u||^2 + \sum_i ||y_i||^2 \right)$ è necessario per regolarizzare il modello ed evitare l'overfitting durante il training. Il valore esatto di λ dipende dai dati e si determina tramite cross validation. Quando il vettore latente degli utenti o degli oggetti rimane fissato, la funzione obbiettivo diventa quadratica è si può calcolare un minimo globale. Questo porta ad un processo di ottimizzazione con il metodo dei minimi quadrati, in cui si alterna tra il ricalcolare il vettore degli utenti mantenendo fissato quello degli oggetti e viceversa. Ad ogni step il valore della funzioni di costo diminuisce.

2.6 Neural collaborative filtering

Gli algoritmi basati su matrix factorization sono sicuramente i più popolari nell'ambito dei sistemi di raccomandazione collaborative filtering. Nonostante l'efficacia di questi modello, le loro performance posso variare in base alla scelta della funzione di interazione. Ad esempio, per quanto riguarda il task di rating prediction su feedback espliciti, è noto che le performance di MF possono essere migliorate incorporando il bias di utenti e oggetti nella funzione di interazione. Anche se si tratta solo di una piccola modifica, dimostra l'effetto positivo di progettare una migliore funzione usata per modellare l'interazione delle feature latenti di utenti e oggetti. Secondo gli autori di [13] il prodotto scalare, che combina la moltiplicazione delle features latenti in modo lineare, potrebbe non essere sufficiente per catturare la complessa struttura dei dati che rappresentano le interazioni dell'utente.

La Figura 5 mostra come il prodotto scalare può limitare l'espressività di MF. La similarità tra due utenti può essere misurata con il prodotto scalare o in modo equivalente con il coseno dell'angolo tra i loro vettori latenti. Dalla matrice utenti-oggetti in figura indicata con (a), l'utente u_4 è più simile a u_1, seguito da u_3, e in ultimo da u_2. Tuttavia, nello spazio latente indicato con (b), posizionare p_4 più vicino a p_1 rende p_4 più vicino a p_2 e non a p_3.

Viene quindi proposto di usare le reti neurali che sono considerate approssimatori universali [14] per imparare le interazioni utente-oggetto. In particolare sono proposti tre modelli basati su deep neural networks:

