UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea magistrale in
Informatica

STUDIO E SVILUPPO DI UN SISTEMA DI RACCOMANDAZIONE CONTEXT-AWARE PER SISTEMI MOBILI E PERVASIVI

Relatore: Prof. Elena Pagani
Correlatori: Dott. Franca Delmastro
Dott. Mattia Campana

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:

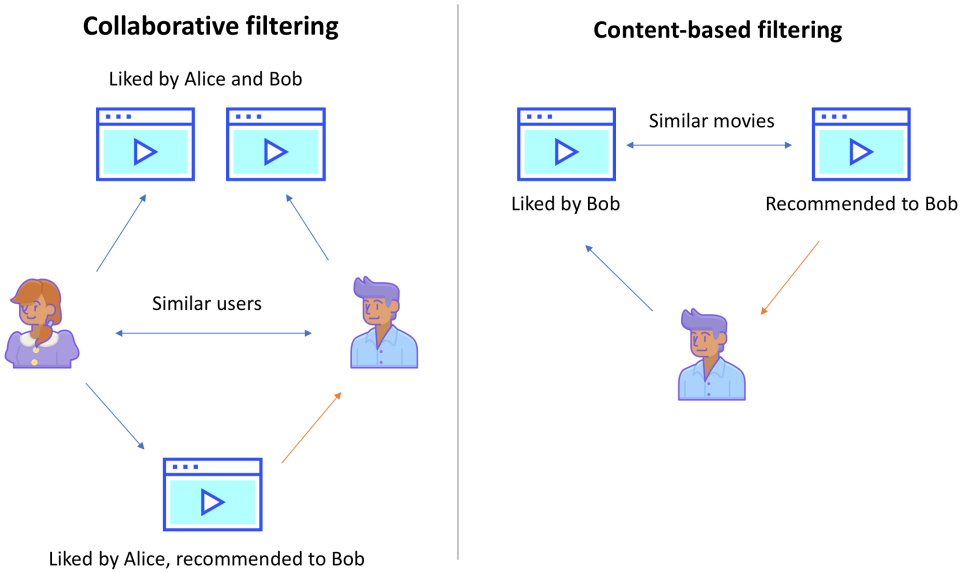

Figura 1: Collaborative filtering vs Content-based
[3]

- 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. In letteratura sono stati proposti diversi modelli CF: il più conosciuto è *matrix factorization* [5] (discusso nella sottosezione 2.1.1), che fattorizza la matrice dei rating in due matrici a dimensionalità minore. Per mantenere una complessità computazionale bassa su dataset di grandi dimensioni sono state proposte le factorization machines [6], che rappresentano le interazioni utente - oggetto come tuple di vettori di feature. Più recentemente sono stati proposti approcci basati sul deep learning [7] [8], che utilizzano *deep neural network* (discusse nella sottosezione 2.1.3) per imparare le interazioni utente - oggetto.

2.1.1 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, 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 indicano 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 [5]. 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.

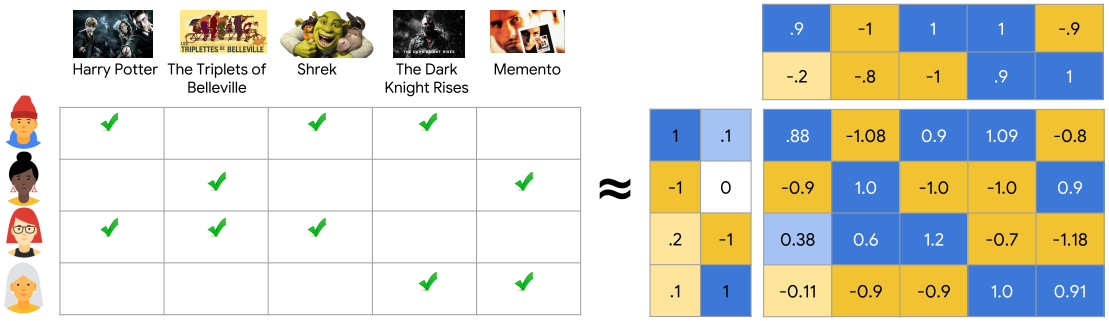

Figura 2: Approssimazione matrice dei rating con matrix factorization [9]

- 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 [9]. In letteratura esistono diversi algoritmi per calcolare i fattori latenti di utenti e oggetti. Il primo algoritmo proposto è stato FunkSVD [10] che supporta unicamente feedback espliciti, SVD++ [11] aggiunge il bias di utenti e oggetti e il supporto a feedback impliciti; ALS [12] e logistic MF [13] sono invece algoritmi progettati con i feedback impliciti in mente.

2.1.2 Alternating least square

Alternating Least Square (ALS) [12] 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 [14].

2.1.2.1 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 [12]:

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

2.1.2.2 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 2.1.1, 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 obiettivo:

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

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 obiettivo 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.

2.1.3 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 prestazioni 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 prestazioni 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 [8] 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 3 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 3 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 [15] per imparare le interazioni utente-oggetto. In particolare sono proposti tre modelli basati su deep neural network:

1. Generalized Matrix Factorization (GMF), illustrato in Sez. 2.1.3.2
2. Multi-Layer Perceptron (MLP), illustrato in Sez. 2.1.3.3
3. Neural Matrix Factorization (NeuMF), , illustrato in Sez. 2.1.3.4

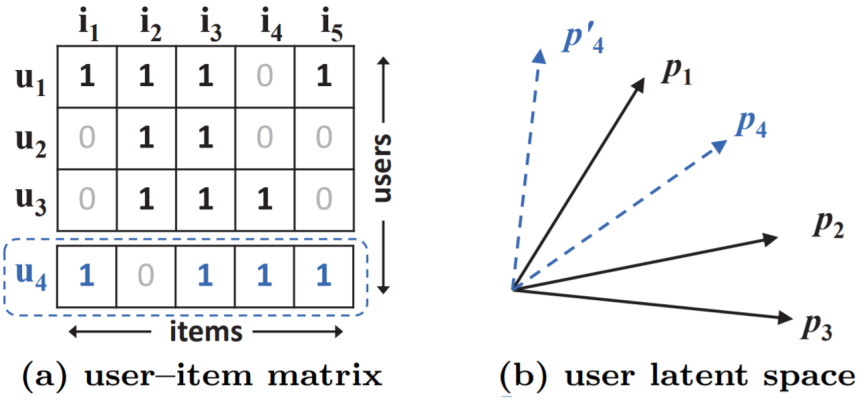

Figura 3: Limitazioni di matrix factorization
[8]

2.1.3.1 Approccio 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 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 approccio generale è chiamato *neural collaborative filtering* (NCF), ed è mostrato in Figura 4.

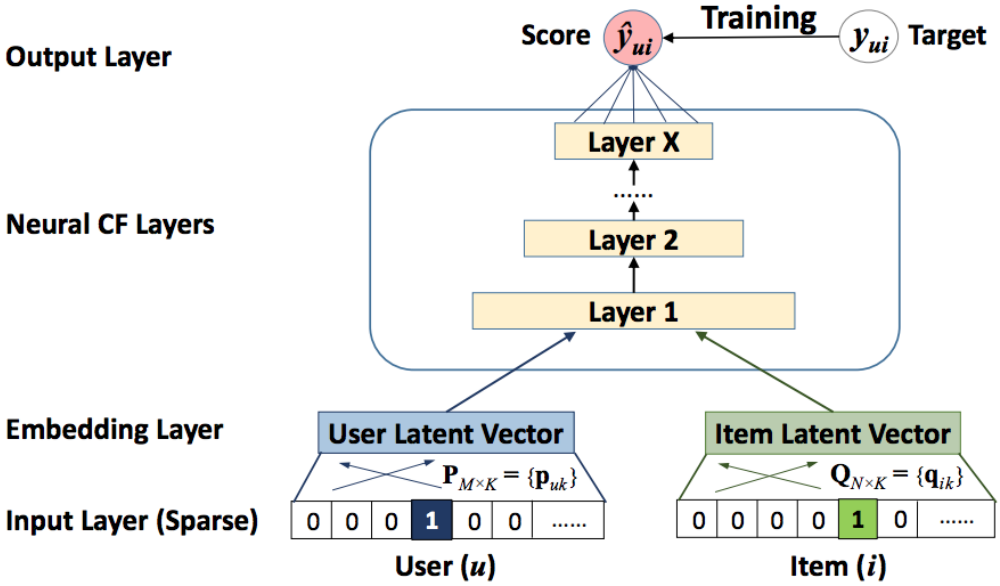

Figura 4: Approccio neural collaborative filtering
[8]

2.1.3.2 Generalized matrix factorization

In [8] viene mostrato come la famiglia di metodi basata su matrix factorization possa essere considerata un caso particolare dell'approccio NCF. 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. Se si usa una funzione identità per a_{out} e si impone che h sia un vettore uniforme di 1, si può ricostruire esattamente il modello MF. Questo modello è chiamato generalized matrix factorization (GMF).

2.1.3.3 Multi-layer perceptron

In questa istanza di NCF, i vettori di embedding di utenti e oggetti ottenuti dai layer di embedding sono concatenati. Sopra al layer di concatenazione 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, invece di imparare un prodotto elemento per elemento come GMF.

2.1.3.4 Neural matrix factorization

A questo punto si hanno due diverse implementazioni del framework NCF: GMF che applica un funzione lineare per imparare le interazioni delle feature latenti, e MLP che invece applica un funzione 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 5. A sinistra è rappresentata la rete GMF, a destra il MLP.

2.1.4 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 [16]. 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 [16] [17].

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

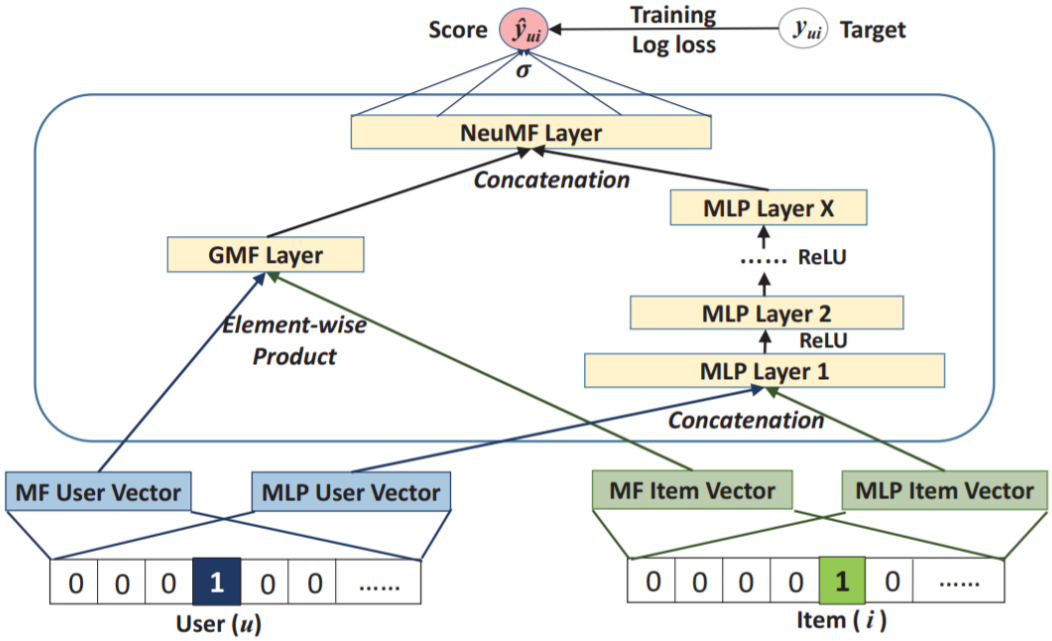

Figura 5: Neural matrix factorization
[8]

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 [18].

- *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 [16].

- *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 [16].

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 [19].

Mentre nei CF la similarità tra due oggetti (o due utenti) è calcolata 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 [16].

- *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 [20].

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 [16].
- *Raccomandazioni multi-dominio:* è difficile creare RS multi-dominio 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 [19] 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 [21] 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.

In letteratura sono stati proposti diversi sistemi di raccomandazione che integrano informazioni contestuali: in [22] vengono ideati tre diversi metodi per consentire a MF di supportare informazioni di contesto, in [23] viene proposto un modello di tensor factorization context aware, in [24] è usato un approccio deep-learning per produrre raccomandazioni context-aware con una rete neurale. Per quanto riguarda le raccomandazioni in ambito mobile, in [25] viene usato il contesto corrente dell'utente e i suoi ascolti passati per generare una playlist di canzoni. In [26] viene usato il contesto raccolto da uno smart device per raccomandazioni musicali, in [27] viene sfruttata la posizione dell'utente per fornire raccomandazioni su luoghi o eventi nelle vicinanze. Tutti questi sistemi producono raccomandazioni context-aware per dispositivi mobili, ma il RS non è implementato direttamente sul dispositivo dell'utente.

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].

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

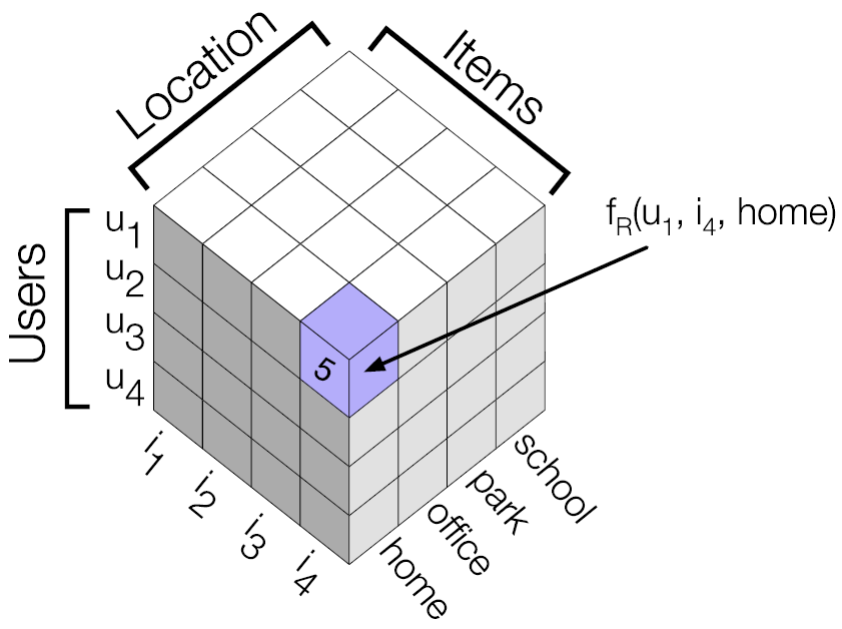

Figura 6: Esempio di un cubo multidimensionale per $User \times Item \times Location$ [2]

In questo caso, il risultato è un cubo n-dimensionale invece di 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 feature del contesto [19]. In Figura 6 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 7 si possono identificare tre paradigmi principali per integrare il contesto nei sistemi di raccomandazione [21]:

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.

