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 (RS) sono algoritmi mirati a generare consigli significativi a un 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, punti di interesse, etc. Quando gli utenti interagiscono con il sistema di raccomandazione 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 aperti, gli elementi visualizzati, etc. Basandosi sui feedback passati, i sistemi di raccomandazione 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 *rating 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 oggetti, e $R = \{r_1, \ldots, r_k\}$ rappresenta l'insieme dei possibili rating che un utente ha espresso riguardo a degli oggetti [2]. Un valore mancante nella rating 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. La matrice dei rating è tipicamente molto sparsa: il numero di oggetti valutati da un utente è molto minore rispetto al numero totale di oggetti presenti nel database. Lo scopo di un RS è quello di predire i rating mancanti per tutte le coppie utente - oggetto.

I recommender system si dividono principalmente in quattro categorie:

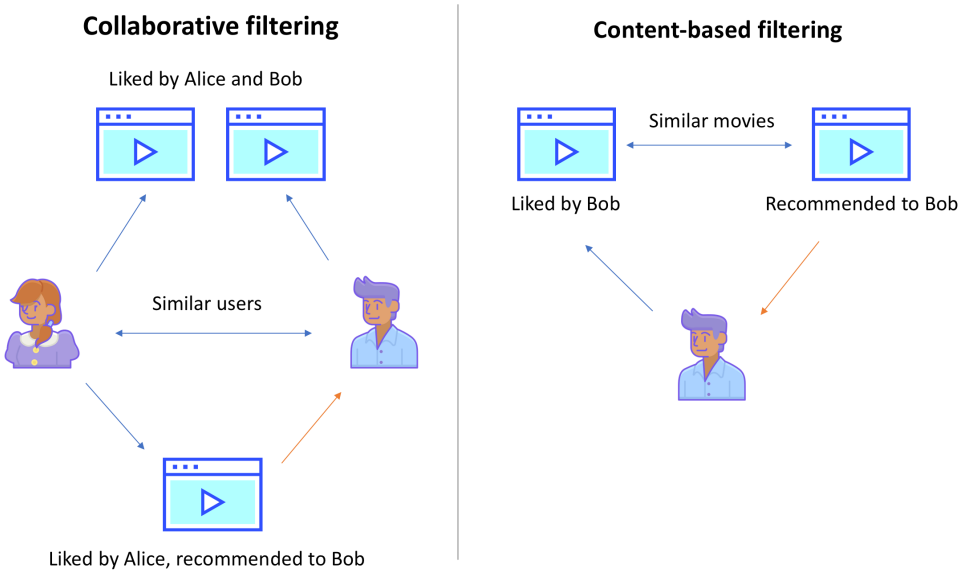

Figura 1: Collaborative filtering vs Content-based
[3]

- collaborative filtering
- content-based
- ibridi
- context-aware

In Figura 1 è schematizzato il modo diverso in cui operano i metodi collaborative filtering rispetto a quelli content-based. Le quattro categorie sono descritte in dettaglio nelle sezioni successive.

2.1 Collaborative filtering recommender system

Collaborative filtering (CF) è 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 rating 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 [4]. 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 [5]; di deep learning si parlerà diffusamente in applicazione ai metodi context-aware ( ).

2.1.1 Vantaggi e svantaggi dell'approccio CF

Vantaggi:

- *Nessuna conoscenza del dominio necessaria:* basandosi solo su interazioni utente - oggetto, il modello può funzionare in domini in cui non c'è nessun contenuto associato agli oggetti [6]. Questo permette di impiegarlo facilmente per realizzare RS multi-domain, che raccomandano oggetti di natura diversa (audio, film, testo, etc.).
- *Serendipity:* il modello ha la capacità di fornire consigli fortuiti, il che significa che può consigliare elementi pertinenti per l'utente senza che il contenuto si trovi nel profilo dell'utente, permettendo così all'utente di scoprire nuovi interessi [6] [7].

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 [8].
- *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 [6].
- *Scalabilità:* questo è un altro problema associato agli algoritmi di raccomandazione perché il tempo di computazione cresce linearmente sia con il numero di utenti, sia con il numero di oggetti. Un sistema di raccomandazione efficiente con un dataset limitato potrebbe non esserlo con un dataset di dimensioni maggiori [6].

2.1.2 Matrix factorization

Gli algoritmi basati su matrix factorization (MF) caratterizzano utenti e oggetti mediante dei vettori di fattori estratti dai pattern sui rating. Questi vettori,

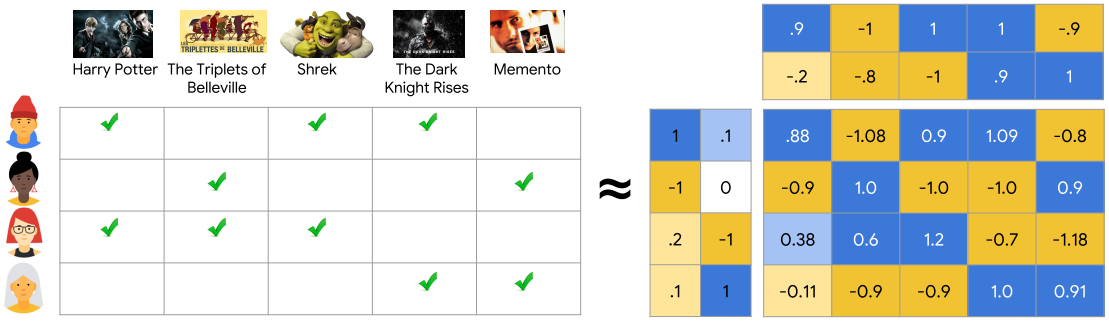

Figura 2: Approssimazione matrice dei rating con matrix factorization [10]

chiamati fattori latenti, sono delle feature nascoste che descrivono utenti e oggetti. 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 d, 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 $q_i \in \mathbb{R}^d$, e ogni utente u con un vettore $p_u \in \mathbb{R}^d$. Per un dato oggetto i, gli elementi di q_i indicano la misura in cui l'oggetto 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 le varie caratteristiche rappresentate dai fattori latenti, positivi o negativi. Il prodotto scalare $q_i^T p_u$ indica l'interesse dell'utente u per le caratteristiche dell'oggetto i [9]. 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 in vettori $q_i, p_u \in \mathbb{R}^d$. 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 rating $A \in \mathbb{R}^{m \times n}$, con m numero di utenti e n numero di oggetti, viene decomposta in due matrici di dimensionalità molto minore:

- Una matrice di fattori latenti per gli utenti $P \in \mathbb{R}^{m \times d}$, in cui la riga i contiene i fattori latenti dell'utente i.

- Una matrice di fattori latenti per gli oggetti $Q \in \mathbb{R}^{n \times d}$, in cui la riga j contiene i fattori latenti dell'oggetto j.

Le due matrici di fattori latenti P e Q sono imparate in modo tale che il prodotto PQ^T sia una buona approssimazione della matrice dei rating A [10].

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 che il processo di raccomandazione si focalizza sui contenuti piuttosto che sulle interazioni utente - oggetto. Nei metodi content-based i rating degli utenti sono combinati con le informazioni disponibili sugli oggetti, per poi essere usati come training data per creare un modello di classificazione o regressione specifico per l'utente. Nei problemi di classificazione si prevede se ad un utente può piacere un oggetto; nei problemi di regressione si prevede il rating dato da un utente ad un oggetto la cui valutazione è ancora sconosciuta [11].

Mentre nei CF la similarità tra due oggetti (o due utenti) è calcolato come la correlazione o la similarità tra i rating 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 nel sistema.

2.2.1 Vantaggi e svantaggi dell'approccio CB

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 [6].

- *Trasparenza:* le spiegazioni su come funziona il sistema di raccomandazione possono essere fornite elencando esplicitamente le caratteristiche del contenuto che hanno causato la presenza di un oggetto nell'elenco delle raccomandazioni [12].

Svantaggi:

- *Feature degli oggetti:* la precisione del modello dipende dall'insieme delle feature che descrivono gli oggetti. Identificare le feature 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 [6].

- *Raccomandazioni multi-domain:* è difficile creare RS multi-domain con un approccio content-based. Questo perché è complicato definire un insieme di feature che valgano per contenuti di natura diversa.

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 [11] 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: (i) usare sistemi di raccomandazione già esistenti, (ii) produrre uno score/ranking unico. Il problema di questo approccio è la complessità della soluzione: il risultato è ottenuto con l'esecuzione di q algoritmi di raccomandazione diversi.

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 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 metodo tende a integrare più strettamente le varie fonti di dati e

non è possibile visualizzare facilmente i singoli componenti come black-box separate.