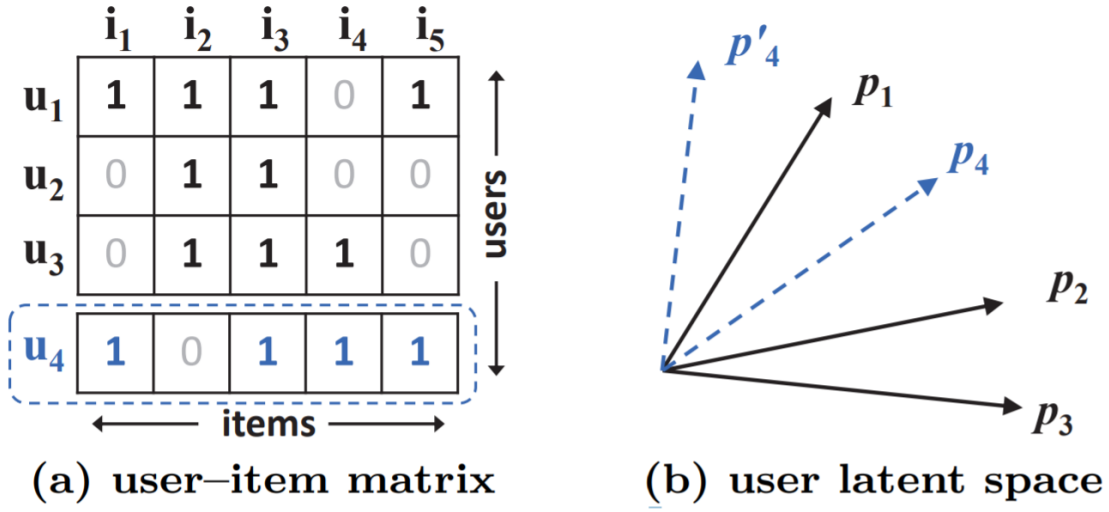

Figura 5: Limitazioni di matrix factorization

1. Multi-Layer Perceptron (MLP)
2. Generalized Matrix Factorization (GMF)
3. Neural Matrix Factorization (NeuMF)

2.6.1 Multi-layer perceptron

Il primo modello proposto è una rete neurale multi-layer che modella le interazioni utente-oggetto y_{ui}. Il layer di input consiste di due vettori v_u^U e v_i^I che descrivono rispettivamente l'utente u e l'oggetto i. Dato che l'input nei modelli collaborative filtering è composto dagli ID univoci di u e i, essi devono essere convertiti con one-hot encoding in dei vettori binari sparsi. Sopra al layer di input c'è il layer di embedding; è un layer fully connected che proietta la rappresentazione sparsa in un vettore denso. Gli embedding di utenti e oggetti ottenuti possono essere visti come i vettori latenti di utenti e oggetti nel contesto dei modelli a fattori latenti come matrix factorization. I vettori di embedding di user e item sono dati in input a un architettura neurale multi-layer, chiamati neural collaborative filterings layers, e infine al layer di output per calcolare lo score predetto y_{ui}. In Figura 6 è rappresentata l'architettura del MLP con X neural CF layers.

2.6.2 Generalized matrix factorization

In [13] viene poi mostrato come la famiglia di metodi basata su matrix factorization possa essere approssimata utilizzando una rete neurale. In questa rete il layer di

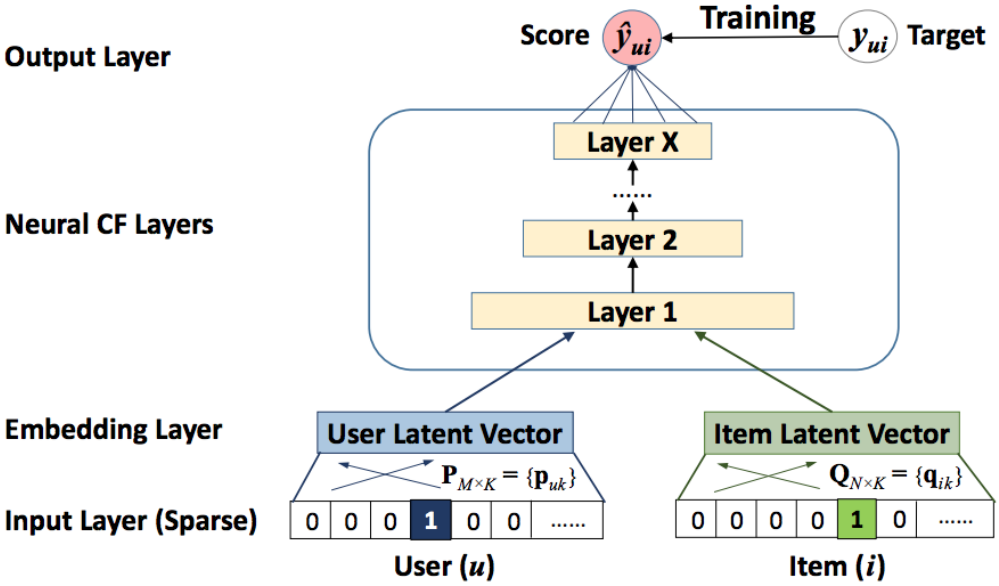

Figura 6: multi-layer perceptron

input e il layer di embedding sono gli stessi del modello MLP, la differenza è nei neural collaborative filtering layers. Come detto prima i vettori di embedding possono essere visti come i vettori latenti di utenti e oggetti imparati dagli algoritmi di MF. Siano p_u il vettore latente degli utenti e q_i il vettore latente degli oggetti. Si può ridefinire la funzione del primo neural CF layer per eseguire la moltiplicazione tra i due vettori di embedding:

$$\phi(p_u, q_1) = p_u \odot q_i$$

in cui $\odot$ è il prodotto elemento per elemento dei vettori. Si proietta poi il vettore risultato sul layer di output:

$$y_{ui} = a_{out}(h^T(p_u \odot q_i))$$

dove a_{out} e h sono rispettivamente la funzione di attivazione e i pesi del layer di output. In questo modo la rete neurale è utilizzata per simulare la famiglia di metodi basata su MF ed è chiamato generalized matrix factorization (GMF).