2. *Contextual post-filtering:* In questo paradigma, le informazioni contestuali sono inizialmente ignorate e i rating sono predetti utilizzando qualsiasi

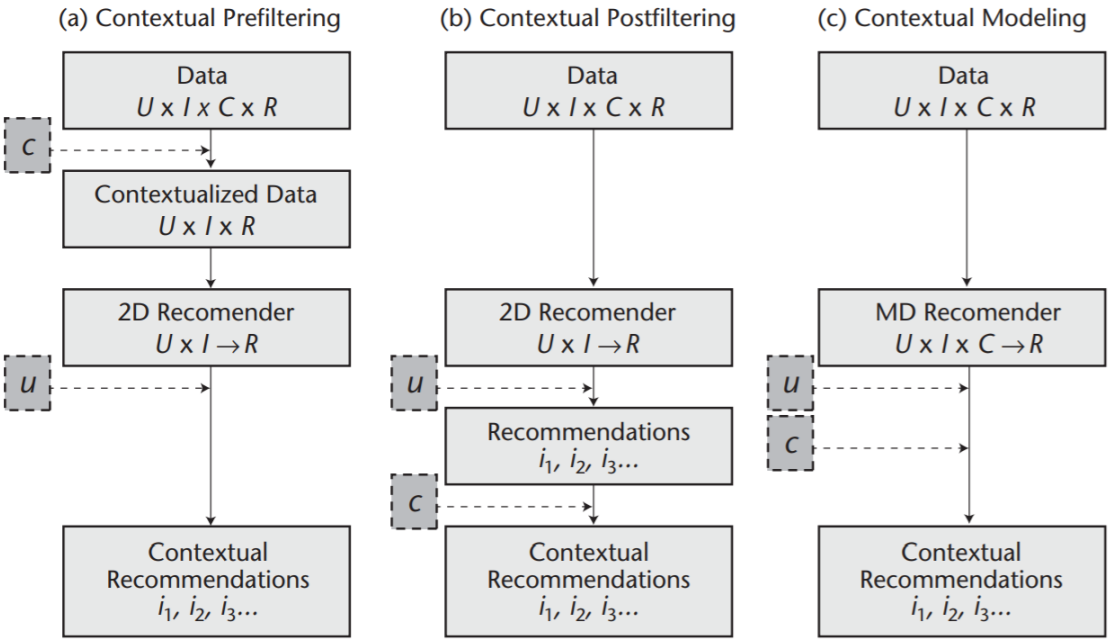

Figura 7: Paradigmi per incorporare il contesto nei sistemi di raccomandazione [28]

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 filtering 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.

2.4.2 Raccomandazioni context-aware con modelli di deep learning

I sistemi di raccomandazione context-aware tradizionali come matrix factorization [22] o tensor factorization [23], 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

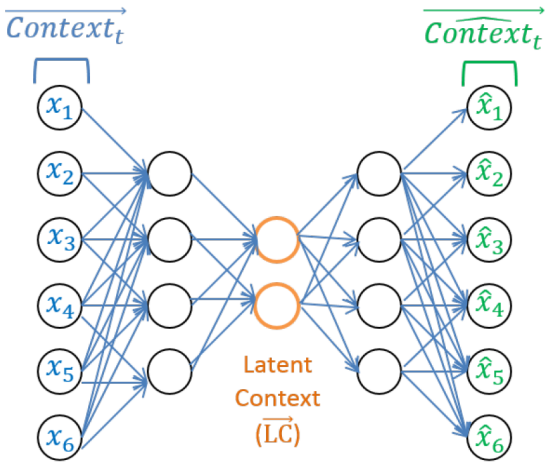

Figura 8: Esempio struttura di un autoencoder [31]

bassa dimensionalità del contesto che permette di integrarlo facilmente nei sistemi di raccomandazione esistenti [24]. Infatti un dataset con molte feature porta naturalmente ad uno spazio multidimensionale e quindi a sparsità.

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 [24]. 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 Principal Component Analysis (PCA) [29] [30], di costruire una rappresentazione gerarchica del contesto [31], e di usare dei modelli di deep learning in grado di supportare molte feature di contesto[24].

2.4.2.1 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 [29]. 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) [32]. Limitando il numero di unità nei layer nascosti di un AE, la rete è costretta a 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.

Nonostante la rappresentazione compressa del contesto possa portare a delle raccomandazioni più precise, l'applicazione di questa tecnica non è consigliabile in ambiente mobile. Il problema in ambiente mobile è che il dataset S è in continua evoluzione, e nuovi campioni t sono aggiunti al dataset quando un utente interagisce con altri dispositivi tramite comunicazione D2D. Questo significa che il contesto latente calcolato con la matrice W ottenuta dall'autoencoder con training sul dataset S, dovrà essere aggiornato nel momento in cui sono ottenuti nuovi campioni t di contesto. Aggiornare il contesto latente significa continuare il training dell'autoencoder sull'insieme dei nuovi campioni S^1, ricavare la matrice W

Algoritmo 1 Estrarre il contesto latente usando un auto-encoder [29]

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

dei pesi a training finito, e calcolare il contesto latente per i campioni vecchi e nuovi. Questo processo aggiunge una complessità indesiderabile che non giustifica il miglioramento nelle prestazioni del RS.

2.4.2.2 Rappresentazione gerarchica del contesto

Il metodo per ridurre la dimensionalità del contesto descritto nella sottosottosezione 2.4.2.1 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 [31] 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".

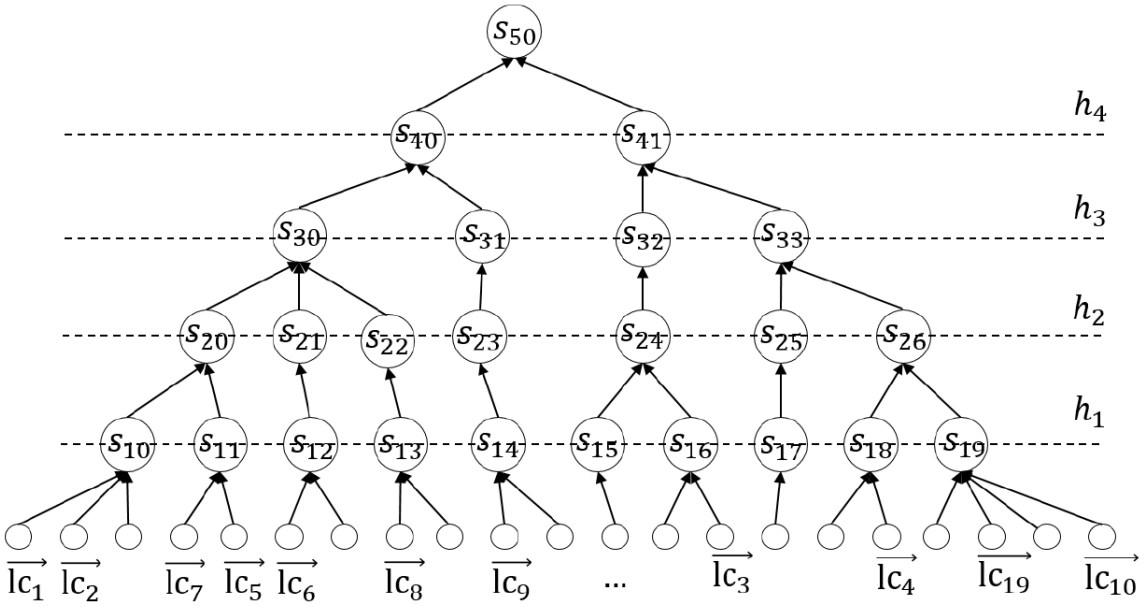

Figura 9: Gerarchia delle situazioni contestuali
[31]

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) [33] 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 $\overrightarrow{hlc_{19}} = [s_{19}, s_{26}, s_{33}, s_{41}]$.

L'estrazione del contesto gerarchico è un'operazione irrealizzabile su dispositivo mobile. Infatti, come è sottolineato in [31], il training di un autoencoder su n vettori di contesto per comprimere le feature di contesto originali da l a r dimensioni, ha una complessità di $O(n \cdot l \cdot r)$. Per estrarre il contesto gerarchico dai vettori di contesto latente viene applicato AHC che ha complessità $O(n^2)$ per n osservazioni. AHC richiede inoltre $O(n^2)$ memoria, che è un requisito troppo alto per dispositivi mobili.

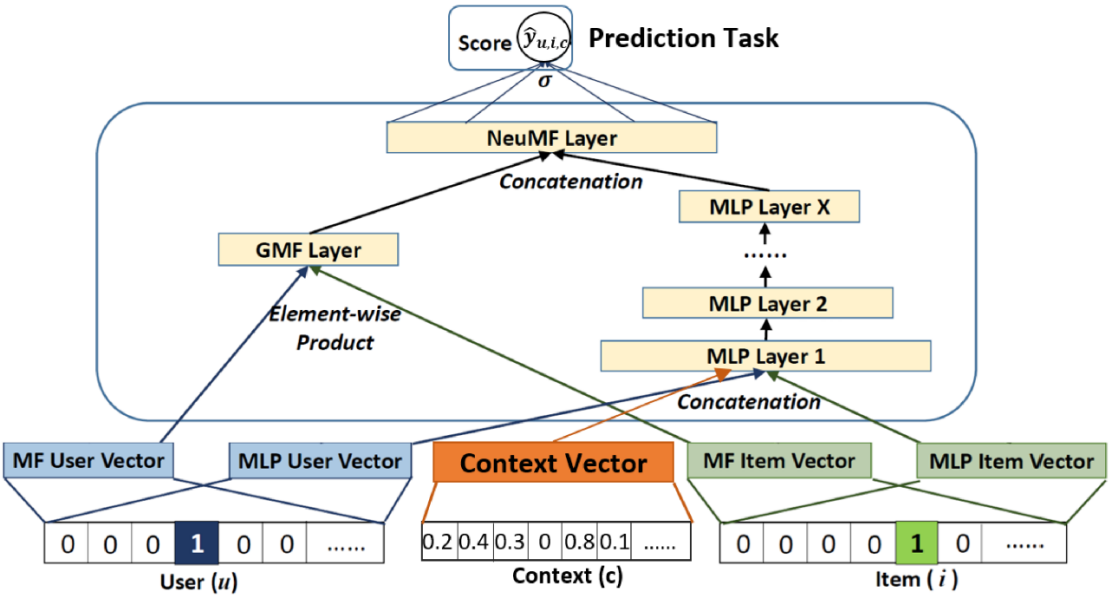

Figura 10: Estensione di NeuMF con features di contesto [24]

2.4.2.3 Modelli CARS deep learning

In [24] viene proposto di estendere il modello NeuMF [8] descritto nella sottosezione 2.1.3 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 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 [29] come descritto nella sottosottosezione 2.4.2.1 prima di essere incorporato nel modello.

3. *Hierarchical context-aware model (HCAM)*: il contesto viene organizzato in un albero gerarchico [31] come descritto nella sottosottosezione 2.4.2.2 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.

2.5 Problemi RS client-server

In letteratura la stragrande maggioranza dei RS proposti si basa su un'architettura client-server. Il RS che esegue sul server ha una conoscenza completa di utenti, oggetti e rating. Il client, fisso o mobile, esegue delle query al RS per ricevere raccomandazioni. I problemi nel realizzare un RS context-aware che esegue sul dispositivo locale dell'utente sono principalmente due:

1. I sistemi di raccomandazione mobile hanno una conoscenza solo parziale di utenti, oggetti, e dei feedback che gli utenti hanno lasciato sugli oggetti. Questo è dovuto al fatto che il RS inizialmente è a conoscenza solo dei feedback dell'utente locale, e ne scopre di nuovi tramite comunicazione D2D con altri dispositivi.

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.

Più nel dettaglio per gli algoritmi visti in questo capitolo:

ALS
L'input di ALS (sottosezione 2.1.2) è una matrice R di dimensione $u \times i$ con u numero di utenti, e i numero di oggetti. 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 nella sottosezione 2.4.1, ogni feature di contesto è una dimensione aggiunta alla matrice dei rating R. Per mantenere una complessità computazionale bassa, per i modelli MF context-aware si seleziona un insieme molti limitato di feature contestuali, che potrebbero non rappresentare pienamente il contesto corrente dell'utente.

NeuMF
Neural matrix factorization (sottosezione 2.1.3) implementa un RS collaborative-filtering con una deep neural network. 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 predire i rating solo degli stessi utenti/oggetti già presenti durante il training. Questo perché l'input deve avere la dimensione definita a livello di compilazione, e di conseguenza il numero di utenti e oggetti è lo stesso definito a livello di compilazione. Come per ALS, aggiungere un nuovo utente od un nuovo oggetto significa dover compilare di nuovo il modello ed eseguire il training da zero. Questo modello inoltre, come presentato in [8] non ha nessun supporto per le feature di contesto.