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

2.4 Context-aware recommender system

La maggior parte degli approcci esistenti per sviluppare sistemi di raccomandazione si concentra sul raccomandare gli oggetti più rilevanti ai singoli utenti senza considerare informazioni aggiuntive come il tempo, il luogo, etc. In altre parole, i sistemi di raccomandazione tradizionalmente si occupano di applicazioni che hanno solo due tipi di entità, utenti ed oggetti, e non li inseriscono in un contesto quando forniscono raccomandazioni. Tuttavia, in molte applicazioni potrebbe non essere sufficiente considerare solo utenti ed oggetti, ma è anche importante incorporare informazioni contestuali nel processo di raccomandazione al fine di consigliare oggetti agli utenti in determinate circostanze. I sistemi di raccomandazione che producono le loro raccomandazioni utilizzando il contesto sono chiamati recommender system context-aware (CARS). Alcuni esempi di contesto sono:

1. *Data e ora:* Dalle informazioni di data e ora è possibile estrarre diverse feature 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 relazioni sociali di un utente possono avere un impatto sul processo di raccomandazione. Per esempio un ragazzo 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 [13] 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 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 l'apprendimento di una funzione che associa le coppie utente-oggetto ai rating. La funzione corrispondente 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 rating. I CARS generalizzano questo metodo utilizzando un approccio multidimensionale in cui la rating function può essere vista come un mapping da una matrice n-dimensionale all'insieme dei rating [2].

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

In questo caso, il risultato è un cubo n-dimensionale al posto che una matrice bidimensionale. Le diverse dimensioni sono denotate come $D_1 \ldots D_n$. Due di queste dimensioni saranno sempre utenti e oggetti, le altre D_i dimensioni corrispondono alle feature del contesto [11]. In Figura 3 è mostrato un esempio di un cubo tridimensionale che memorizza i ratings per $User \times Item \times Location$, 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 per integrare il contesto nei sistemi di raccomandazione [13]:

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 rating). Poi i rating mancanti dai dati selezionati possono essere predetti utilizzando qualsiasi sistema di raccomandazione 2D tradizionale.

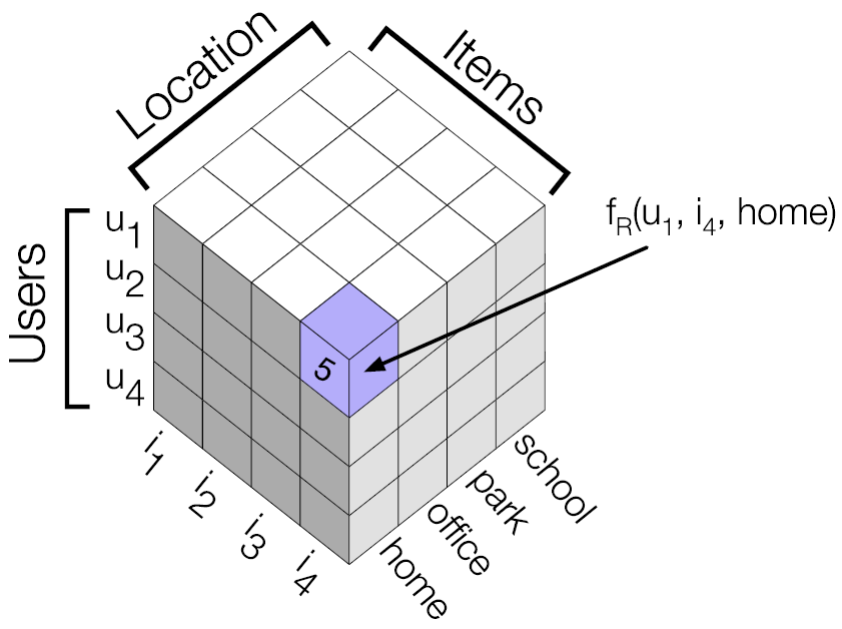

Figura 3: Esempio di un cubo multidimensionale per $User \times Item \times Location$ [2]

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

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) [15] 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 [16].

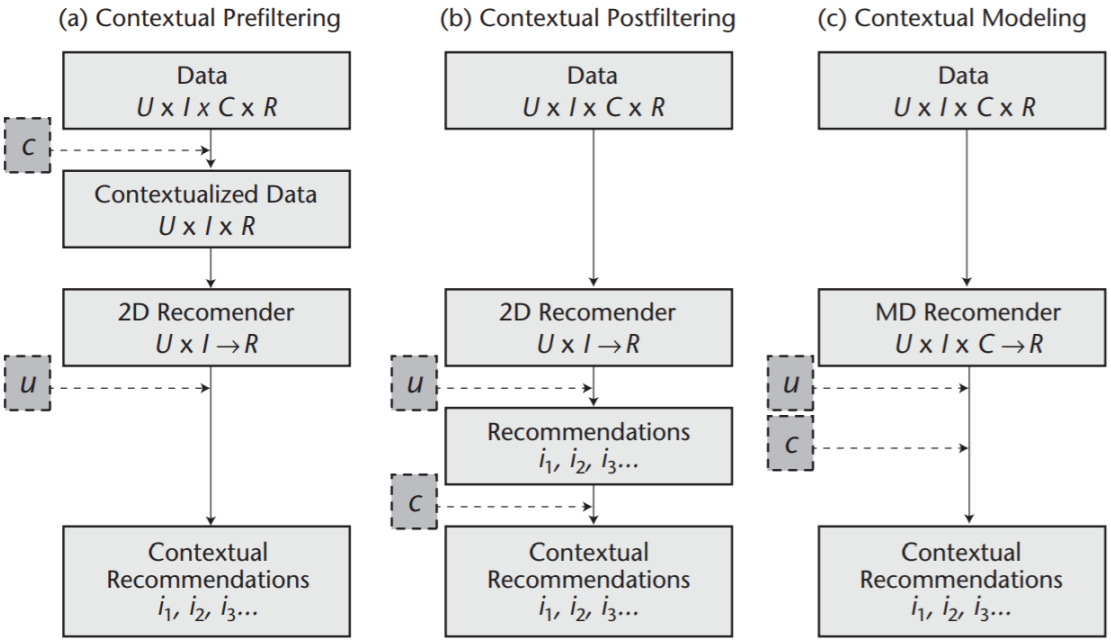

Figura 4: Paradigmi per incorporare il contesto nei sistemi di raccomandazione [14]

2.5.1 Feedback impliciti

I feedback impliciti sono più semplici da collezionare rispetto ai feedback espliciti. Infatti, mentre i feedback espliciti sono ottenuti da valutazioni che l'utente lascia intenzionalmente sugli oggetti (es. valutazione da 1 a 10 di un film); i feedback impliciti sono collezionati automaticamente osservando il comportamento di un utente (es. 1 se l'utente ha guardato un film, 0 altrimenti). 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 [15]:

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 rating 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. Diversamente, se u non ha mai consumato i la preferenza è uguale a 0. Inoltre, 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'obiettivo 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 e 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 funzione 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 modelli, 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 funzione migliore per modellare l'interazione delle feature latenti di utenti e oggetti. Secondo gli autori di [17] il prodotto scalare, che combina la moltiplicazione delle feature 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 tra i loro vettori latenti, o in modo equivalente con il coseno dell'angolo tra i loro vettori latenti. Dalla matrice utenti-oggetti, in Figura 5 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 [18] per imparare le interazioni utente-oggetto. In particolare sono proposti tre modelli basati su deep neural networks:

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

2.6.1 Framework generale

Per permettere un trattamento neurale di collaborative filtering, viene adottata una rappresentazione multi-layer per modellare le interazioni utente-oggetto y_{ui}, in cui l'output di un layer è l'input per il layer successivo. 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

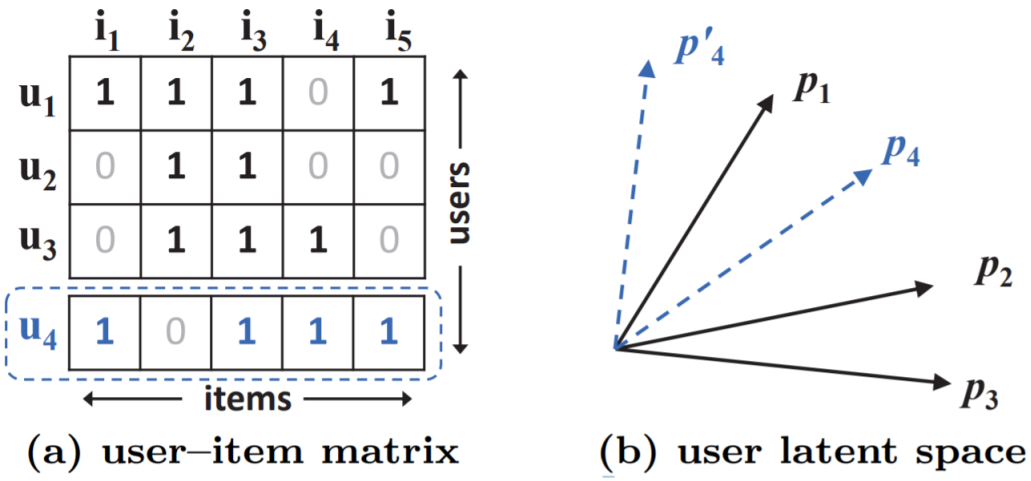

Figura 5: Limitazioni di matrix factorization
[17]