2.6.3 Neural matrix factorization

A questo punto si hanno due diverse implementazioni di recommender system collaborative filtering basati su reti neurali: GMF che applica un kernel lineare per

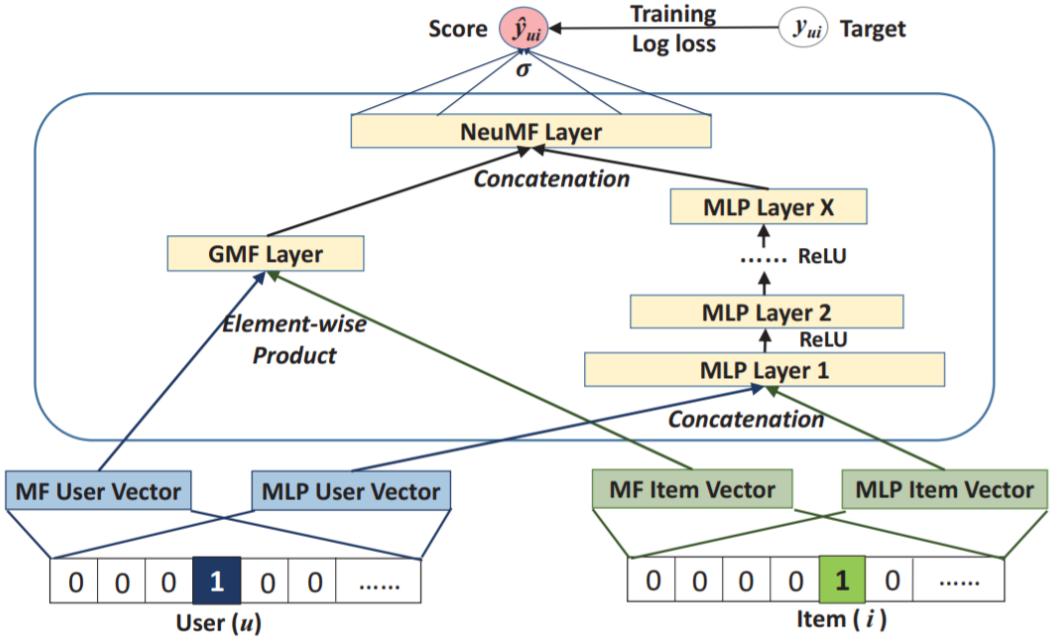

Figura 7: Neural matrix factorization

per imparare le interazioni delle features latenti, e MLP che invece applica un kernel non lineare. Le due reti possono essere combinate per modellare in modo più preciso le complesse interazioni utente-oggetto. Una soluzione semplice è permettere a GMF e MLP di condividere lo stesso layer di embedding, questo può limitare le performance del modello perchè GMF e MLP devono usare embedding della stessa dimensione. Per fornire più flessibilità al modello fuso, GMF e MLP imparano vettori di embedding separati, e i due modelli sono combinati concatenando l'output del loro ultimo layer nascosto. L'architettura del modello fuso chiamata neural matrix factorization (NeuMF) è mostrata in Figura 7.

2.7 Raccomandazioni context-aware con modelli di deep learning

I sistemi di raccomandazione context-aware tradizionali usano principalmente un set predefinito di informazioni contestuali esplicite. Il contesto specifico descrive le circostanze in cui le informazioni sono state raccolte come il meteo (soleggiato, nuvoloso, etc.), o il tempo (giorno della settimana, ora, etc.). Il vantaggio principale è la bassa dimensionalità del contesto che permette di integrarlo facilmente

nei sistemi di raccomandazione esistenti [15]. Infatti un dataset con molte features porta naturalmente ad uno spazio multidimensionale e quindi a sparsità. Questo fenomeno è conosciuto come curse of dimensionality [16].