Context-aware NeuMF
Context-aware neural matrix factorization è un estensione di NeuMF che supporta le feature di contesto. 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

Come abbiamo visto nel capitolo precedente, la maggior parte dei RS produce le proprie raccomandazioni usando unicamente informazioni su utenti, oggetti, e le valutazioni che gli utenti hanno dato agli oggetti (rating). Meno comuni sono invece i sistemi di raccomandazione context-aware, che considerano nel processo di raccomandazione anche il contesto in cui un rating è stato generato. La maggior parte dei CARS sfruttano un insieme limitato di feature di contesto selezionate manualmente, in modo da evitare di dover gestire una matrice dei rating con dimensionalità elevata. Recentemente sono stati proposti i primi sistemi di raccomandazioni context-aware che integrano nel processo di raccomandazione un alto numero di feature contestuali, come può essere quello estratto dai sensori di un dispositivo mobile. Come spiegato in conclusione al Capitolo 2, le soluzioni proposte non sono tuttavia adatte per essere implementate direttamente su dispositivo mobile, anche se dimostrano come un approccio deep learning possa essere un'ottima soluzione per gestire la dimensionalità del contesto. Per questi motivi in questo capitolo è descritto un nuovo sistema di raccomandazione context-aware basato su deep learning per sistemi mobili e pervasivi, progettato nel corso di questo lavoro di tesi. Il sistema di raccomandazione è stato pensato per eseguire le fasi di training e inferenza su dispositivo mobile - così da ridurre problemi di privacy derivanti da trasferimento dei dati ed elaborazione in un'infrastruttura cloud remota - e per supportare un grande numero di feature di contesto. Per le caratteristiche appena elencate, questo nuovo RS è chiamato *moveCARS* (MObile pervasiVE Context-Aware Recommender System).

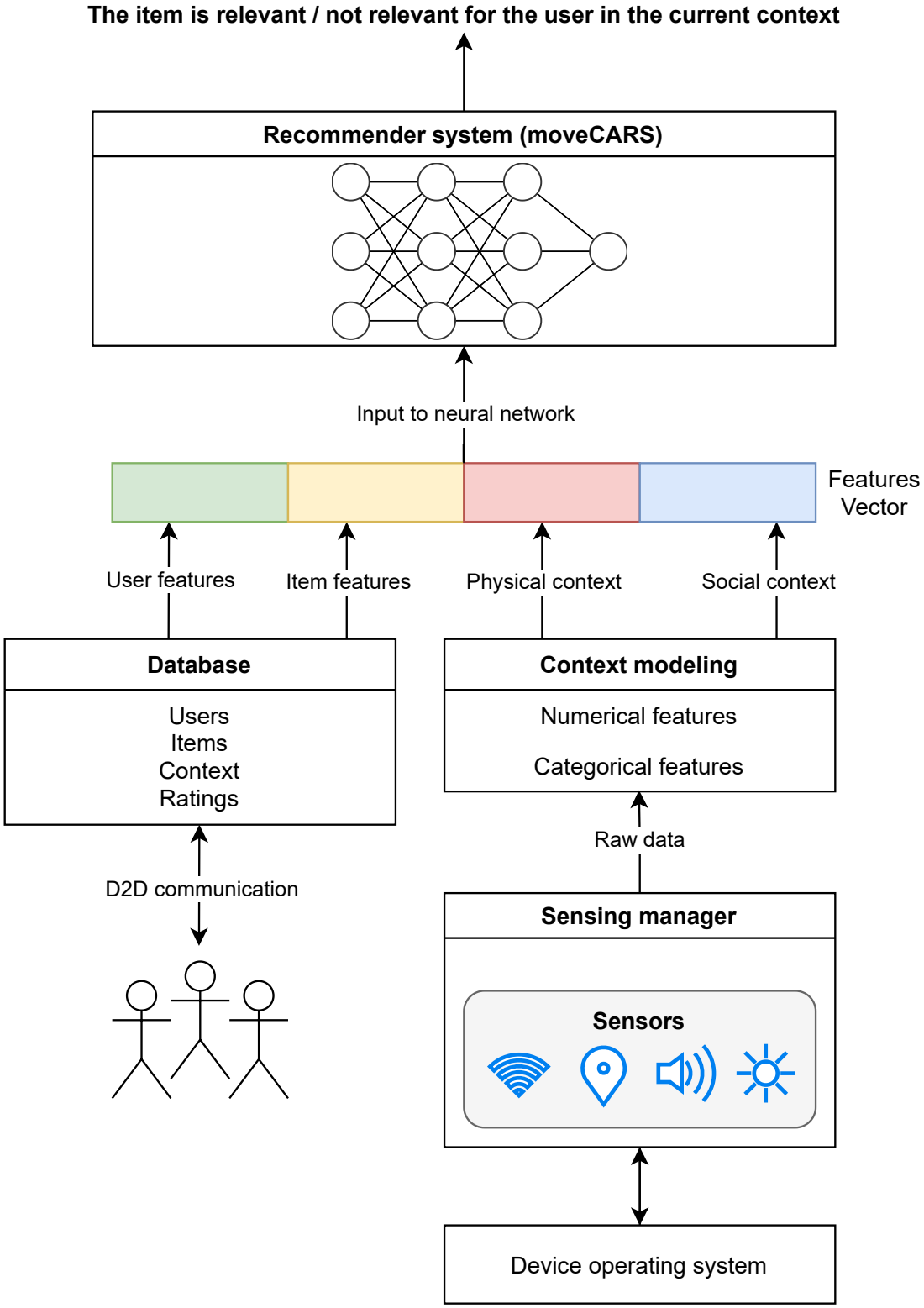

Figura 11: Architettura ad alto livello del sistema di raccomandazione

3.1 Architettura generale

Generare raccomandazioni sul dispositivo mobile non è sufficiente per realizzare un RS pervasivo che non dipenda da un server centralizzato. È necessario spostare anche le fasi di raccolta ed elaborazione dei dati su dispositivo mobile. Per questo motivo il modello moveCARS è inserito in un architettura più complessa, che si occupa di generare i dati che saranno input per il RS. L'architettura ad alto livello è composta da quattro componenti (Figura 11):

1. *Sensing manager.* Il primo componente (SM) interagisce con il sistema operativo per raccogliere continuamente dati di contesto via sensori a bordo (come GPS e accelerometro), e dati che rappresentano informazioni sullo hardware e il software del dispositivo come lo stato del display e il livello di batteria. Tutte queste informazioni sono chiamate dati raw perchè non sono ancora state processate.

2. *Context modeling.* I dati raw prodotti dal sensing manager devono essere processati 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 (es. attività utente, luogo in cui si trova l'utente, temperatura). Il contesto è diviso in contesto fisico e sociale. Il contesto fisico è ottenuto dai sensori del dispositivo dell'utente, mentre il contesto sociale è ottenuto considerando gli individui in prossimità dell'utente. Dell'insieme di feature eterogenee ottenute viene fatto l'encoding per poi essere combinate in un singolo vettore di feature che rappresenta una fotografia del contesto corrente dell'utente. L'insieme di feature che compongono il contesto fisico e sociale dell'utente è descritto nel dettaglio nella sezione 3.3.

3. *Database.* Interagendo con altri dispositivi tramite comunicazione D2D, il device dell'utente scopre nuovi utenti e oggetti che sono identificati con le feature che li caratterizzano. Oltre a questo il dispositivo riceve anche i feedback che gli utenti hanno generato sugli oggetti in un certo contesto. Tutte queste informazioni sono date in input al sistema di raccomandazione durante la fase di apprendimento per imparare a prevedere i feedback dell'utente locale. A training terminato, le feature degli oggetti presenti nel database, e le feature dell'utente corrente, sono usate per produrre raccomandazioni. Le fasi di training e inferenza sono descritte nella sottosezione 3.2.3.

4. *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 ad 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 3.2.

3.2 moveCARS

In questa sezione è descritto il sistema di raccomandazione moveCARS. Nella prima parte è descritto l'input, e in che modo si differenzia dai sistemi di raccomandazione collaborative filtering e content-based. Nella seconda parte sono descritti nel dettaglio la struttura e il training della rete neurale che genera le raccomandazioni context-aware. In conclusione sono descritti i vantaggi e gli svantaggi del modello.

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`. Invece di 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 non specifiche come età o genere, o feature specifiche per l'ambiente in cui il RS è implementato. Tornando all'esempio dei ristoranti, si potrebbe chiedere all'utente 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 moveCARS è quindi una tupla `(user_features, item_features, rating, physical_context, social_context)`.

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'obiettivo è prevedere il valore di una variabile che può

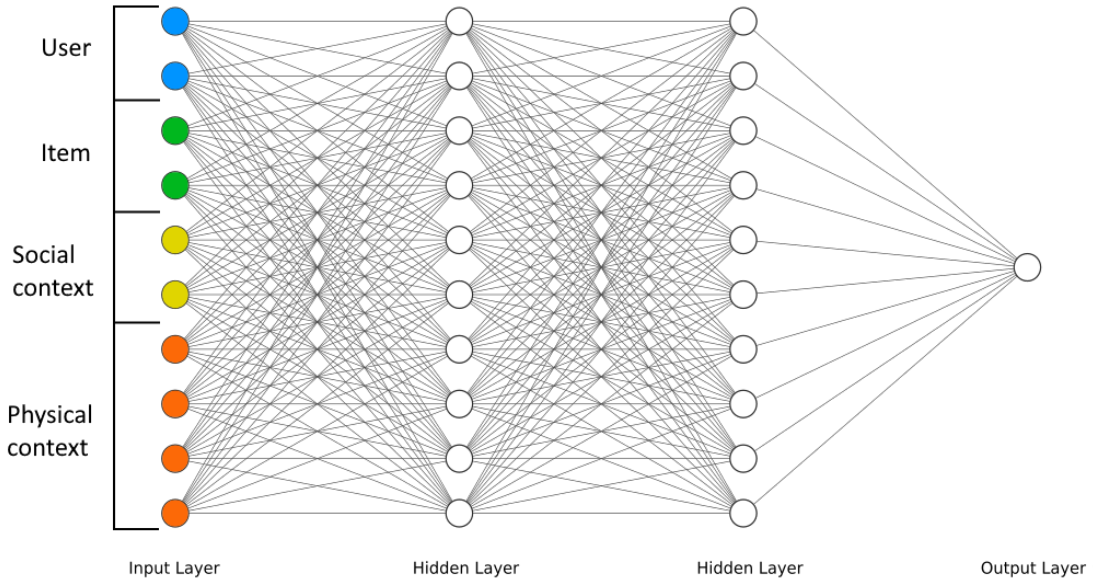

Figura 12: Schema di moveCARS

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 [34].

Layer e neuroni La rete neurale scelta rientra nella categoria feed-forward fully connected. Una rete feed-forward non contiene cicli nel suo grafo [35], 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 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 [35], 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 [36]. 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 [37].

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 [38].

Ottimizzatore Un ottimizzatore è un algoritmo che modifica i pesi del modello in modo da minimizzare la funzione di loss e rendere le previsioni della rete più accurate possibile. Come ottimizzatore ho scelto Adam; nell'articolo in cui é 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 [39]. Adam ha diversi iperparametri configurabili, il più importante è il learning rate (chiamato α in Adam) che regola la velocità con cui il modello è adattato al problema. 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 calcolate tramite grid search nel Capitolo 5. La batch size corrisponde al 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 prima di concludere il training.

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

In Figura 12 è rappresentata la struttura della rete neurale. In questo caso il modello ha due layer nascosti ed ogni layer contiene 10 neuroni, eccetto il layer di output che contiene un solo neurone.

3.2.3 Training e inferenza

Training
Il processo di training di una rete neurale si basa sul trovare un insieme di pesi nella rete che permettano di risolvere nel modo migliore possibile un problema specifico. Il processo di training è iterativo, il che significa che procede passo dopo passo con piccoli aggiornamenti nei pesi del modello ad ogni iterazione, migliorando le performance del modello. Il processo di training iterativo di una rete neurale risolve un problema di ottimizzazione per dei parametri, (i pesi del modello) che ha come risultato un errore minimo durante la valutazione degli esempi nel training dataset. In ambiente mobile e pervasivo inizialmente il numero di esempi nel training dataset è limitato, e non rappresenta la conoscenza globale su tutti gli utenti, oggetti e rating. Questo non è un problema per moveCARS che può iniziare il training sui campioni disponibili, per poi riprenderlo in un secondo momento quando il device utente tramite comunicazione D2D avrà scoperto nuovi campioni. Questo è possibile per la struttura dell'input della rete neurale. L'input è un vettore formato dalla concatenazione di `user_features, item_features, physical_context, social_context`, la cui dimensione è fissata. Infatti utenti, oggetto e contesto sono definiti da un insieme di feature che non cambia nel tempo. Non è quindi necessario ridefinire il modello ogni volta che il numero di utenti o oggetti cambia, cioè quando sono scoperti nuovi utenti od oggetti.

Inferenza
Il task per un sistema di raccomandazione che fa classificazione è determinare se un utente è interessato ad un oggetto in una situazione descritta dal contesto fisico e sociale. La rete restituisce valore 1 se l'oggetto è rilevante, 0 altrimenti. Nel caso di moveCARS utenti e oggetti non sono definiti con degli ID numerici, come nei metodi collaborative filtering, ma da feature che li caratterizzano. Il task di moveCARS quindi può essere riformulato in modo più specifico come prevedere se all'utente con feature `user_feature` interessa un oggetto con feature `item_feature`, nel contesto fisico `physical_context`, e nel contesto sociale `social_context`.

3.2.4 Vantaggi e svantaggi

Di seguito sono elencati vantaggi e svantaggi del modello moveCARS. Dato che può essere considerato un sistema ibrido che unisce caratteristiche degli approcci collaborative filtering e content-based, eredita alcuni vantaggi e svantaggi da entrambe le categorie di algoritmi.

Vantaggi:

1. *Nessuna conoscenza del numero di utenti e oggetti.* Utenti e oggetti sono rappresentati con delle feature e non con il loro ID, non è necessario conoscere a priori quanti utenti e oggetti sono presenti nel sistema.
2. *Consigliare nuovi oggetti.* Il modello può consigliare nuovi oggetti anche se non ci sono valutazioni fornite dagli utenti, a differenza dei metodi collaborative filtering.
3. *Feature di contesto.* Il modello permette di integrare una grande quantità di feature di contesto che possono migliorare sensibilmente la capacità di predizione, come dimostrato nel Capitolo 5.
4. *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, a differenza dei metodi content-based.

Svantaggi:

1. *Feature di utenti e oggetti:* la precisione del modello dipende dall'insieme delle feature che descrivono gli utenti e gli oggetti. È necessario selezionare attentamente le feature più adatte che descrivono utenti e oggetti in un'applicazione specifica.
2. *Raccomandazioni multi-dominio:* è difficile creare RS multi-dominio perché è complicato definire un insieme di feature che valgano per contenuti di natura diversa.