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 vettori binari sparsi. One-hot encoding è un processo che converte una variabile categorica in un vettore di uno e zero. La lunghezza del vettore in questo caso è pari al numero di utenti o oggetti, ed ogni elemento nel vettore rappresenta un utente/oggetto. Tutti gli elementi del vettore sono posti a zero, tranne l'elemento corrispondente all'utente/oggetto corrente che è posto ad uno. 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, i cui livelli sono chiamati neural collaborative filtering layer, e infine al layer di output per calcolare lo score predetto y_{ui}. Questo framework generale è chiamato *neural collaborative filtering* (NCF), ed è mostrato in Figura 6.

2.6.2 Multi-layer perceptron

In questa istanza di NCF, sopra al layer di embedding sono aggiunti dei layer nascosti, in modo da usare un multi-layer perceptron (MLP) standard per imparare le interazioni tra le feature latenti di utenti e oggetti. In questo caso, il modello impara una funzione non-lineare per le interazioni tra p_u e q_i, al posto che imparare un prodotto elemento per elemento come GMF.

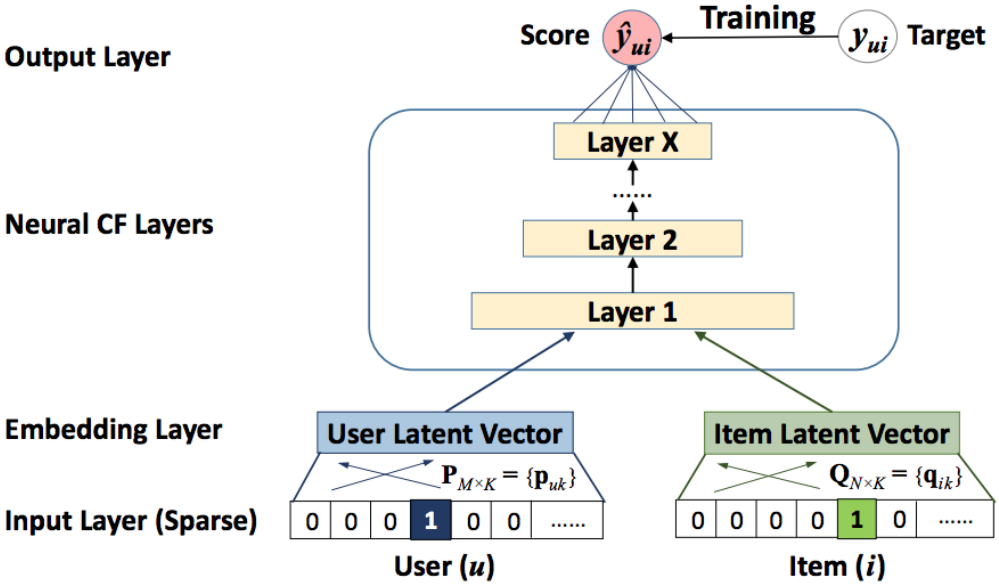

Figura 6: multi-layer perceptron
[17]

2.6.3 Generalized matrix factorization

In [17] viene mostrato come la famiglia di metodi basata su matrix factorization possa essere approssimata utilizzando una rete neurale. In questa rete il layer di input e il layer di embedding sono gli stessi del modello MLP, la differenza è nei neural collaborative filtering layer. 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 chiamata generalized matrix factorization (GMF).

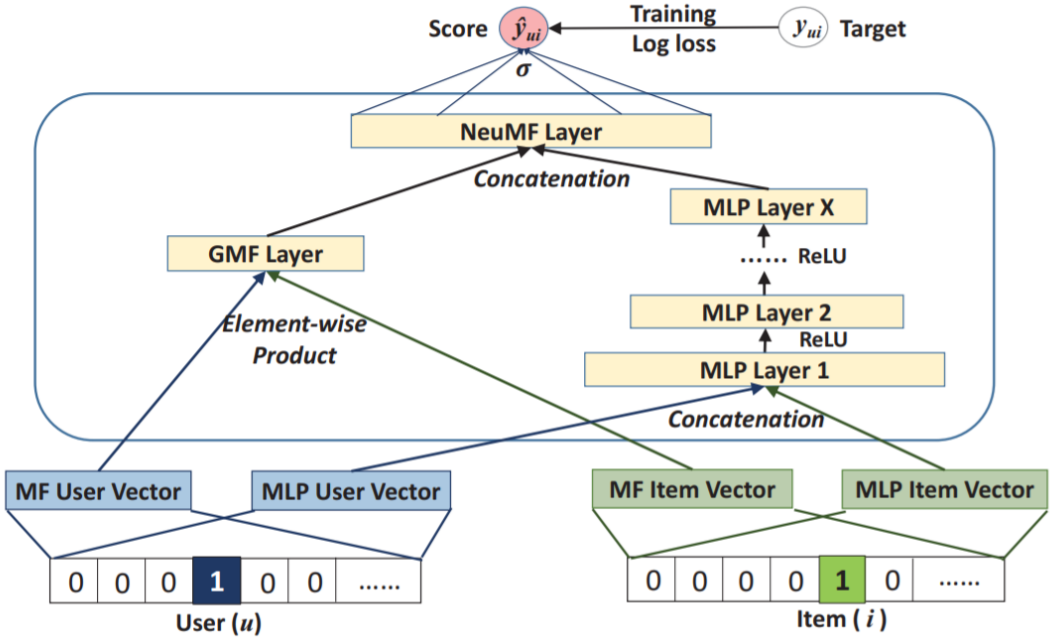

Figura 7: Neural matrix factorization
[17]

2.6.4 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 imparare le interazioni delle feature 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 approccio però 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. A sinistra è rappresentata la rete GMF, a destra il MLP.

2.7 Raccomandazioni context-aware con modelli di deep learning

I sistemi di raccomandazione context-aware tradizionali usano principalmente un insieme selezionato di informazioni contestuali. Il contesto specifico descrive le circostanze in cui le informazioni sono state raccolte, quali ad esempio 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 [19]. Infatti un dataset con molte feature porta naturalmente ad uno spazio multidimensionale e quindi a sparsità. Questo fenomeno è conosciuto come curse of dimensionality [20].

Tuttavia, i CARS tradizionali 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 l'insieme di feature 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 [19]. La limitazione sul numero di feature contestuali potrebbe essere un problema in quegli ambienti in cui il contesto è complesso e dinamico. Ad esempio, sfruttando i numerosi sensori presenti sugli smartphone come accelerometro, campo magnetico, GPS, e sensore di luminosità, possono essere raccolte 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), mentre con il GPS si può inferire la posizione (es. a casa, al lavoro, all'aperto).

Per risolvere i problemi legati all'utilizzo del contesto multidimensionale, viene proposto in letteratura di ridurre la dimensionalità del contesto con autoencoder o PCA [21] [22], di costruire una rappresentazione gerarchica del contesto [23], e di usare dei modelli di deep learning in grado di supportare molte feature di contesto[19].