Tuttavia, i CARS espliciti hanno le seguenti limitazioni: (1) la selezione del contesto specifico è un task che richiede tempo essendo fatto a mano da esperti di dominio, (2) il contesto selezionato potrebbe non rappresentare il set di features contestuali più efficace per il recommender system in questione; (3) l'utilizzo di contesti espliciti, come la posizione dell'utente, può sollevare problemi di privacy [15]. La limitazione sul numero di features contestuali potrebbe essere un problema in quegli ambienti in cui il contesto è complesso e dinamico. Ad esempio, sfruttando in numerosi sensori presenti sugli smartphone come accelerometro, campo magnetico, GPS, e sensore di luminosità, possono essere raccolta informazioni di contesto ad alta dimensionalità. Queste informazioni sono poi utilizzate per inferire il contesto e il comportamento dell'utente; dall'accelerometro si può capire l'attività dell'utente (es. camminare, stare seduto, correre, etc.), mentre con il GPS si può inferire la posizione (es. a casa, al lavoro, etc.).

Per risolvere i problemi legati all'utilizzo del contesto multidimensionale viene proposto di ridurre la dimensionalità del contesto con autoencoder o PCA[17] [18], di costruire una rappresentazione gerarchica del contesto [19], e di usare dei modelli di deep learning in grado di supportare molte features di contesto[15].

2.7.1 Estrazione del contesto latente

Come detto prima, il contesto ad alta dimensionalità è spesso composto da dati di sensori (GPS, accelerometro, etc.) che sono altamente correlati. Si può usare un auto-encoder per scoprire le correlazioni tra features differenti ed estrarre una rappresentazione a bassa dimensionalità del contesto [17]. Un autoencoder è una rete neurale che trasforma l'input ad alta dimensionalità in una rappresentazione latente a bassa dimensionalità (encoder), poi esegue una ricostruzione dell'input originale a partire rappresentazione latente (decoder) [20].

Algorithm 1 Estrazione del contesto latente usando un autoencoder

Input: S
Output: O

```
1: y ← 1
2: if n < 0 then
3:    X ← 1/x
4:    N ← −n
5: else
6:    X ← x
7:    N ← n
8: end if
9: while N ≠ 0 do
10:   if N is even then
11:      X ← X × X
12:      N ← N/2
13:   else {N is odd}
14:      y ← y × X
15:      N ← N − 1
16:   end if
17: end while
```

Capitolo 3

Capitolo 3

Capitolo 4

Capitolo 4

Capitolo 5

Capitolo 5

Capitolo 6

Conclusioni

6.1 Conclusioni

Conclusioni...

6.2 Sviluppi futuri

Sviluppi futuri...

Bibliografia

[1] Prem Melville and Vikas Sindhwani. *Recommender Systems*, pages 829–838. Springer US, Boston, MA, 2010.

[2] Mattia G. Campana and Franca Delmastro. Recommender systems for online and mobile social networks: A survey. *Online Social Networks and Media*, 3-4:75–97, 2017.

[3] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, USA, 1st edition, 2010.

[4] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system. *ACM Computing Surveys*, 52(1):1–38, Feb 2019.

[5] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273, 2015.

[6] Google Developers. Collaborative filtering advantages and disadvantages. `https://developers.google.com/machine-learning/recommendation/collaborative/summary`, 2020.

[7] Jesùs Bobadilla, Fernando Ortega, Antonio Hernando, and Jesùs Bernal. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-Based Systems*, 26:225–238, 2012.

[8] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[9] Charu C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016.

[10] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[11] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008.

[12] Victor Köhler. Als implicit collaborative filtering. `https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe`, 2017.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 173–182, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[15] Moshe Unger, Alexander Tuzhilin, and Amit Livne. Context-aware recommendations based on deep learning frameworks. *ACM Trans. Manage. Inf. Syst.*, 11(2), May 2020.

[16] Shaina Raza and Chen Ding. Progress in context-aware recommender systems — an overview. *Computer Science Review*, 31:84–97, 2019.

[17] Moshe Unger, Ariel Bar, Bracha Shapira, and Lior Rokach. Towards latent context-aware recommendation systems. *Knowledge-Based Systems*, 104, 04 2016.

[18] Moshe Unger, Bracha Shapira, Lior Rokach, and Amit Livne. Inferring contextual preferences using deep encoder-decoder learners. *New Review of Hypermedia and Multimedia*, 24(3):262–290, 2018.

[19] Moshe Unger and Alexander Tuzhilin. Hierarchical latent context representation for context-aware recommendations. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[20] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.