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. Nel Capitolo 5 è

dimostrato che un insieme ampio di feature contestuali può portare a raccomandazioni molto più accurate, mentre un insieme di feature contestuali poco esteso ha un impatto decisamente minore.

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 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 livello 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).

Applicazioni Applicazioni in esecuzione sul dispositivo.

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 messaggiare con gli amici. É stato provato in letteratura che esiste una forte correlazione tra le attività umane e i dati sociali [40]. Questo implica che modellare una rete di relazioni sociali specifica per l'utente può contribuire a sottolineare le differenze tra i vari contesti in cui è coinvolto.

3.3.2.1 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 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 13: 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 (es. i famigliari). Il secondo layer (sympathy group) contiene le persone che possono essere considerate 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 [40].

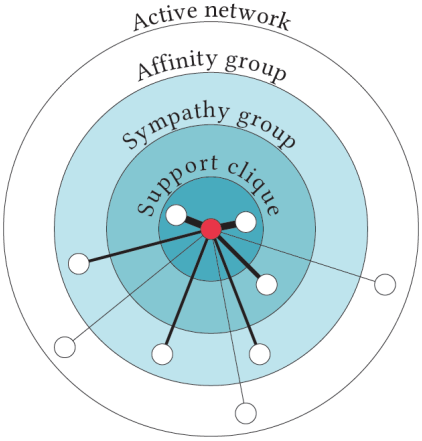

Figura 13: Ego network
[40]

3.3.2.2 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 la 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 prese 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 suoi 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 fisicamente abbastanza vicine (in raggio radio) 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

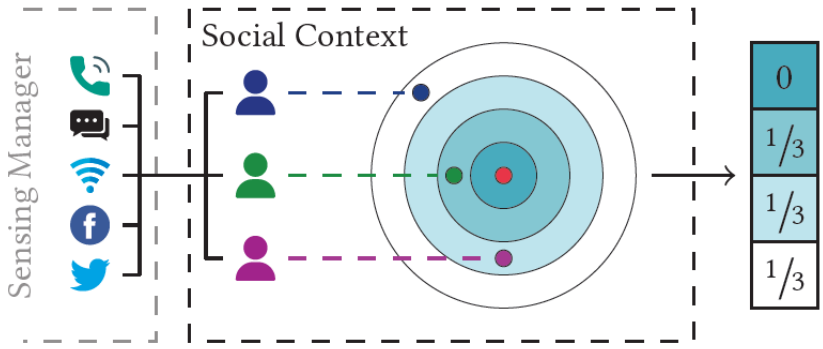

Figura 14: Riconoscimento del contesto sociale di un utente [40]

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 raggruppati 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 [40].

La Figura 14 mostra un esempio del processo di riconoscimento del contesto. Basandosi sui dati raccolti dal Sensing Manager, viene riconosciuta la rilevanza degli alter per l'utente locale in base al livello in cui essi sono posizionati nella ego network. L'output caratterizza il contesto sociale dell'utente, indicando chiaramente che sta interagendo con tre persone diverse che si trovano nel secondo, terzo e quarto layer rispettivamente.

Nei prossimi due capitoli sono descritti nel dettaglio due dataset context-aware, ed è presentata una valutazione delle prestazioni di moveCARS comparata con le principali soluzioni in letteratura.

Capitolo 4

Dataset

In questo capitolo sono descritti i dataset context-aware usati per valutare il modello moveCARS, e confrontarlo con altre soluzioni stato dell'arte. I risultati sono riportati nel Capitolo 5. Uno dei problemi nella valutazione dei CARS è la scarsità di dataset pubblici che contengono informazioni di contesto ad alta dimensionalità. Molti dataset pubblici infatti, hanno informazioni di contesto limitate unicamente al timestamp dei rating o alla posizione dell'utente, come ad esempio Yelp[1], Nowplaying-RS[2], Travel TripAdvisor[3]. Esistono invece dataset privati come CARS [24] e Hearo [29], che contengono i dati raccolti da esperimenti sul campo, in cui gli utenti interagivano con il proprio telefono con un RS che raccomandava punti di interesse nelle vicinanze. I rating raccolti sono associati a numerose feature di contesto estratte da sensori di apparati mobili come accelerometro, microfono, giroscopio, etc. È difficile trovare dataset pubblici simili a Hearo e CARS, che caratterizzano il contesto fisico e sociale dell'utente con una grande quantità di feature. Dato che il modello moveCARS si inserisce in un ambiente mobile e pervasivo, ho selezionato due dataset context-aware in cui i feedback impliciti corrispondono alle applicazioni in esecuzione sui dispositivi Android degli utenti: (i) My Digital Footprint[4], (ii) Frappe[5].

4.1 Frappe

Frappe [41] è un dataset di feedback impliciti pubblicamente disponibile collezionato da un sistema di raccomandazione context-aware di applicazioni Android.

[1] `https://www.yelp.com/dataset`

[2] `https://zenodo.org/record/3247476#.YK9FxqgzY2x`

[3] `https://github.com/irecsys/CARSKit/blob/master/context-aware_data_sets/Travel_TripAdvisor_v1.zip`

[4] `https://github.com/contextkit/MyDigitalFootprint`

[5] `https://www.baltrunas.info/context-aware`

L'applicazione che monitora l'utilizzo degli smartphone, è stata installata da 957 utenti che hanno utilizzato un totale di 4082 applicazioni. Le informazioni raccolte descrivono la frequenza di utilizzo di un'applicazione da parte di un utente per un periodo di due mesi. Il numero totale di feedback presenti è 96203. L'obiettivo per un recommender system su questo dataset è prevedere se un'applicazione Android è rilevante per un utente in un determinato contesto.

4.1.1 Feature di contesto

Le feature di contesto descrivono la situazione dell'utente nel momento in cui ha utilizzato un'applicazione Android. Frappe può essere considerato un dataset con un contesto a bassa dimensionalità, e non contiene informazioni raccolte dai sensori dei dispositivi Android. Di seguito sono descritte tutte le feature di contesto presenti nel dataset.

Daytime è il momento della giornata in cui un'applicazione è stata utilizzata. La giornata è divisa in sette momenti diversi: mattina, mezzogiorno, pomeriggio, sera, tramonto, alba, notte.

Weekday è il giorno della settimana in cui un'applicazione è stata utilizzata. I possibili valori sono ovviamente i sette giorni della settimana.

Isweekend indica se un'applicazione è stata utilizzata nel fine settimana oppure in un giorno lavorativo. Può assumere due valori diversi: weekend e workday.

Homework indica se l'utente si trova al lavoro o a casa. Può assumere tre diversi valori: casa, lavoro, sconosciuto.

Weather descrive la situazione meteo nel momento in cui un'applicazione è stata utilizzata. Può assumere nove valori differenti: soleggiato, nuvoloso, nebbioso, temporalesco, piovoso, nevoso, piovigginoso, nevischio, sconosciuto.

Country indica la nazione in cui si trovava l'utente nel momento in cui ha utilizzato un'applicazione. Ci sono 80 stati diversi, ma il 55% dei feedback sono stati generati da USA, Spagna e Regno Unito.

City è un valore numerico che rappresenta la città in cui si trovava l'utente nel momento in cui ha utilizzato un'applicazione. Ci sono 233 città diverse, ma per il 40% dei feedback la città è sconosciuta.

Delle feature di contesto appena descritte sono eliminate `homework, country`, e `city` perché poco utili a definire il contesto dell'utente. In particolare `city` è stata eliminata perché contiene troppi valori nulli, ed in corrispondenza di una città sconosciuta spesso anche il valore della feature `country` è sconosciuto. Anche `homework` è stato eliminato perché il numero di feedback che hanno la feature `homework` con valore sconosciuto è pari al 78%. `Country` non viene considerata perché la maggior parte dei feedback utente sono associati a poche nazioni, e ci sono un gran numero di nazioni a cui sono associati un numero non sufficiente di feedback. Le feature rimaste compongono il contesto: `daytime, weekday, isweekend, weather`. L'unica di queste feature che può assumere valore sconosciuto è `weather`. Le righe del dataset in cui `weather` è sconosciuto sono eliminate. Il risultato è un dataset con 78335 righe, 857 utenti e 3180 oggetti.

Per quanto riguarda l'encoding, essendo tutte variabili categoriche sono codificate con one-hot encoding. Il vettore del contesto risultato contiene 24 feature: 7 per `daytime`, 7 per `weekday`, 8 per `weather` e 2 per `isweekend`.

4.1.2 Feedback

Qualsiasi dataset per sistemi di raccomandazione ha tre feature fondamentali: user, item, rating. User e item sono valori numerici che identificano univocamente gli utenti e gli oggetti. Il rating in Frappe è il numero di volte in cui un oggetto (un'applicazione) è stato utilizzato da un utente in un determinato contesto. Ad esempio, se una riga del dataset è composta da `(user:1, item:20, rating:50, daytime:morning, weekday:monday)`, significa che l'utente 1, ha utilizzato l'applicazione 20, il lunedì mattina, 50 volte. Il numero di volte è ottenuto sommando tutti gli utilizzi durante il periodo di raccolta dei dati. Il valore minimo dei rating è 1, il valore massimo 21262, e la media 88.26. Questi rating vanno convertiti in feedback impliciti con valore 0 o 1 per essere compatibili con l'input di moveCARS, come spiegato nella sottosezione 3.2.1. Considerando il valore medio dei rating, e cercando di avere un dataset bilanciato, ho deciso di convertire tutti i rating con valore maggiore di 4 in feedback con valore 1, mentre i rating con valore 4 o minore in feedback negativi. Il risultato è un dataset con il 62% di feedback positivi (48604 campioni).

4.1.3 Feature degli oggetti

Le feature degli oggetti sono caratteristiche che descrivono le applicazioni usate dagli utenti di Frappe. Il dataset contiene tre feature rilevanti per il task di classificazione: la categoria dell'applicazione, la lingua, e il costo.

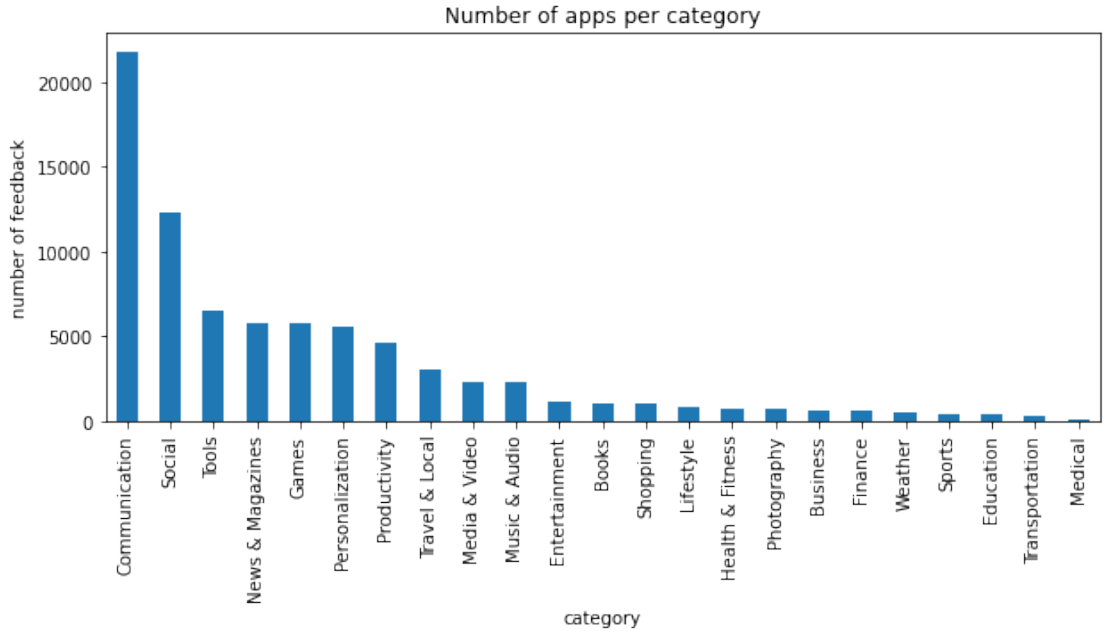

Figura 15: Numero di campioni per ogni categoria di applicazioni nel dataset Frappe

Category è la categoria delle applicazioni ottenuta dal Google Play Store. In Frappe le applicazioni sono divise in 23 categorie che descrivono la loro funzionalità principale (es. videogiochi, notizie, social). Come si può vedere in Figura 15, il numero di campioni per ogni categoria è molto sbilanciato: la categoria Communication che comprende applicazioni di messaggistica come WhatsApp e Telegram ha più di 20k campioni. La seconda categoria più popolare è Social che comprende applicazioni come Facebook, Instagram e Twitter.

Language indica la lingua dell'applicazione ottenuta dal Google Play Store. Dato che il 96% dei campioni hanno applicazioni in lingua inglese, ho assegnato a tutte le altre lingue il valore other.

Cost indica se l'applicazione è gratuita o a pagamento.

Come per le feature di contesto, anche le feature degli oggetti sono variabili categoriche. Vengono codificate con one-hot encoding; il risultato è un vettore di 27 feature: 23 per `category`, 2 per `language`, e 2 per `cost`.

4.1.4 Feature degli utenti

Il dataset Frappe non contiene nessuna informazione associata agli utenti oltre all'ID dell'utente. Per questo motivo, ho generato le feature degli utenti a partire dalle feature degli oggetti e di contesto.

User category indica la categoria di applicazioni più utilizzata dall'utente. Come era prevedibile dalla distribuzione delle categorie mostrata in Figura 15, le categorie preferite dagli utenti sono Communication e Social.

User weekday indica il giorno della settimana in cui l'utente ha generato il maggior numero di feedback. I giorni più popolari sono venerdì e sabato.

User daytime indica il momento della giornata in cui l'utente ha generato il maggior numero di feedback. I momenti della giornata più popolari sono la sera e il pomeriggio.

User weather indica la condizione meteo in cui l'utente ha generato il maggior numero di feedback. Le condizioni meteo più popolari sono nuvoloso e soleggiato.

User weekend indica se l'utente ha generato più feedback in settimana o nel weekend. La maggior parte degli utenti (85%), ha generato più feedback in settimana.