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 possono essere correlati tra loro. Si può usare un autoencoder (AE) per scoprire le correlazioni tra feature differenti ed estrarre una rappresentazione a bassa dimensionalità del contesto [21]. 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 dalla rappresentazione latente (decoder) [24]. Limitando il numero di unità nei layer nascosti di un AE, la rete è costretta a

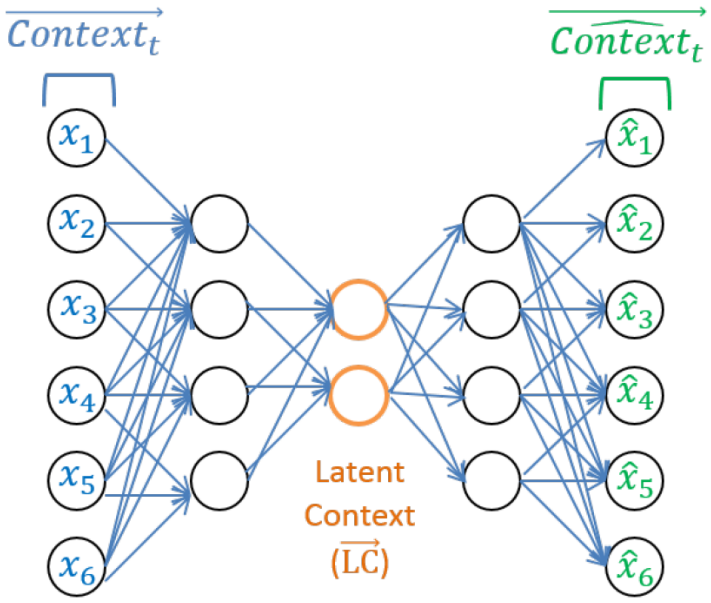

Figura 8: Esempio struttura di un autoencoder [23]

imparare una rappresentazione compressa dell'input. In Figura 8 è rappresentata la struttura di un autoencoder che ricostruisce il vettore di contesto dato in input $\overrightarrow{Context_t}$. Il contesto ottenuto può essere usato al posto delle feature di contesto estratte dai dati raw, e può portare ad un miglioramento nelle raccomandazioni prodotte dal recommender system.

L'algoritmo 1 descrive come utilizzare un AE per estrarre gli attributi latenti del contesto. L'input per l'algoritmo è un training set $S = \{s_1, s_2, \ldots, s_n\}$, in cui ogni campione è $r-$dimensionale e contiene le feature di contesto estratte dai dati raw, f è la funzione di attivazione dell'autoencoder. L'output è O, l'insieme delle feature latenti estratte da S. L'algoritmo inizia normalizzando il dataset S (riga 1); la normalizzazione include la conversione delle variabili categoriche in valori binari e la normalizzazione delle variabili numeriche. Il risultato è il dataset normalizzato S'. Poi viene eseguito il training di un AE sul training set normalizzato (riga 2). A training terminato si ricava la matrice W, in cui w_{ij} è il peso dell'arco che connette il neurone di input j-esimo con il neurone nascosto i-esimo (riga 3). Dopo aver inizializzato O (riga 4), si itera su ogni sample di S' (riga 5), moltiplicandolo per la trasposta di W (riga 6), e applicando la funzione di attivazione f su ogni elemento del vettore risultato (riga 7). Questo vettore è concatenato a O che a ciclo terminato viene ritornato (riga 9).

Le righe 10, 11 e 12 descrivono come estrarre il contesto latente da un nuovo

campione t. Per prima cosa t viene normalizzato esattamente come nella riga 1, poi usando la matrice W e la funzione di attivazione f si ottiene il contesto latente res.

Algoritmo 1 Estrarre il contesto latente usando un auto-encoder [21]

Input: S - training set, n latent context size, f - activation function.
Output: O - latent context attributes of the training set; extraction function for a new sample t

1: $S' \leftarrow$ Normalize all samples in S
2: $AE \leftarrow$ Train an auto-encoder (n, f) on the normalized training dataset S'
3: $W \leftarrow$ Retrieve weight matrix from AE
4: $O \leftarrow \varnothing$
5: **for all** $s' \in S'$ **do**
6: $\quad o \leftarrow s'W^T$
7: $\quad O \leftarrow O \cup$ activate f on each element in o
8: **end for**
9: Return O
Extraction for a new data sample t:
10: $t' \leftarrow$ normalize t
11: $res \leftarrow$ activate f on each element in $t'W^T$
12: **return** res

2.7.2 Rappresentazione gerarchica del contesto

Il metodo per ridurre la dimensionalità del contesto descritto nella sottosezione 2.7.1 modella le informazioni contestuali in vettori a dimensionalità minore, ignorando però la struttura delle variabili latenti di contesto. In particolare, questi metodi, mentre riducono la dimensionalità dello spazio contestuale, non tengono conto della struttura delle variabili latenti di contesto e delle relazioni semantiche tra queste variabili. In [23] viene proposta una nuova rappresentazione strutturata del contesto latente organizzata in maniera gerarchica che include gruppi di contesti latenti simili chiamati *situazioni contestuali*. Per esempio, se un vettore latente del contesto rappresenta le variabili di contesto "mattina", "rumoroso", "fermo" e la posizione è "università", allora queste variabili di contesto collettivamente rappresentano la situazione di uno studente che segue una lezione in università. Le situazioni contestuali possono poi essere organizzate in una struttura gerarchica aggregando i vettori latenti del contesto in una rappresentazione ad alto livello. Per esempio, "seguire una lezione in università" e "pranzare in università" possono essere aggregati in una situazione contestuale più ad alto livello come "essere situati in università".

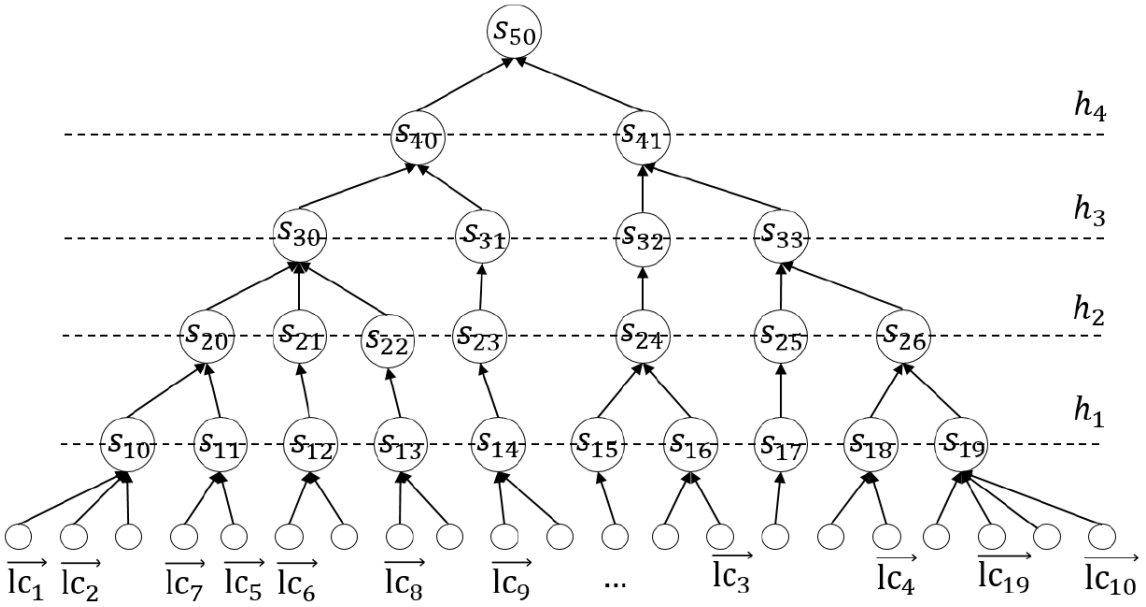

Figura 9: Gerarchia delle situazioni contestuali
[23]

Il processo di costruzione del modello gerarchico è svolto raggruppando l'insieme di tutti i vettori latenti non strutturati del contesto in un insieme finito di cluster, in cui ogni cluster rappresenta una situazione contestuale. Si applica agglomerative hierarchical clustering (AHC) [25] ai vettori latenti per stimare in automatico il numero di situazioni contestuali S che sono rappresentate come cluster. Poi si applica l'algoritmo k-means per raggruppare vettori simili nella stessa situazione contestuale. Il clustering gerarchico produrrà un albero come quello in Figura 9. Per ogni vettore del contesto latente $\overrightarrow{lc_j}$, i quali sono sempre nodi foglia dell'albero, esiste un nodo $s_{h_t i}$ che è un antenato del nodo $\overrightarrow{lc_j}$. Il nodo $s_{h_t i}$ rappresenta una situazione contestuale simile per il vettore $\overrightarrow{lc_j}$ al livello h_t della gerarchia. Il contesto gerarchico per un vettore $\overrightarrow{lc_j}$ è il percorso dalla sua foglia fino alla radice dell'albero. Per esempio il contesto gerarchico per il vettore di contesto latente $\overrightarrow{lc_{19}}$ in Figura 9, è $\overrightarrow{hlc_{19}} = [s_{19}, s_{26}, s_{33}, s_{41}]$.

2.7.3 Modelli CARS deep learning

In [19] viene proposto di estendere il modello NeuMF [17] descritto nella sezione 2.6 aggiungendo un nuovo componente di informazioni contestuali: un vettore di contesto denotato come "Context (c)". Il vettore del contesto c è concatenato ai vettori di embedding degli utenti u e degli oggetti i per imparare una nuova

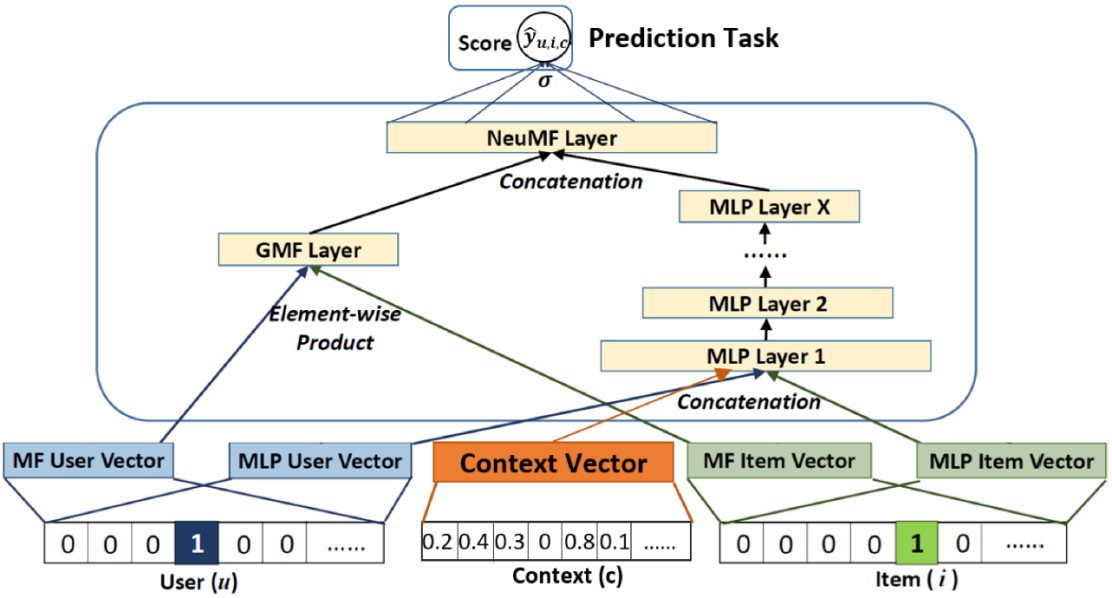

Figura 10: Estensione di NeuMF con features di contesto [19]

funzione tra i tre componenti (utenti, oggetti, e contesto). In questo modo, la dimensione del contesto viene considerata all'interno del framework neurale e la rete apprende automaticamente la sua influenza sul valore di output. Il contesto è concatenato solo agli embedding del multi-layer perceptron, mentre la parte della rete denominata come generalized matrix factorization e i suoi embedding rimangono invariati. È importante notare che il vettore di contesto può avere un alta dimensionalità, infatti questo modello non soffre del problema di curse of dimensionality che affligge i CARS tradizionali.

Le informazioni contestuali sono modellate come un insieme di feature di contesto esplicite, latenti non strutturate o latenti strutturate (gerarchiche). Questo dà luogo a tre diverse estensioni di NeuMF:

1. *Explicit context-aware model (ECAM)*: tutto il contesto disponibile viene incorporato nel modello.

2. *Unstructured context-aware model (UCAM)*: il contesto viene processato da un autoencoder [21] come descritto nella sottosezione 2.7.1 prima di essere incorporato nel modello.

3. *Hierarchical context-aware model (HCAM)*: il contesto viene organizzato in un albero gerarchico [23] come descritto nella sottosezione 2.7.2 prima di essere incorporato nel modello.

In Figura 10 è schematizzato il modello NeuMF esteso per supportare il vettore del contesto. A sinistra è rappresentata la rete GMF, a destra il MLP con il vettore del contesto.

2.8 Problemi RS client-server

[Forse è da mettere nell'introduzione?] In questo capitolo viene descritta l'implementazione di un algoritmo di raccomandazione pensato per dispositivi mobili e pervasivi inserito in un ambiente totalmente distribuito come quello delle reti opportunistiche. In queste reti, di solito, un dispositivo mobile dovrebbe essere in grado di condividere informazioni con altri dispositivi in prossimità in modo da scoprire contenuti utili per il suo proprietario. Un sistema di raccomandazione può essere utile per filtrare i contenuti più adatti ad un utente tra quelli scoperti nelle vicinanze. Non è però semplice estendere le soluzioni esistenti per adattarle alla limitazioni imposte da questo nuovo scenario. I recommender system tradizionali si appoggiano su un modello client/server centralizzato, in cui il sistema di raccomandazione esegue sul server, e processa le richieste in arrivo dai client che possono essere dispositivi mobili o fissi. Oltre a questo, il task di raccomandazione per un RS pervasivo è diverso rispetto a quello di un RS client/server. Tipicamente un RS deve imparare a prevedere i rating sulle coppie utente-oggetto per poter riempire i valori mancanti nella matrice dei rating. Questa matrice può essere vista come la conoscenza globale del sistema su tutti gli utenti e gli oggetti disponibili, e sulle valutazioni che gli utenti hanno dato agli oggetti. Nel caso dei RS collaborative-filtering il recommandation engine impara un modello singolo che sarà usato per fare raccomandazione su tutti gli utenti del sistema, mentre nel caso dei RS content-based sarà istanziato un modello per ogni utente che comunque ha una conoscienza globale di tutti gli oggetti del sistema. Al contrario nelle reti opportunistiche ogni dispositivo potrebbe essere a conoscenza solo di una piccola parte dell'informazione globale. Questa conoscenza è inizialmente circoscritta alle sole informazioni sull'utente locale, per poi allargarsi con lo scambio di informazioni tramite comunicazione device-to-device (D2D) con altri utenti o dispositivi di altra natura. Di conseguenza, un RS implementato in un ambiente distribuito impara un modello di raccomandazione basato unicamente sulle informazioni raccolte dal suo utente locale.

Un'altra caratteristica importante per un sistema di raccomandazione per dispositivi mobili è il supporto per il contesto utente. Sfruttando i numerosi sensori presenti sui dispositivi degli utenti è possibile generare un contesto ad alta dimensionalità che caratterizza in modo accurato la situazione dell'utente e permette raccomandazioni più accurate. Al contesto fisico estratto dai sensori o altre fonti, si può aggiungere il contesto sociale che descrive quali tipologie di persone l'utente

ha intorno a se (amici, colleghi, sconosciuti, etc.) e da che tipo di persone ha ricevuto le raccomandazioni in passato, dando più peso ad esempio a raccomandazioni ricevute da amici e conoscenti piuttosto che da sconosciuti.

Da quanto detto prima si può dedurre che i problemi principali nell'implementare un sistema di raccomandazione per sistemi mobili e pervasivi sono principalmente due:

1. Non si conosce a priori il numero di utenti e oggetti presenti nel sistema perchè l'utente ne scopre di nuovi gradualmente tramite interazione D2D.
2. È difficile integrare le numerose informazioni di contesto fisico generate dai dispositivi mobili e le informazioni di contesto sociale che caratterizzano la situazione dell'utente.

2.8.1 Dettaglio sullo stato dell'arte

Gli algoritmi di raccomandazione stato dell'arte descritti nel Capitolo 2 hanno alcune limitazioni che rendono difficile l'implementazione senza alcuna modifica nell'ambiente di riferimento.

ALS
Alternating least square (descritto in ...) è un algoritmo di matrix factorization per feedback impliciti. L'input di ALS è una matrice R di dimensione $u \times i$ con u numero di utenti, e i numero di oggetti. Un elemento r_{ui} della matrice R indica quante volte l'utente u ha consumato l'oggetto i. In ambiente mobile inizialmente la matrice R è vuota perché l'utente non ha espresso nessuna valutazione e non ha incontrato altri utenti. Ogni volta che un nuovo utente o un nuovo oggetto viene scoperto si aggiunge una nuova riga/colonna alla matrice R. Il primo problema di questo algoritmo, e più in generale degli algoritmi di matrix factorization è l'aggiunta di un nuovo utente/oggetto che comporta un cambiamento nella dimensione della matrice R. L'algoritmo deve essere addestrato nuovamente da zero sulla nuova matrice R per poter raccomandare i nuovi oggetti e tenere conto delle valutazioni dei nuovi utenti. Il secondo problema di MF è l'aggiunta del contesto. Come detto in [sezione context-aware] ogni feature di contesto è una dimensione aggiunta alla matrice dei rating R. Con l'aumento delle dimensioni, lo spazio dimensionale aumenta in modo esponenziale [26] mentre il numero dei punti nello spazio rimane sempre lo stesso. Questa crescita esponenziale porta un'alta sparsità dei dati, che impedisce al modello di generalizzare correttamente. La Figura 11, mostra graficamente il problema. Nell'immagine a) lo spazio 1D è diviso in 4 regioni. In b) aggiungendo una dimensione lo spazio cresce esponenzialmente a 16 regioni. In c) aggiungendo una terza dimensione lo spazio è diviso in 64 regioni ma il numero di punti in ogni regione rispetto ad a) è molto minore.