User paid apps indica se l'utente ha mai utilizzato un'applicazione a pagamento. Il 59% degli utenti ha utilizzato almeno una volta un'applicazione a pagamento.

Come le feature di contesto e degli oggetti, anche le feature degli utenti sono tutte variabili categoriche. Vengono codificate con one-hot encoding in un vettore di 47 feature: 22 per `user category`, 7 per `user weekday`, 7 per `user daytime`, 7 per `user weather`, 2 per `user weekend`, 2 per `user paid apps`.

Ricapitolando, il dataset Frappe processato contiene 78335 campioni, 857 utenti e 3180 oggetti. Oltre alle colonne user, item e feedback, il dataset contiene 47 feature degli utenti, 22 feature degli oggetti, e 24 feature di contesto.

4.2 My Digital Footprint

My Digital Footprint (MDF) [42] è un nuovo dataset composto da dati di sensori di smartphone, informazioni di prossimità fisica, e interazioni sugli online social

network. Il dataset include due mesi di misurazioni e informazioni collezionate dai dispositivi personali di 31 volontari, nel loro ambiente naturale, senza limitare il loro comportamento usuale. I dati raccolti costituiscono un insieme completo di informazioni per descrivere il contesto utente in ambiente mobile.

Il dataset è organizzato in cartelle, una per ogni utente, che contengono diversi file csv, ognuno contenente misurazioni e informazioni di tipo diverso. Ogni campione in qualsiasi file csv contiene il timestamp in cui è stato acquisito. Alcuni sensori sono stati campionati molto frequentemente, mentre informazioni come il meteo sono state raccolte ogni ora. Di seguito è riportata la struttura delle cartelle:

```
MDF
├── user_0
│   ├── activities.csv
│   ├── audio.csv
│   ├── running_apps.csv
│   ├── wifi_scans.csv
│   └── ...
├── user_1
│   ├── activities.csv
│   ├── audio.csv
│   ├── running_apps.csv
│   ├── wifi_scans.csv
│   └── ...
└── ...
```

L'obiettivo è costruire un unico file csv, in cui ogni riga ha una struttura del tipo `(user_ID, item_ID, context)`. Siccome l'obiettivo di un recommender system su questo dataset è prevedere se un'applicazione Android è rilevante per un utente in un determinato contesto, il punto di partenza per costruire il dataset sono le applicazioni in esecuzione nel file `running_apps.csv`. Qui sotto è riportato il codice Python per generare il dataset:

```
1  data_path = 'Datasets/MDF/'
2  df = pd.DataFrame()
3  # foreach user folder
4  for user in range(31):
5      user_dir = data_path + 'user_' + str(user)
6      # read running_apps.csv and use it as a starting point
7      df1 = pd.read_csv(user_dir + '/running_apps.csv', header=0)
8      df1['time'] = pd.to_datetime(df1['time'], unit='ms')
9      df1.sort_values('time', inplace=True)
10     df1.reset_index(drop=True, inplace=True)
11     df1.insert(1,'user',user) # insert user ID column
12
13     rows = []
```

```
14      # foreach row in running apps dataframe find the closest row
        in all other csv file using timestamp
15      for dt in df1['time']:
16          row = []
17          # foreach csv file in user folder
18          for filename, columns in file_dict.items():
19              file_path = user_dir + '/' + filename
20              # single row with all the context features
21              row = row + get_closest_row(file_path, columns, dt).
        tolist()
22          rows.append(row)
23
24      df2 = pd.DataFrame(rows, columns=np.concatenate(list(file_dict
        .values())))
25      df3 = pd.concat([df1, df2], axis=1) # concat by column
26      df = pd.concat([df, df3], axis=0) # concat by row
27
28  df.reset_index(drop=True, inplace=True)
```

Per ogni cartella utente (riga 4) si legge il file `running_apps.csv` (riga 7). Per ogni elemento nel file con timestamp t (riga 15), e per ogni altro file csv nella cartella dell'utente corrente, si seleziona la riga con timestamp più vicino a t (riga 21). Il risultato è una tupla `(user_ID, item_ID, context)` (riga 25) che viene concatenata al dataset finale (riga 26).

4.2.1 Negative sampling

In MDF sono presenti solo i log indicanti che un'applicazione era in esecuzione sul dispositivo dell'utente, ad un certo timestamp t, in una situazione contestuale c. Per eseguire il training di una rete neurale sono però necessari degli esempi negativi, i quali indicano che un'applicazione non era in uso da parte di un utente al tempo t', in una specifica situazione contestuale c'. Ad ogni campione è associata un'etichetta, che riassume il contesto dell'utente ad alto livello con i seguenti valori: `home, school, workplace, external school` (quando gli autori del dataset incontravano i volontari), `free time`, e `holiday`. Questa etichetta non è usata come feature di contesto, ma per fare negative sampling del dataset. L'algoritmo 2 mostra il procedimento: per ogni campione d nel dataset D con struttura $(user, item, feedback, context, label)$, vengono identificate le etichette in cui $d.item$ non è mai stato utilizzato (riga 2). Per ogni etichetta n viene scelto in modo casuale un campione $\in D$ con etichetta $= n$ (riga 4). Di questo campione viene mantenuto solo il contesto $context_{neg}$ scartando user, item e feedback. Il campione negativo d_{neg} è ottenuto concatenando $d.user$ e $d.item$, con 0 (il feedback negativo) e $context_{neg}$ (riga 6). d_{neg} è aggiunto al dataset D_{neg} che contiene solo esempi negativi (riga 7). In ultimo il dataset D_{neg} è unito al dataset D, ed

è eliminata la colonna corrispondente alle etichette (righe 10 e 11). Il risultato è una dataset con 31 utenti, 338 oggetti, e 73176 feedback, di cui il 66% con valore 1.

Algoritmo 2 Negative sampling di MDF

Input: D - dataset
Output: D_{neg} - dataset D with negative samples

1: **for all** $d \in D$ **do**
2: $labels_{neg} \leftarrow$ labels where $d.item$ was never used
3: **for all** $n \in labels_{neg}$ **do**
4: $context_{neg} \leftarrow$ context of a random data sample $\in D$ with $label = n$
5: $feedback \leftarrow 0$
6: $d_{neg} \leftarrow$ concatenate $d.user$, $d.item$, $feedback$ and $context_{neg}$
7: $D_{neg} \leftarrow D_{neg} \cup d_{neg}$
8: **end for**
9: **end for**
10: $D_{neg} \leftarrow D \cup D_{neg}$
11: $D_{neg} \leftarrow$ drop all labels from samples $\in D_{neg}$
12: Return D_{neg}

4.2.2 Feature di contesto

Il dataset MDF contiene numerosi dati estratti dai sensori e dal sistema operativo dei dispositivi personali degli utenti. Di seguito sono elencate solo le feature selezionate che compongono il contesto fisico e sociale dell'utente.

Attività utente L'attività utente, riconosciuta da Android Activity Recognition system[6], include sia movimenti a piedi che su mezzi di trasporto. Le attività possibili sono `in vehicle, on bicycle, on foot, running, still, tilting, walking, unknown`. Ogni feature rappresenta la probabilità da 0 a 100 che l'utente stia facendo quell'attività specifica.

Modalità audio `Ringer mode` indica se la modalità audio del telefono è impostata su silenzioso, vibrazione o suono.

[6] `https://developers.google.com/location-context/activity-recognition`

Volume `Alarm volume, music volume, notification volume` e `ring volume`, sono quattro feature con valore tra 0 e 1 che indicano il livello audio della sveglia, della musica, delle notifiche e della suoneria.

Musica `music active` è un valore booleano che indica se il dispositivo sta riproducendo della musica, `speaker on` specifica se è riprodotta dall'altoparlante del telefono; `headset connected` indica se sono collegate delle cuffie.

Batteria Alla batteria sono associate due feature: `level` indica la carica della batteria (molto bassa, bassa, media, alta, carica), `charging` è un valore booleano che indica se la batteria si sta ricaricando oppure no.

Schermo Associate al display ci sono due feature: `state` indica se il display è spento, acceso, o se si sta per spegnere. `Rotation` indica se l'utente sta usando il telefono in verticale o in orizzontale.

Meteo Il meteo è descritto da sei variabili diverse: temperatura, umidità, pressione atmosferica, velocità del vento, nuvole, e se ha piovuto nelle ultime tre ore.

Wifi La feature `connected` indica se il dispositivo dell'utente è connesso oppure no ad una rete Wi-Fi.

Data e ora Dai timestamp dei feedback in formato `YYYY-MM-DD HH:MM:SS`, sono estratte nuove feature: `daytime` (mattina, pomeriggio, sera, e notte) `weekday` e `isweekend`. Oltre a queste, con la libreria Python Holidays[7], è calcolato se è un giorno di vacanza o no, in base al calendario delle festività italiano.

Feature sociali Le feature sociali sono ottenute dalla ego network descritta nella sottosezione 3.3.2. Ci sono quattro feature `social_c1, social_c2, social_c3, social_c4`, che corrispondono alle quattro cerchie sociali della ego network. I valori tra 0 e 1 di queste feature indicano la percentuale di alter in ogni cerchia sociale in prossimità dell'utente, nel momento in cui ha utilizzato un'applicazione. La feature `layer` indica il posizionamento nella ego network dell'utente da cui è stata ricevuta una raccomandazione. Questo presuppone un dataset diverso per ogni utente perché la ego network è personale (es. l'utente u_1 potrebbe essere uno sconosciuto per u_2, ma un amico per u_3, e quindi u_1 è posizionato su un layer della ego network di u_2 diverso dal layer della ego network di u_3).

[7] `https://pypi.org/project/holidays/`

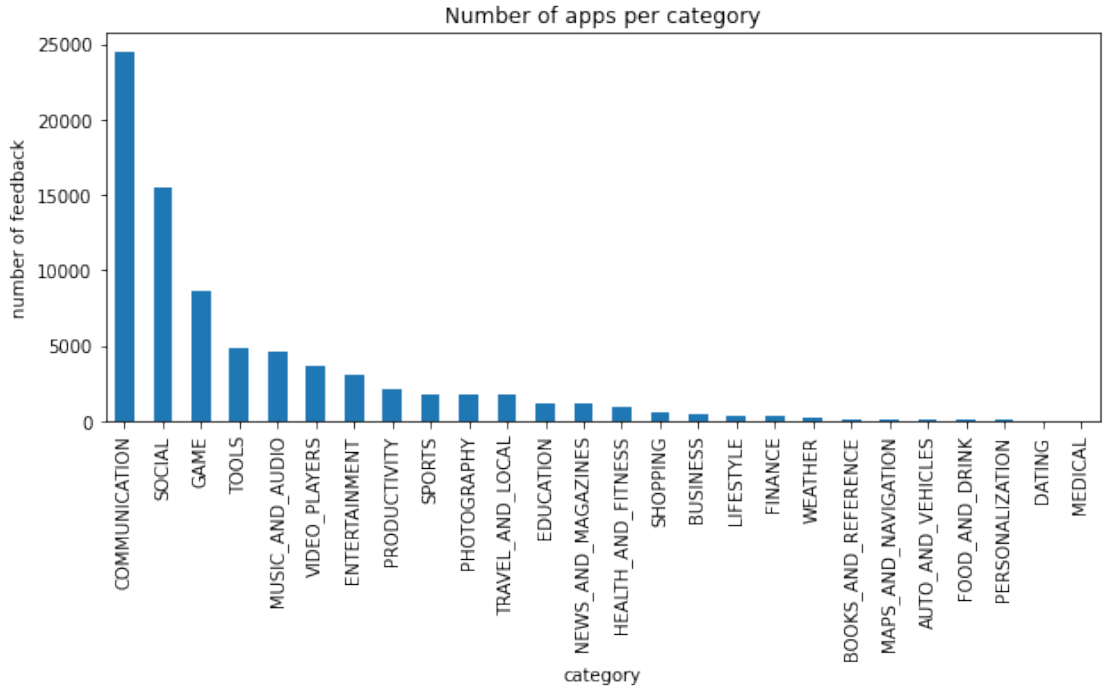

Figura 16: Numero di campioni per ogni categoria di applicazioni nel dataset My Digital Footprint

Le feature categoriche sono codificate con one-hot encoding, mentre le feature numeriche sono normalizzate. Il risultato è un vettore che contiene 63 feature che descrivono il contesto fisico dell'utente, e 8 feature che descrivono il contesto sociale dell'utente.

4.2.3 Feature degli oggetti

Category è la categoria delle applicazioni ottenuta dal Google Play Store. In MDF le applicazioni sono divise in 26 categorie che descrivono la loro funzionalità principale (es. videogiochi, notizie, social). Come si può vedere dalla Figura 16, anche in MDF il numero di campioni per ogni categoria è molto sbilanciato: la categoria Communication che comprende applicazioni di messaggistica come Whatsapp e Telegram ha più di 20k campioni. La seconda categoria più popolare è Social che comprende applicazioni come Facebook, Instagram e Twitter. La categoria è codificata con one-hot encoding.

4.2.4 Feature degli utenti

Come per Frappe, il dataset MDF non contiene nessuna informazione sugli utenti. Per questo motivo ho generato quattro feature utente dalle feature di contesto e degli oggetti.

User category indica la categoria di applicazioni più utilizzata dall'utente. Anche in questo caso le categorie più popolari sono Communication e Social.

User weekday indica il giorno della settimana in cui l'utente ha generato il maggior numero di feedback. I giorni più popolari sono giovedì e venerdì.

User daytime indica il momento della giornata in cui l'utente ha generato il maggior numero di feedback. Il momento della giornata più popolare è la mattina.

User weekend indica se l'utente ha generato più feedback in settimana o nel weekend. La maggior parte degli utenti (97%), ha generato più feedback in settimana.

Come le feature degli oggetti, anche le feature degli utenti sono tutte variabili categoriche. Vengono codificate con one-hot encoding in un vettore di 27 feature: 14 per `user category`, 7 per `user weekday`, 4 per `user daytime`, 2 per `user weekend`.

Ricapitolando, il dataset MDF processato contiene 73176 campioni, 31 utenti e 338 oggetti. Oltre alle colonne user, item e feedback, il dataset contiene 27 feature degli utenti, 26 feature degli oggetti, e 71 feature di contesto.

Capitolo 5

Risultati

In questo capitolo sono riportati i risultati sui dataset Frappe e MDF. Nella sezione 5.1 sono spiegati i modelli scelti per comparare le prestazioni di moveCARS e il motivo della selezione. Nella sezione 5.2 è spiegata la metrica utilizzata per la valutazione dei modelli e i vantaggi che comporta. Nelle sezioni 5.3 e 5.4 sono spiegati k-fold e grid search per identificare gli iperparametri migliori dei modelli. Nella sezione 5.5 sono riportati i risultati sui dataset presentati nel Capitolo 4, e in ultimo i tempi di esecuzione su dispositivi Android (sezione 5.6).

5.1 Modelli di confronto

Ho comparato il modello moveCARS con tre modelli collaborative-filtering già discussi nel Capitolo 2:

1. *ALS*: Ho scelto questo modello per valutare la differenza di prestazioni su task di classificazione tra un algoritmo più classico di matrix factorization, e i nuovi approcci basati su deep learning. Ho utilizzato l'algoritmo ALS della libreria Python Implicit[1]. Questo progetto open-source fornisce le implementazioni di diversi algoritmi popolari per dataset con feedback impliciti, con supporto a multi-threading su CPU e kernel GPU.

2. *NeuMF:* Ho scelto questo modello come algoritmo base di deep learning non context-aware. È stato utile per valutare l'impatto del contesto nella produzione di raccomandazioni, confrontandolo con la sua variante context-aware ECAM NeuMF e con moveCARS. È infatti l'unico algoritmo di deep learning selezionato senza informazioni di contesto. Ho implementato il modello in Python con la libreria Keras, seguendo il codice pubblicamente disponibile

[1] `https://github.com/benfred/implicit`

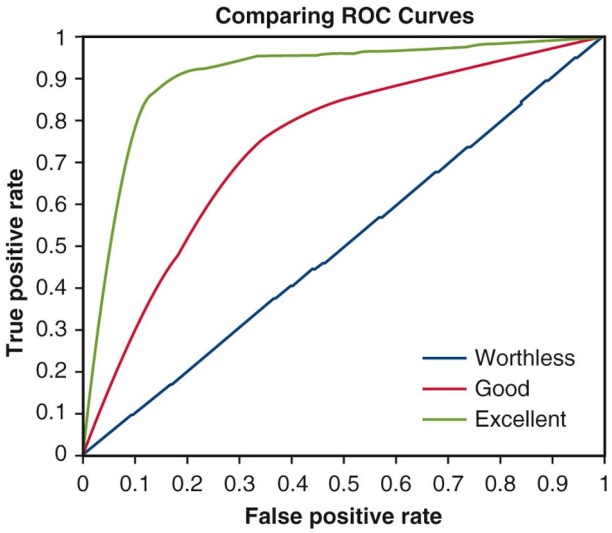

Figura 17: Curve ROC per tre classificatori con prestazioni diverse [43]

sul profilo GitHub di Xiangnan He[2], uno degli autori di [8] in cui il modello è stato proposto.

3. *ECAM NeuMF:* Delle tre estensioni context-aware di NeuMF (ECAM, UCAM, HCAM), ho scelto ECAM NeuMF che utilizza un vettore di contesto senza ulteriori elaborazioni. Nonostante ECAM NeuMF sia la versione meno performante delle tre varianti [24], è l'unica plausibile per essere implementata su dispositivi mobili. Questo modello è utile per un confronto diretto con moveCARS, dato che i due modelli utilizzano lo stesso insieme di feature di contesto. Gli autori del modello non hanno rilasciato il codice sorgente o dettagli sull'implementazione. Per questo motivo ho implementato ECAM NeuMF in Keras, a partire da NeuMF.

5.2 AUC

Una curva ROC (Receiver Operating Characteristic) è un grafico che mostra le prestazioni di un modello di classificazione binario a tutte le possibili soglie di classificazione. Questa curva traccia due parametri: (i) True Positive Rate (TPR), (ii) False Positive Rate (FPR). I due parametri sono definiti come:

$$TPR = \frac{TP}{TP + FN}$$

[2] `https://github.com/hexiangnan/neural_collaborative_filtering`

$$FPR = \frac{FP}{FP + TN}$$

in cui TP sono gli esempi positivi correttamente classificati come positivi, TN sono gli esempi negativi correttamente classificati come negativi, FP sono gli esempi negativi erroneamente classificati come positivi, FN sono gli esempi positivi erroneamente classificati come negativi. Una curva ROC traccia TPR e FPR a diverse soglie di classificazione. Abbassando la soglia di classificazione il modello classifica più oggetti come positivi, aumentando di conseguenza i falsi positivi e i veri positivi.

L'AUC è l'area sottostante alla curva ROC. Quest'area è sempre rappresentata da un valore tra 0 e 1, così come sia TPR sia FPR possono variare tra 0 e 1. L'obiettivo è cercare di massimizzare l'area, in modo tale da avere il più alto TPR possibile e il più basso FPR possibile per una data soglia. Ne consegue che un classificatore binario con una curva ROC uguale alla bisettrice del primo quadrante ha un AUC pari a 0.5 e risponde in modo casuale, mentre un classificatore con AUC pari ad 1 risponde in modo perfetto. Il valore dell'AUC può anche essere visto come la probabilità che il modello sia in grado di distinguere tra la classe positiva e la classe negativa. La Figura 17 mostra in blu la curva ROC di un classificatore casuale, in viola quella di un buon classificatore, e in verde quella di un classificatore ottimo.

L'AUC possiede tre proprietà fondamentali che la rendono un'ottima metrica per valutare le prestazioni generali di un classificatore binario [44]:

1. *Invarianza di scala:* L'AUC non dipende dalla scala delle predizioni. Moltiplicando l'output del modello per un fattore casuale, la forma della curva ROC e l'AUC non cambiano. Solo la soglia cambia modificando la scala.

2. *Invarianza alla soglia:* L'AUC misura la qualità delle predizioni del modello indipendentemente dalla soglia di classificazione scelta.

3. *Distribuzione delle classi:* L'AUC è insensibile a cambiamenti nella distribuzione delle classi. Per questo motivo è una metrica adatta a dataset leggermente sbilanciati come MDF e Frappe.

Per le proprietà elencate, ho scelto di usare l'AUC come metrica di paragone sia per ottimizzare gli iperparametri, sia per valutare le prestazioni dei modelli proposti nella sezione 5.1 rispetto a moveCARS.

5.2.1 AUC per ALS

Per calcolare l'AUC di una rete neurale si divide il dataset D in due sottoinsiemi D_{train} e D_{val}. Il modello è addestrato su D_{train} e valutato su D_{val}, calcolando

l'AUC in base ai valori predetti dalla rete y_{pred} e i valori reali y_{val} nel dataset D_{val}. Per gli algoritmi di matrix factorization come ALS, la divisione in train e test è effettuata mascherando una percentuale dei feedback positivi dalla matrice originale. L'AUC è poi calcolata sulla lista prodotta delle k migliori raccomandazioni (AUC@k), o sull'errore di ricostruzione della matrice prodotta da ALS rispetto alla matrice originale. Per confrontare ALS con i modelli di deep learning ho seguito l'approccio suggerito in [45], che punta a calcolare l'AUC solo per gli utenti che hanno avuto uno o più feedback mascherati, anziché su tutta la matrice. L'algoritmo 3 riporta lo pseudocodice della procedura. Gli input sono una matrice D_{train} che corrisponde alla matrice originale D con una percentuale di feedback positivi alterati in feedback negativi, D_{test} che è una matrice i cui feedback positivi corrispondono ai feedback mascherati da D_{train}, e D_{pred} che è la matrice ricostruita dal modello ALS dopo essere stato addestrato su D_{train}. L'output è l'AUC di ALS su D. L'algoritmo inizia creando una lista di tutti gli utenti che hanno almeno un feedback alterato (riga 1). Per ogni utente u nella lista degli utenti alterati (riga 2), si estraggono i vettori u_{train}, u_{test} e u_{pred} che corrispondono alla riga dell'utente u nelle matrici D_{train}, D_{test} e D_{pred} (righe 3, 4, 5). Successivamente viene generato un vettore di indici u_{idx} che denota le posizioni in u_{train} che contengono un feedback negativo (riga 6). Il motivo per cui si fa questo è che sono da escludere dal calcolo dell'AUC i feedback positivi noti in fase di training, i quali sono zero in u_{test}. Dei vettori u_{test} e u_{pred} vengono quindi considerati solo gli elementi in posizione u_{idx} (righe 7, 8). A questo punto viene calcolata con la funzione *auc_score*, l'AUC per l'utente u tra u_{test} e u_{pred} (riga 9). Il risultato u_{auc} è sommato all'AUC globale (riga 10). Calcolata l'AUC per tutti gli utenti u, l'AUC globale viene ritornata come media di tutte le AUC degli utenti (riga 12).

5.3 K-Fold Cross-Validation

La convalida incrociata è una delle tecniche di ricampionamento dei dati più utilizzata per stimare l'errore di predizione e regolare i parametri dei modelli. Nella k-fold cross-validation, il dataset D è partizionato in k insiemi disgiunti approssimativamente della stessa dimensione. In questo contesto, "fold" si riferisce al numero di sottoinsiemi che risultano dal partizionamento del dataset originale. Questo partizionamento viene eseguito selezionando casualmente esempi da D senza rimpiazzo. Il modello è addestrato utilizzando $k-1$ sottoinsiemi che congiuntamente rappresentano il training set. Successivamente, le prestazioni del modello sono calcolate sul sottoinsieme rimanente denotato come test set. Il processo è ripetuto fino a quando tutti i k sottoinsiemi sono stati usati come test set. La media delle k prestazioni misurate sui k test set costituisce il risultato della convalida incrociata [46]. La Figura 18 illustra il processo per $k = 10$, cioè 10-fold cross-validation.

Algoritmo 3 AUC per ALS

Input:
D_{train} - $users \times items$ matrix with masked user-item interactions
D_{test} - $users \times items$ matrix containing masked interactions from D_{train}
D_{pred} - $users \times items$ matrix containing predicted feedbacks
Output: AUC

$altered\_users \leftarrow$ list of users that have at least one item masked
for all $u \in altered\_users$ **do**
    $u_{train} \leftarrow$ user u feedbacks in D_{train}
    $u_{test} \leftarrow$ user u feedbacks in D_{test}
    $u_{pred} \leftarrow$ user u feedbacks in D_{pred}
    $u_{idx} \leftarrow$ indices where $u_{train} = 0$
    $u_{test} \leftarrow u_{test}$ elements with indices in u_{idx}
    $u_{pred} \leftarrow u_{pred}$ elements with indices in u_{idx}
    $u_{AUC} \leftarrow auc\_score(u_{test}, u_{pred})$
    $AUC = AUC + u_{AUC}$
end for
Return $AUC/altered\_user.length$

Nel primo fold, il primo sottoinsieme funge da test set D_{val} e i rimanenti nove sottoinsiemi costituiscono il training set. Nel secondo fold, il secondo sottoinsieme è il test set e i rimanenti sottoinsiemi sono il training set. Quando tutti i dieci sottoinsiemi sono stati utilizzati come test set il processo termina.

Non esiste una regola formale per selezionare il valore corretto di k, i valori più comuni sono $k = 5$ (l'80% dei dati è usato come training set), oppure $k = 10$ (il 90% dei dati è usato come training set). Al crescere di k la dimensione del training set aumenta, mentre la dimensione del test set diminuisce. In questa tesi per misurare le prestazioni dei modelli ho scelto $k = 10$.

5.4 Grid search

Un iperparametro di un modello è una caratteristica esterna al modello il cui valore non può essere stimato dai dati. Il valore dell'iperparametro va impostato prima di iniziare il processo di apprendimento. All'opposto, un parametro è una caratteristica interna al modello e il suo valore può essere stimato dai dati. La grid search è usata per trovare gli iperparametri ottimali di un modello che portano ad ottenere le predizioni più accurate.

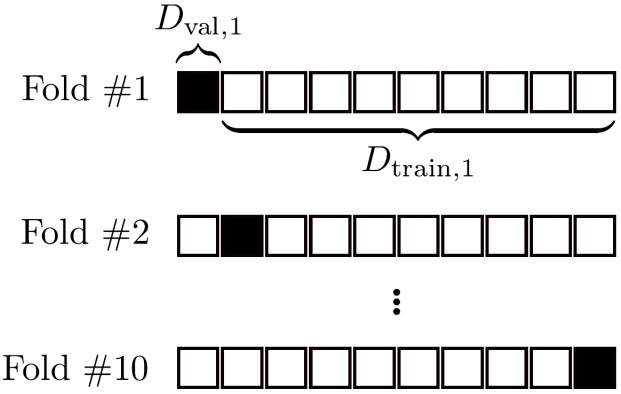

Figura 18: 10-fold cross-validation. Il dataset è diviso in modo casuale in dieci sottoinsiemi disgiunti, ognuno contenente il 10% dei dati
[46]

Uno spazio di ricerca è un volume in cui ogni dimensione rappresenta un iperparametro, ed ogni punto rappresenta una specifica configurazione del modello. Un punto nello spazio di ricerca è un vettore con una valore specifico per ogni iperparametro. La grid search è un processo che testa l'algoritmo selezionato in modo esaustivo, considerando un sottoinsieme definito manualmente dello spazio di ricerca. In pratica, per ogni iperparametro viene specificato manualmente un sottoinsieme di valori che può assumere, e la grid search testa l'algoritmo scelto considerando tutte le possibili combinazioni di iperparametri. In questa tesi, è stata scelta come combinazione di iperparametri migliore quella con cui il modello ottiene un valore di AUC più alto. Per maggiore robustezza, l'AUC è calcolata con 5-fold cross validation, scegliendo gli iperparametri che portano ad un AUC più alta su una media di 5 fold. La Tabella 1 mostra gli iperparametri selezionati per la calibrazione dei modelli, e i possibili valori che possono assumere. Le ultime due colonne riportano il risultato della grid search sui dataset MDF e Frappe.

5.5 Risultati sui dataset

In questa sezione sono riportati i risultati sui dataset MDF e Frappe. Come già spiegato, la metrica scelta per confrontare i modelli è l'AUC calcolata con 10-fold cross-validation. Gli iperparametri dei modelli sono impostati al valore migliore calcolato tramite grid search riportato nella Tabella 1.