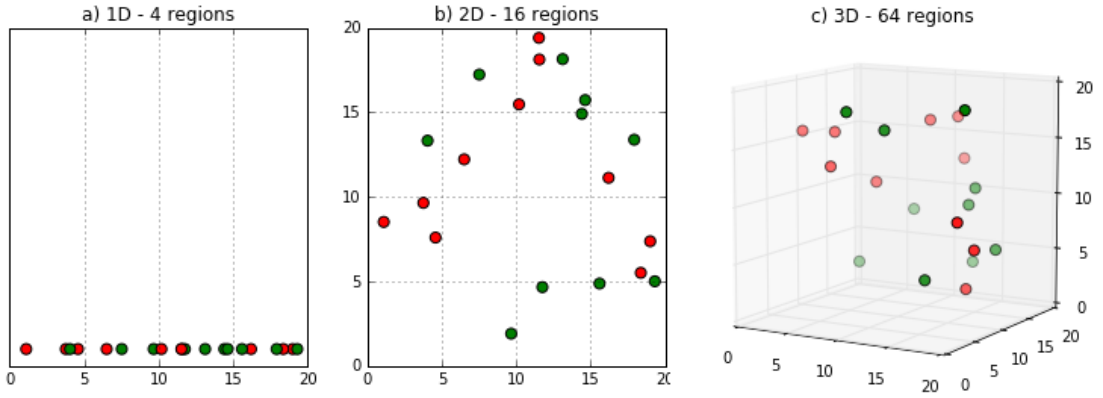

Figura 11: Curse of dimensionality

NeuMF

Neural matrix factorization (descritto...) implementa un RS collaborative-filtering con una deep neural network. L'input della rete è un record formato da `[user_id, item_id, rating]`. Per essere correttamente processati dalla dalla rete neurale `user_id` e `item_id` sono convertiti da numeri interi in vettori binari con one-hot encoding. Questi vettori hanno lunghezza pari al numero di user/item e hanno valore 1 solo in corrispondenza dell'elemento numero `user_id`/`item_id`, il resto degli elementi ha valore 0. La dimensione dell'input della rete e di conseguenza la dimensione dell'input del layer di embedding è da definire a livello di compilazione della rete prima di eseguire il training. Una volta terminato il training, la rete è in grado di predirre il rating degli stessi utenti/oggetti già presenti durante il training. Questo perché l'input deve avere la stessa dimensione di quello definito a livello di compilazione, e quindi il numero di utenti e oggetti è lostesso definito a livello di compilazione. Come per ALS quindi aggiungere un nuovo utente od un nuovo oggetto significa dover compilare di nuovo il modello ed eseguire di nuovo il training da zero. Questo modello inoltre, come presentato nel paper originale [17] non ha nessun supporto per le feature di contesto.

Context-aware NeuMF

Context-aware neural matrix factorization (descritto ...) è un estensione di NeuMF che supporta le feature di contesto. Questo modello essendo basato su reti neurali, è meno soggetto di matrix factorization al problema della curse of dimensionality. L'input della rete è un record formato da `[user_id, item_id, context, rating]`, in cui il vettore context può avere un'alta dimensionalità. Risolve quindi il problema dell'integrare le numerose feature di contesto fisico estratte dai sensori degli smartphone e le features sociali che descrivono il contesto sociale dell'utente.

Rimangono le stesse limitazioni di NeuMF sull'input, dato che gli ID di utenti e oggetti sono dati in input come vettori in one-hot encoding con dimensione fissata.

Capitolo 3

RS per dispositivi mobili e pervasivi

3.1 Introduzione

In questo capitolo è descritto un sistema di raccomandazione context-aware per sistemi mobili e pervasivi deep learning che è in grado di generare raccomandazioni direttamente dal device dell'utente. Il RS è inserito all'interno di un architettura più complessa che permette sempre da device utente di raccogliere e processare le feature di contesto sociale e fisico. L'architettura ad alto livello è composta da tre componenti:

1. *Sensing manager.* Il primo componente interagisce con il sistema operativo per raccogliere continuamente dati raw di contesto (come GPS e accelerometro), e dati che rappresentano informazioni più astratte relative allo stato del dispositivo come lo stato del display e il livello di batteria.

2. *Context modeling.* I dati raw dei sensori devono essere processati ulteriormente per inferire una rappresentazione più astratta del contesto dell'utente. A questo scopo, il componente context modeling (CM) raccoglie periodicamente gli ultimi dati disponibili del SM. Queste osservazioni sono processate per estrarre feature numeriche e categoriche che caratterizzano il contesto dell'utente locale. Esempi di queste feature sono ad esempio le statistiche ottenute dai dati dei sensori fisici o la posizione dell'utente. Di queste feature eterogenee viene fatto l'encoding e vengono combinate in un singolo vettore di feature che rappresenta una fotografia del contesto corrente di un utente. L'insieme di feature che compongono il contesto fisico e sociale dell'utente sono descritte nella sezione 3.3.

3. *Sistema di raccomandazione.* Le feature di contesto c sono concatenate alle feature degli utenti u e degli oggetti i in un unico vettore. Questo vettore è dato in input ha una rete neurale che restituisce valore 1, se per l'utente con feature u, l'oggetto con feature i è rilevante nel contesto c, 0 altrimenti. L'input, la struttura e l'output della rete sono descritti nella sezione sezione 3.2.

3.2 Sistema di raccomandazione

In questa sezione è descritto il sistema di raccomandazione vero e proprio. Nella prima parte è descritto l'input, e in che modo si differenzia dai sistemi di raccomandazione collaborative filtering e content-base. Nella seconda parte invece è descritta nel dettaglio la struttura della rete neurale che genera le raccomandazioni context-aware.

3.2.1 Input

Solitamente l'input dei modelli collaborative filtering context-aware è composto da tuple `(user_ID, item_ID, rating, context)`, in cui `user_ID` è l'utente che ha valutato l'oggetto `item_ID` con una valutazione `rating` in una situazione descritta dal contesto `context`. Al posto che identificare l'oggetto con un valore numerico intero `item_ID`, si possono usare delle feature che caratterizzano l'oggetto, esattamente nello stesso modo in cui sono solitamente descritti gli oggetti nei sistemi di raccomandazione content-based. Ad esempio, se si sta sviluppando un RS per consigliare ristoranti agli utenti, si può sostituire il valore `item_ID` che identifica il ristorante con delle feature che lo caratterizzano nel dettaglio come il tipo di cibo servito, il prezzo medio, l'atmosfera, se ha sedute all'aperto, etc. Allo stesso modo si può sostituire il valore `user_ID` con delle feature che descrivono l'utente. Queste possono essere feature molto generiche come età o sesso, o feature specifiche per l'ambiente in cui il RS è implementato. Tornando all'esempio dei ristoranti, si potrebbe chiedere all'utente (es. attraverso un'applicazione mobile) quanto è disposto a spendere per mangiare fuori e il tipo di cucina preferita. A feature di utente e oggetto si aggiungono le feature del contesto fisico e sociale generate dal modulo di Context modeling. Un'istanza di rating per il modello proposto in questa tesi è quindi una tupla `(user_features, item_features, physical_context, social_context)`.

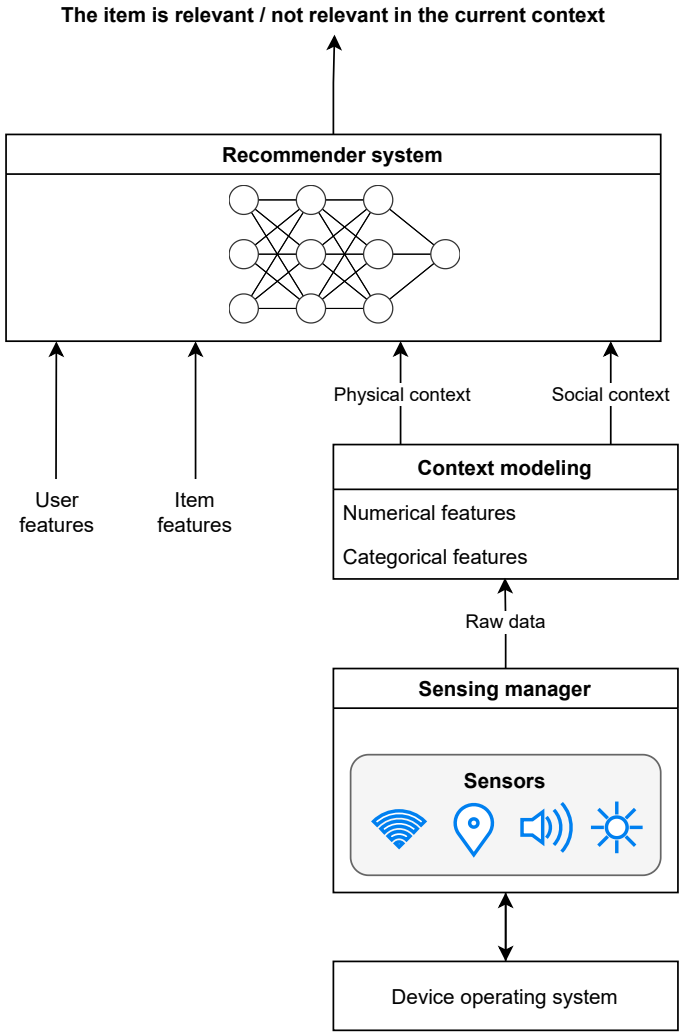

Figura 12: Architettura ad alto livello del sistema di raccomandazione

3.2.2 Struttura della rete neurale

Il vettore di feature appena descritto è dato in input ad una rete neurale che deve prevedere se l'utente caratterizzato da `user_feature` è interessato all'oggetto caratterizzato da `item_feature` nei contesti fisici e sociali `physical_context` e `social_context`. Si tratta quindi di un problema di classificazione binaria. Nei problemi di classificazione, l'obbiettivo è prevedere il valore di una variabile che può assumere diversi valori discreti. I problemi di classificazione in cui una variabile può assumere solo due valori possibili (come 0 o 1) sono chiamati problemi di classificazione binaria [27].

Layer e neuroni La rete rientra nella categoria feed-forward fully connected. Una rete feed-forward non contiene cicli nel suo grafo [28], fully connected indica che ogni neurone del layer i è connesso a tutti i neuroni del layer $i + 1$. La rete ha un layer di input, un layer di output e l layer nascosti. Il layer di input ha un numero di neuroni pari alle feature in ingresso (sommando user, item e context feature), il layer di output ha sempre un neurone, mentre il numero di neuroni nei layer nascosti l, e il numero di layer nascosti è calcolato facendo il tuning della rete tramite grid search, scegliendo la combinazione che ottiene i risultati migliori. In questa tesi è stato utilizzato lo stesso numero di neuroni in ogni layer nascosto, ma si può ad esempio adottare un design a torre in cui i layer più profondi contengono meno neuroni rispetto ai layer meno profondi.

Funzione di attivazione Una funzione di attivazione di un neurone definisce l'output di quel neurone in base all'insieme dei suoi input. Come funzione di attivazione del layer di input e dei layer nascosti ho scelto la funzione rectified linear unit (ReLU) definita come $f(x) = max\{0, x\}$. La funzione ReLU è consigliata per la maggior parte delle reti feed-forward [28] e ha diversi vantaggi rispetto a funzioni di attivazione come sigmoide e tanh: è più plausibile biologicamente, non viene saturata (a differenza di tanh e sigmoide che hanno un output massimo uguale a 1), e incoraggiando l'attivazione sparsa dei neuroni rende più difficile che si verifichi l'overfitting del modello durante il training [29]. Come funzione di attivazione del layer di output ho scelto la funzione sigmoide definita come

$$f(x) = \frac{1}{1 + e^{-x}}$$

che limita l'output della rete a valori tra 0 e 1, ed è quindi adatta per problemi di classificazione binaria [30].

Funzione di loss Una funzione di loss è una misura dell'errore tra il valore previsto dal modello e il valore effettivo. Come funzione di loss la scelta più

comune per un classificatore binario è la funzione binary cross-entropy / log loss, definita come

$$C = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

dove y è il valore reale del feedback di un utente su un oggetto (0 oppure 1), $p(y)$ è la probabilità predetta dalla rete che y abbia valore 1, e $1 - p(y_i)$ è la probabilità che y abbia valore 0 [31].

Ottimizzatore Un ottimizzatore è un algoritmo che modifica i pesi del modello in modo da minimizzare la funzione di loss e rendere le previsioni più corrette possibile. Come ottimizzatore ho scelto Adam, nel paper in cui viene introdotto viene dimostrato empiricamente di essere generalmente migliore rispetto ad altri algoritmi di ottimizzazione stocastici e di risolvere in modo efficiente problemi di deep learning [32]. Adam ha diversi parametri configurabili, il più importante è il learning rate (chiamato α in Adam) che viene deciso tramite grid search, gli altri parametri ($\beta_1, \beta_2, \varepsilon$) sono lasciati al valore di default della libreria Keras.[1].

Epoche e batch size Epoche e batch size sono due parametri molto importanti da ottimizzare, il numero di epoche e la dimensione della batch size ideali sono trovate tramite grid search nel Capitolo 5. La batch size è il numero di campioni processati prima di aggiornare i parametri del modello. Il numero di epoche indica quante volte viene presentato alla rete il training set.

In Figura 13 è rappresentata la struttura della rete neurale. In questo caso il modello ha due layer nascosti ed ogni layer contiene 10 neuroni tranne il layer di output che contiene un solo neurone.

3.3 Informazioni di contesto

In questa sezione sono descritte le informazioni di contesto fisico e sociale che vengono date in input al sistema di raccomandazione mobile. Tutte le feature sono raccolte e processate direttamente sul device dell'utente.

3.3.1 Contesto fisico

Il contesto fisico è composto da tutte quelle informazioni rilevanti che possono essere utilizzate per caratterizzare la situazione di un utente. Le feature del contesto

[1] `https://keras.io/api/optimizers/adam/`

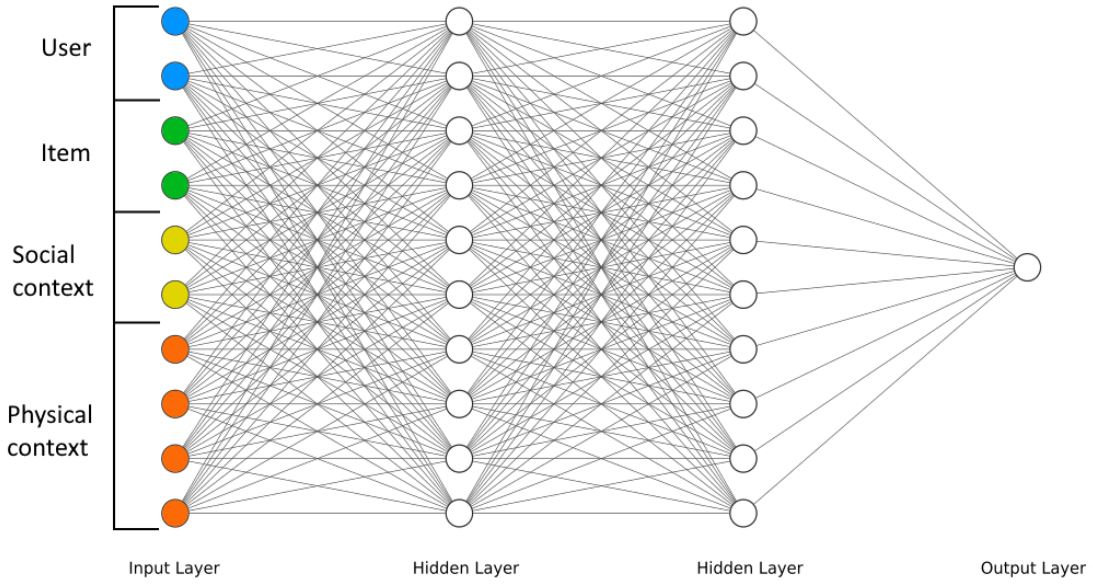

Figura 13: Schema di nome modello

fisico sono ricavate dai sensori fisici dello smartphone di un utente (es. attività utente dall'accelerometro) e dal sistema operativo del telefono (es. stato display e livelo batteria). A queste feature si vanno a integrare informazioni esterne come il meteo, la data e l'ora. Più nel dettaglio il contesto utente è caratterizzato dalle seguenti informazioni:

Posizione Informazioni relative alla posizione geografica che includono latitudine, longitudine, precisione della posizione e direzione del movimento. La posizione geografica può essere usata per capire il luogo in cui si trova l'utente (a casa, al lavoro, etc.) o per raccomandare punti di interesse nelle vicinanze.