Modelli	**Iperparametri**	**Valori**	**MDF**	**Frappe**
ALS	Fattori latenti	[64, 128, 256, 512]	128	64
	Regolarizzazione	[0.01, 0.1, 1, 5, 7, 10]	5	10
	Iterazioni	[1, 10, 50, 100, 200]	10	1
NeuMF	Epoche	[5, 10, 15, 20]	10	5
	Batch size	[64, 128, 256]	64	64
	Learn rate	[0.0001, 0.001, 0.005, 0.01]	0.001	0.001
ECAM NeuMF	Epoche	[5, 10, 15, 20]	10	5
	Batch size	[64, 128, 256]	256	128
	Learn rate	[0.0001, 0.001, 0.005, 0.01]	0.001	0.001
moveCARS	Epoche	[5, 10, 15, 20]	10	10
	Batch size	[64, 128, 256]	64	128
	Learn rate	[0.0001, 0.001, 0.005, 0.01]	0.005	0.001
	Layer nascosti	[3, 4, 5]	3	3
	Neuroni	[100, 200, 300]	100	200

Tabella 1: Spazio di ricerca della grid search e risultati su MDF e Frappe

5.5.1 Risultati MDF

I risultati su MDF sono divisi in tre parti: (i) confronto di moveCARS con gli altri modelli, (ii) analisi dell'importanza delle feature di contesto nel processo di raccomandazione, (iii) analisi dell'importanza di conoscere da chi è stata ricevuta una raccomandazione.

Confronto con altri modelli
La Figura 19 riporta un grafico a barre con i risultati sul dataset MDF. Per prima cosa, si può notare come i modelli basati su reti neurali (NeuMF, ECAM NeuMF e moveCARS), hanno un AUC superiore rispetto all'unico sistema di raccomandazione non neurale ALS. Secondariamente, si può vedere come i modelli context-aware (ECAM NeuMF e moveCARS) sono superiori ai modelli non-context aware (ALS e NeuMF). In particolare ECAM NeuMF ha un AUC più alta del 6,15% rispetto alla controparte non-context aware NeuMF, dimostrando che integrare informazioni contestuali nel processo di raccomandazione può migliorare la qualità delle raccomandazioni. Il modello moveCARS si è rivelato leggermente inferiore rispetto a ECAM NeuMF, ma allo stesso tempo ha un AUC superiore del 5,33% rispetto a NeuMF. Nonostante moveCARS non si sia rivelato il modello migliore ci sono due considerazioni importanti da fare: (i) moveCARS non soffre delle restrizioni imposte dalla struttura della rete come ECAM NeuMF (**??**), (ii) il dataset MDF contiene poche informazioni associate agli utenti e oggetti, non è da escludere un miglioramento delle prestazioni con l'utilizzo di feature più descrittive.

Confronto sulle feature di contesto
La Figura 20 riporta un grafico a barre con i risultati di moveCARS addestrato su sottoinsiemi diversi delle feature di MDF, per valutare l'impatto del contesto sulle raccomandazioni. In particolare viene addestrato con le seguenti combinazioni di feature: (i) solo feature di utenti e oggetti, (ii) feature di utenti, oggetti e contesto sociale, (iii) feature di utenti, oggetti e contesto fisico, (iv) feature di utenti, oggetti, contesto fisico e contesto sociale. Si può notare che il modello addestrato solo su feature di utenti e oggetti ha un AUC molto bassa, pari a 0.7119. Cioè è dovuto alla scarsità di feature che non descrivono in modo esaustivo gli utenti e gli oggetti. Aggiungendo il contesto sociale, che indica il tipo di persone in prossimità dell'utente nel momento in cui ha utilizzato un'applicazione, l'AUC aumenta a 0.7767. Ovviamente le informazioni di contesto sociale non sono sufficienti a descrivere interamente la situazione contestuale di un utente. Aggiungendo il contesto fisico alle feature di utenti e oggetti, l'AUC aumenta a 0.9583. Ciò dimostra ancora una volta che il contesto ha un effetto molto rilevante sulle raccomandazioni. Combinando il contesto fisico è sociale si ottiene l'AUC migliore, pari a 0.9731. Questa

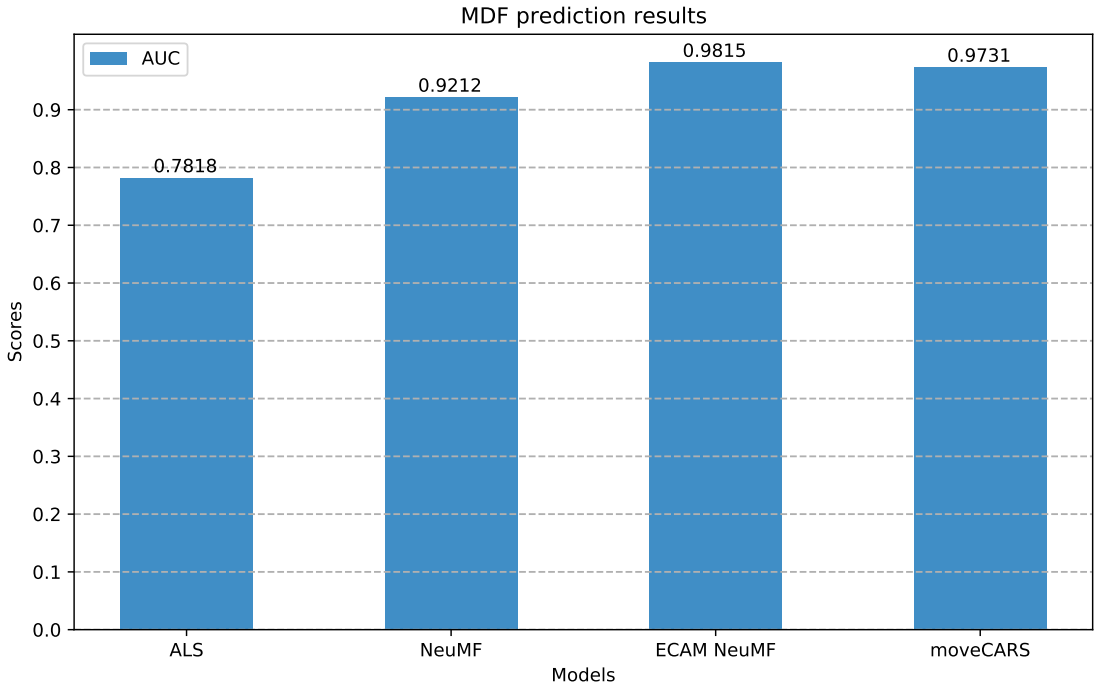

Figura 19: Risultati sul dataset MDF

è un'indicazione che utilizzare un numero elevato di informazioni contestuali può migliorare ulteriormente le raccomandazioni.

Confronto sulla feature sociale layer

Nei test precedenti come feature di contesto sociale sono state considerate solo `social_c1, social_c2, social_c3, social_c4` che descrivono gli alter in prossimità dell'utente nel momento in cui ha utilizzato un'applicazione. Non è stata considerata la feature `layer` che indica la cerchia sociale dell'utente da cui è stata ricevuta una raccomandazione. Questo perchè aggiungere questa nuova feature significa dover generare un dataset diverso e personale per ogni utente, in cui ogni riga contiene un valore che indica il layer dell'utente da cui è stata ricevuta la raccomandazione. Come ultimo test ho quindi generato un dataset per ogni utente (31 dataset), ogni dataset contiene tutti i campioni disponibili e tutte le feature già descritte, con l'aggiunta della feature `layer`. Per la valutazione, moveCARS è addestrato su tutti i dataset degli utenti, e viene memorizzata l'AUC ottenuta su ogni dataset. Di questi valori viene fatta la media per calcolare il risultato finale. L'AUC media è pari a 0.9822, superiore rispetto all'AUC di moveCARS calcolata precedentemente (0.9731), e superiore all'AUC di ECAM NeuMF (0.9815).

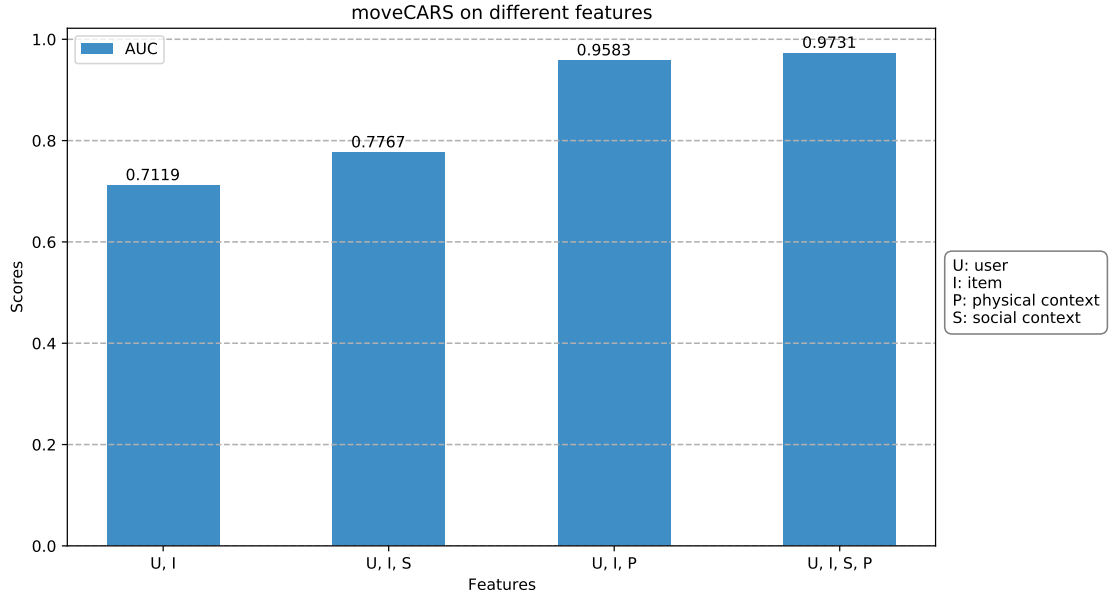

Figura 20: Risultati di moveCARS sul dataset MDF. Il modello è addestrato su feature differenti per valutare l'impatto del contesto sulle raccomandazioni

5.5.2 Risultati Frappe

In Figura 21 è riportato un grafico a barre con i risultati dei modelli sul dataset Frappe. Come per MDF l'algoritmo peggiore è ALS con un AUC pari a 0.7576. NeuMF ha un AUC migliore di ALS pari a 0.8143. Nonostante Frappe sia un dataset con poche informazioni contestuali, il contesto aiuta comunque nel processo di raccomandazione. ECAM NeuMF infatti ha un AUC più alta del 3,16% rispetto a NeuMF. La differenza meno marcata rispetto al dataset MDF si può spiegare nella minore dimensionalità del contesto di Frappe. MoveCARS ha ottenuto un risultato migliore di NeuMF, con un AUC superiore del 2,46%, e leggermente inferiore a ECAM NeuMF. Di nuovo si può notare come con meno restrizioni sull'input moveCARS riesce ad ottenere risultati molto simili a ECAM NeuMF.

5.6 Test su smartphone

L'ultima parte di questo capitolo è dedicata a valutare se il modello moveCARS e gli altri algoritmi di deep learning proposti come confronto possano essere eseguiti su dispositivi mobili, e se ci siano differenze significative in termini di tempo di esecuzione. Non è stata valutata la fase di addestramento dei modelli, che è rimandata agli sviluppi futuri, ma solo il tempo di inizializzazione e di inferenza.

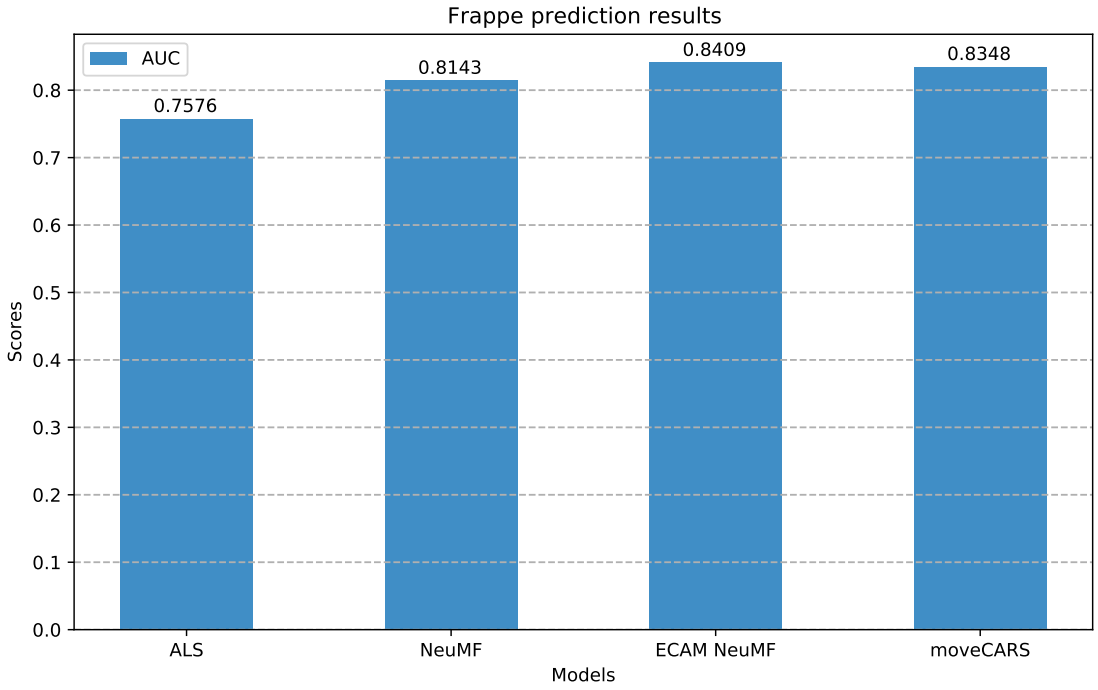

Figura 21: Risultati sul dataset Frappe

Dato che i modelli selezionati sono scritti in Keras con Tensorflow come backend, ho scelto di ricorrere al framework TensorFlow Lite.