Movimento utente Il movimento dell'utente include sia le attività svolte a piedi (correre e camminare), sia il movimento su un mezzo di trasporto (veicolo generico o bicicletta).

Audio Informazioni relative alla configurazione audio dello smartphone, incluse la modalità audio (suono, vibrazione, silenzioso), il volume delle notifiche, e lo stato dell'altoparlante (acceso o spento). Anche l'audio può migliorare il riconoscimento del contesto, per esempio durante una riunione la modalità audio potrebbe essere impostata su silenzioso e l'altoparlante spento.

Batteria Informazioni relative alla batteria del telefono che includono il livello di carica e se la batteria si sta ricaricando.

Display Stato dello schermo dello smartphone (acceso o spento), e orientamento dello schermo (verticale od orizzontale).

Dati dei sensori fisici che includono sensori ambientali (es. temperatura dell'ambiente e luce), sensori di movimento (es. accelerometro e giroscopio) e sensori di posizione (es. rotazione e prossimità).

Celle di rete Lista delle celle di rete mobile rilevate dal dispositivo. Per ogni cella si identifica il tipo di tecnologia (es. GSM o LTE), l'ID della cella, e la forza del segnale. La rete mobile può migliorare l'identificazione della posizione dell'utente.

Wi-Fi Lista di tutti gli access point Wi-Fi disponibili in prossimità, e se l'utente è connesso ad uno di essi.

Meteo Informazioni relative alle condizioni meteo che includono il tempo in atto (es. nuvoloso, piovoso, soleggiato), la temperatura, l'umidità e la velocità del vento.

Data e ora Dalla data si possono generare feature come il giorno della settimana, la stagione, comprendere se è il fine settimana o un periodo di vacanza, etc. Dall'orario invece si può capire il momento della giornata (mattina, pomeriggio, sera, notte).

3.3.2 Contesto sociale

Il contesto sociale si riferisce all'insieme di persone con cui l'utente ha interazioni sociali durante la vita giornaliera, come lavorare con i colleghi o chattare con gli amici. É stato provato in letteratura che esiste una forte correlazione tra le attività umane e i dati sociali. Questo implica che modellare una rete specifica per l'utente di relazioni sociali può contribuire a sottolineare le differenze tra i vari contesti in cui è coinvolto.

Ego Network

Una ego network è una rete sociale composta da un individuo chiamato ego, e dalle persone con cui l'ego ha un collegamento sociale, chiamati alter. I legami sociali

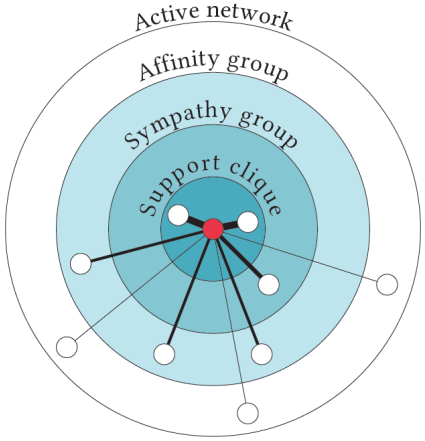

Figura 14: Ego network

in una ego network non hanno tutti la stessa importanza. Ogni individuo ha solo pochi collegamenti forti e molti collegamenti deboli, dovuti alla capacità umana di gestire un numero limitato di relazioni sociali. Una rappresentazione della ego network è mostrata in Figura 14: l'ego è il punto rosso al centro dei quattro cerchi concentrici chiamati layer in cui gli alter sono distribuiti in base alla forza del legame sociale con l'ego. Il cerchio più interno (support clique) è il layer più piccolo, e contiene solo pochi alter che rappresentato le relazioni sociali più forti con l'ego. Il secondo layer (sympathy group) contiene le persone che possono essere considerati gli amici più cari. Il terzo cerchio (affinity group) è composto da amici e membri della famiglia meno vicini, mentre l'ultimo layer include persone con cui l'individuo ha interazioni sociali occasionali.

Modellare il contesto sociale dell'utente

Per modellare il contesto sociale di un utente in ambiente mobile, si caratterizzano le interazioni sociali usando le seguenti sorgenti di dati: (i) chiamate telefoniche e log degli SMS, (ii) dati di prossimità, e (iii) attività svolte dall'utente sugli online social networks (OSN).

Il primo step per costruire l'ego network di un individuo è stimare la forza dei legami sociali con i suoi alter. Un buon indicatore della forza delle relazioni sociale tra due persone è data dal numero di interazioni che le due persone hanno avuto in passato. Basandosi su questa considerazione, per modellare la forza dei legami sociali dell'utente online, sono presi in considerazione diverse attività svolte dall'utente su OSN, inclusi commenti, reazioni (come "mi piace") e persone menzionate. Formalmente, la forza dei legami sociali virtuali tra l'ego e ed uno dei sui alter a,

$\omega_{osn}(e, a)$ è calcolata nel modo seguente:

$$\omega_{osn}(e, a) = \sum_{v \in V} I_S(e, a) \tag{2}$$

dove V è l'insieme delle sorgenti di dati degli OSN nominate prima, e la funzione $I_S(e, a)$ calcola il numero di interazioni tra e ed a per una data sorgente di dati.

Per caratterizzare i link sociali fisici di un utente si calcola il numero di interazioni con altre persone basandosi su telefonate, SMS e contatti faccia a faccia inferiti usando tecnologie wireless disponibili sugli smartphone. In particolare sono considerate il Bluetooth (BT) e il Wi-Fi Direct (WFD) per scoprire persone che sono abbastanza vicine da aver un interazione con l'utente locale. Sono filtrati i dispositivi che non si trovano in prossimità dell'utente, e sono selezionati solo i dispositivi personali dell'utente, in modo tale da non considerare stampanti smart TV etc. In modo simile ai link sociali virtuali, si definisce la forza dei legami fisici sociali tra l'ego e e un alter a , $\omega_{phy}(e, a)$ come il numero delle loro interazioni tramite telefonate, SMS, e prossimità fisica come segue:

$$\omega_{phy}(e, a) = \sum_{p \in P} I_p(e, a) \tag{3}$$

dove P è l'insieme delle sorgenti fisiche considerate, e $I_p(e, a)$ rappresenta il numero di interazioni tra due utenti per una data sorgente di dati. Infine, la forza complessiva del collegamento sociale tra e ed a è data dalla combinazione lineare delle interazioni online e fisiche descritte prima:

$$\omega_s(e, a) = \lambda \cdot \omega_{osn}(e, a) + (1 - \lambda) \cdot \omega_{phy}(e, a) \tag{4}$$

con il parametro λ che regola l'importanza delle interazioni sociali e fisiche. Per ogni alter, solo l'ultimo peso calcolato è mantenuto in memoria, e viene aggiornato quando nuove interazioni sociali sono identificate. I link sociali tra l'utente locale e le altre persone sono raggruppate in base al peso calcolato nell'Equazione 4. L'output finale è un array di valori in cui ogni elemento rappresenta la percentuale di utenti attivi in ogni cerchio della ego network di un utente.

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] Arthur Mello. How do netflix and amazon know what i want? `https://towardsdatascience.com/how-do-netflix-and-amazon-know-what-i-want-852c480b67ac`, 2020.

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

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

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

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

[8] 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.

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

[10] Google Developers. Matrix factorization. `https://developers.google.com/machine-learning/recommendation/collaborative/matrix`, 2020.

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

[12] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*, pages 73–105. 01 2011.

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

[14] Gediminas Adomavicius, Bamshad Mobasher, Francesco Ricci, and Alexander Tuzhilin. Context-aware recommender systems. *AI Magazine*, 32(3):67–80, Oct. 2011.

[15] 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.

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

[17] 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.

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

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

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

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

[22] 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.

[23] Moshe Unger and Alexander Tuzhilin. Hierarchical latent context representation for context-aware recommendations. *IEEE Transactions on Knowledge and Data Engineering*, PP, 09 2020.

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

[25] Ian Davidson and S. S. Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *Knowledge Discovery in Databases: PKDD 2005*, pages 59–70, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[26] Prasad Pore. What is the curse of dimensionality? `https://www.kdnuggets.com/2017/04/must-know-curse-dimensionality.html`, 2017.

[27] Aurélien Géron. Hands-on machine learning with scikit-learn and tensorflow: Concepts. *Tools, and Techniques to build intelligent systems*, 2017.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[29] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15, 01 2010.

[30] Jason Brownlee. How to choose an activation function for deep learning. `https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/`, 2021.

[31] Daniel Godoy. Understanding binary cross-entropy / log loss: a visual explanation. `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`, 2018.

[32] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.