5.6.1 TensorFlow Lite

TensorFlow Lite[3] (TF Lite) è un framework di deep learning open-source, multi-piattaforma, distribuito con TensorFlow 2.0.. TF Lite permette di convertire un modello TensorFlow precedentemente addestrato in un formato speciale che può essere ottimizzato in velocità e memoria occupata. Questo formato speciale può essere distribuito su smartphone con Android o iOS, o dispositivi basati su Linux come il Raspberry Pi e altri microcontrollori.

TF Lite contiene principalmente due componenti fondamentali: il convertitore (TF Lite Converter) e l'interprete (TF Lite Interpreter) che può essere installato indipendentemente sul dispositivo su cui si vuole fare inferenza. Il convertitore ha il compito principale di ottimizzare il modello riducendo le sue dimensioni e aumentando la sua velocità di esecuzione. Sono disponibili diverse ottimizzazioni, ad esempio è possibile ridurre la precisione del modello convertendo tutti i pesi

[3] `https://www.tensorflow.org/lite`

del modello da float a 32 bit a interi a 8 bit, a discapito naturalmente dell'accuratezza delle inferenze. Nei test effettuati in questa tesi sono state mantenute le impostazioni predefinite di TF Lite che riducono le dimensioni del modello senza quantizzazione dei pesi o delle attivazioni.

La conversione di un modello TensorFlow 2.0 in un modello TF Lite compatibile con dispositivi Android si compone di quattro passaggi:

1. *Fase di addestramento:* Il modello TensorFlow viene addestrato su dispositivo fisso con un dataset, allo stesso modo in cui si procede usualmente.

2. *Salvataggio del modello:* Il modello viene serializzato in un singolo file che contiene i pesi, i bias, e la configurazione di training del modello.

3. *Conversione del modello:* Il file salvato è dato in input al convertitore TF Lite che converte il modello TensorFlow in un modello TF Lite applicando le ottimizzazioni selezionate, e lo salva in un nuovo file con estensione tflite.

4. *Copia sul dispositivo:* Il file tflite viene copiato sul dispositivo mobile con Android Debug Bridge (adb), o con un semplice copia-incolla.

A questo punto si può utilizzare l'applicazione Android BenchmarkModel[4] per valutare i tempi di inizializzazione e inferenza dei modelli. Questa applicazione genera casualmente dei campioni compatibili con l'input del modello TF Lite e monitora il tempo che il modello impiega per inizializzare i propri parametri ed essere pronto per fare inferenza, e il tempo effettivo di inferenza sui dati generati casualmente. Ci sono alcuni parametri che possono essere configurati prima di eseguire il benchmark, tra cui:

- `num_thread`
  Il numero di thread usati per eseguire l'interprete di TF Lite. Ho usato il valore 1 in modo da evitare il multithreading.

- `num_benchmark`
  Il numero di benchmark eseguiti. Ogni benchmark è la media di `num_runs` esecuzioni. Ho usato un numero di benchmark pari a 10.

- `num_runs`
  Il numero di esecuzioni in un benchmark. Aumentare questo valore riduce la varianza dei tempi di inizializzazione e inferenza. Ho usato un numero di esecuzioni pari a 1000.

[4] `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/benchmark`

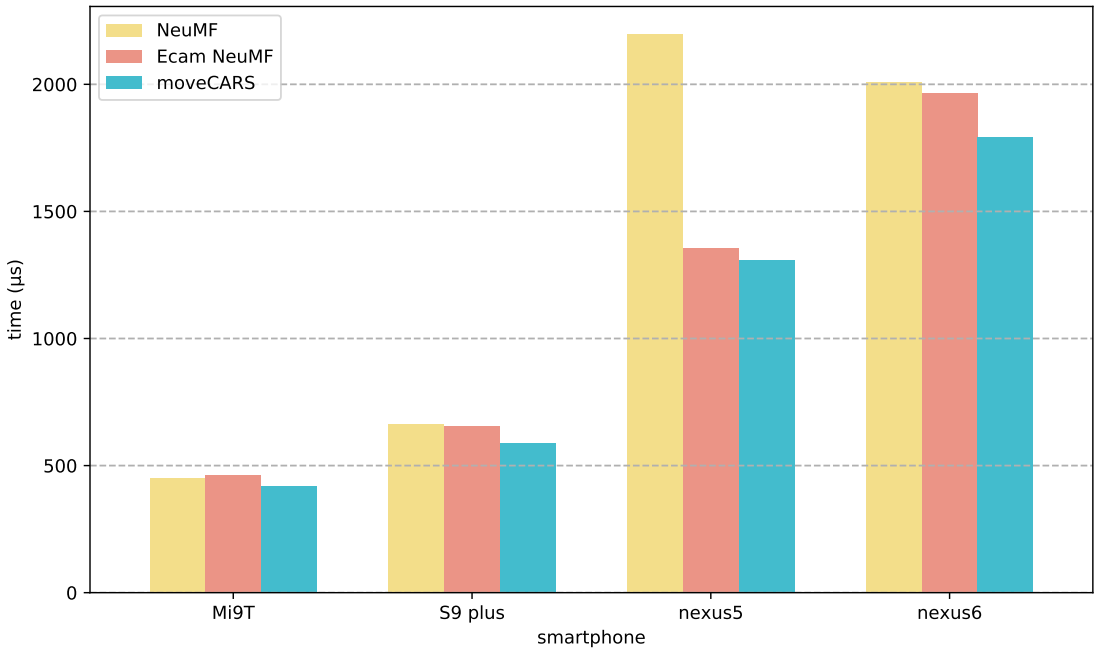

Figura 22: Tempo di inizializzazione in microsecondi dei modelli di deep learning su dispositivi Android

- `use_gpu`
  Valore booleano che permette di selezionare se usare oppure no la GPU del dispositivo Android. Per ragioni di stabilità ho impostato il valore su falso.

5.6.2 Risultati benchmark su Android

Per una valutazione accurata, ho usato quattro diversi smartphone Android su cui sono stati misurati i tempi di inizializzazione e inferenza dei modelli usando l'applicazione Android BenchmarkModel:

- *Google Nexus 5:* È un telefono rilasciato sul mercato a fine 2013 dotato di processore Qualcomm Snapdragon 800 e 2GB di RAM. Monta la versione di fabbrica di Android rilasciata da Google, e non ha installate applicazioni addizionali che potrebbero influire sui test.

- *Google Nexus 6:* È un telefono rilasciato sul mercato a fine 2014 dotato di processore Qualcomm Snapdragon 805 e 3GB di RAM. Come il Nexus 5, anche il Nexus 6 monta la versione di fabbrica di Android, e non ha installate applicazioni aggiuntive.

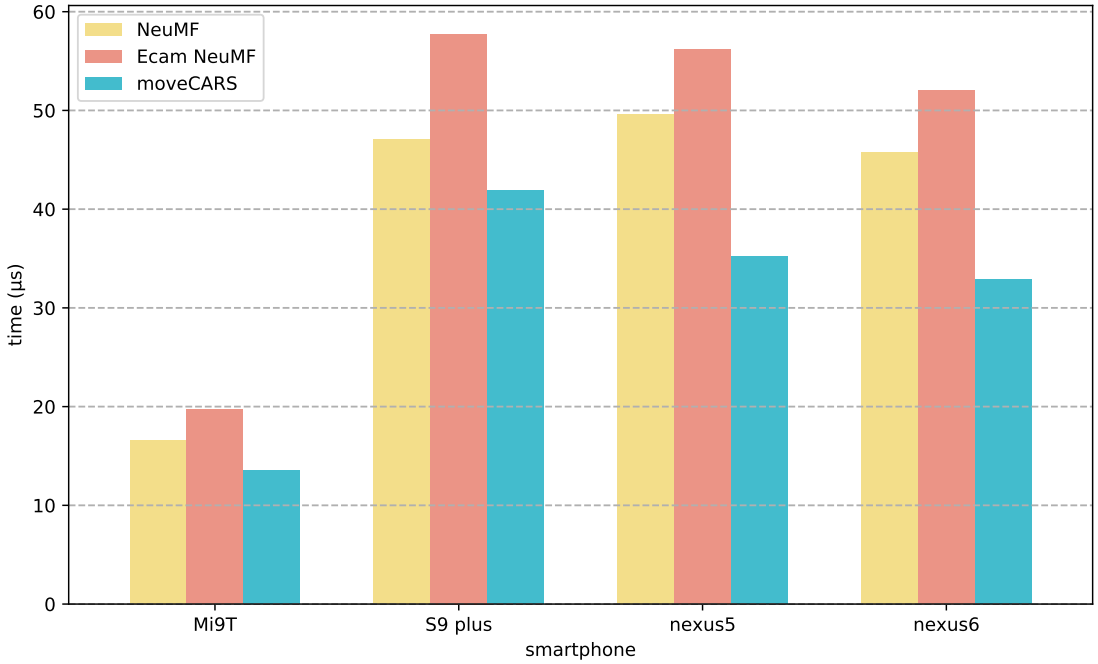

Figura 23: Tempo di inferenza in microsecondi dei modelli di deep learning su dispositivi Android

- *Samsung Galaxy S9 Plus:* É un telefono rilasciato sul mercato a inizio 2018 dotato di processore Samsung Exynos 9810 e 6GB di RAM. Monta una versione Android modificata da Samsung chiamata Samsung Experience. È l'unico tra i dispositivi usati per il confronto ad avere un processore diverso dai Qualcomm Snapdragon.
- *Xiaomi Mi9T:* È un telefono rilasciato sul mercato a inizio 2019 dotato di processore Qualcomm Snapdragon 730 e 6GB di RAM. Monta una versione Android modificata da Xiaomi chiamata MIUI. È il dispositivo più recente tra quelli usati per il benchmark.

In Figura 22 è mostrato un grafico a barre con il tempo di inizializzazione dei modelli basati su deep learning sui quattro smartphone appena citati. Il grafico riporta il tempo di inizializzazione in microsecondi, calcolato come la mediana tra 10 benchmark. Ho usato la mediana al posto della media per eliminare i valori anomali causati probabilmente dall'esecuzione di altri processi o applicazioni sul dispositivo durante la fase di benchmark. Tutti gli algoritmi di deep learning sono molto veloci nella fase di inizializzazione, con un tempo mediano che varia tra 418μs e 2196,50μs. I tempi di inizializzazione più bassi sono ottenuti su Mi9T e

S9 Plus che sono i dispositivi più recenti. MoveCARS è leggermente più veloce di NeuMF e ECAM NeuMF su tutti e quattro i dispositivi testati.

In Figura 23 è mostrato un grafico a barre con il tempo di inferenza su dispositivi mobili. Anche in questo caso il grafico riporta i tempi di inferenza in microsecondi calcolati come la mediana tra 10 benchmark. I tempi di inferenza sono molto più bassi di quelli di inizializzazione, con il tempo mediano per una singola inferenza che varia tra $13.50\mu s$ e $57.74\mu s$. Ancora una volta il dispositivo più veloce è il Mi9T, mentre gli altri smartphone si attestano su risultati simili. Per quanto riguarda i modelli di deep learning, anche in fase di inferenza moveCARS è più veloce, seppure si sta parlando di differenze di pochi microsecondi.

In conclusione, considerando i risultati dei test effettuati su smartphone con hardware diverso tra loro, e rilasciati in anni diversi si può affermare che tutti i modelli sono in grado di eseguire in real-time su tutti i dipositivi Android testati, con moveCARS che si è rivelato il modello più leggero e più veloce sia in fase di inizializzazione che di inferenza.

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

[6] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000, 2010.

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

[8] 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.

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

[10] Simon Funk. Netflix update: Try this at home. `https://sifter.org/~simon/journal/20061211.html`, 2006.

[11] Yancheng Jia, Changhua Zhang, Qinghua Lu, and Peng Wang. Users' brands preference based on svd++ in recommender systems. In *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, pages 1175–1178, 2014.

[12] 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.

[13] Christopher C. Johnson. Logistic matrix factorization for implicit feedback data. 2014.

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

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

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

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

[18] 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.

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

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

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

[22] Linas Baltrunas, Bernd Ludwig, and Francesco Ricci. Matrix factorization techniques for context aware recommendation. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, page 301–304, New York, NY, USA, 2011. Association for Computing Machinery.

[23] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, page 79–86, New York, NY, USA, 2010. Association for Computing Machinery.

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

[25] Casper Hansen, Christian Hansen, Lucas Maystre, Rishabh Mehrotra, Brian Brost, Federico Tomasi, and Mounia Lalmas. Contextual and sequential user embeddings for large-scale music recommendation. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 53–62, New York, NY, USA, 2020. Association for Computing Machinery.

[26] Jiwon Hong, Won-Seok Hwang, Jin-Hyung Kim, and Sang-Wook Kim. Context-aware music recommendation in mobile smart devices. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, page 1463–1468, New York, NY, USA, 2014. Association for Computing Machinery.

[27] Hongzhi Yin, Yizhou Sun, Bin Cui, Zhiting Hu, and Ling Chen. Lcars: A location-content-aware recommender system. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 221–229, New York, NY, USA, 2013. Association for Computing Machinery.

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

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

[30] 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.

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

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

[33] 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.

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

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

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

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

[38] 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.

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

[40] Unsupervised modelling of the user's social context and visited locations at the edge of the internet.

[41] Linas Baltrunas, Karen Church, Alexandros Karatzoglou, and Nuria Oliver. Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild. 05 2015.

[42] Mattia G. Campana and Franca Delmastro. Mydigitalfootprint: an extensive context dataset for pervasive computing applications at the edge. 2021.

[43] Victor A. Ferraris. Commentary: Should we rely on receiver operating characteristic curves? from submarines to medical tests, the answer is a definite maybe! *The Journal of Thoracic and Cardiovascular Surgery*, 157(6):2354–2355, 2019.

[44] Rahul Agarwal. An understandable guide to roc curves and auc and why and when to use them? `https://towardsdatascience.com/an-understandable-guide-to-roc-curves-and-auc-and-why-and-when-to-use-them-92020bc4c5c1`, 2021.

[45] Jesse Steinweg-Woods. A gentle introduction to recommender systems with implicit feedback. `https://jessesw.com/Rec-System/`, 2018.

[46] Daniel Berrar. *Cross-Validation.* 01 2018.