Collaborative Filtering: introduzione e confronto fra algoritmi

Dipartimento di Ingegneria Enzo Ferrari
Corso di Laurea in Ingegneria Informatica

Candidato
Noemi Di Cicco
Matricola 256861

Relatore
Prof. Nicola Bicocchi

Anno Accademico 2020/2021

Collaborative Filtering: introduzione e confronto fra algoritmi
Tesi di Laurea. Università di Modena e Reggio Emilia

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: 246411@studenti.unimore.it

Alle conclusioni e alle nuove avventure

Indice

Capitolo 1

Introduzione

I meccanismi di predizione delle preferenze di un utente, nell'ambito di applicazioni web e mobile, sono un concetto relativamente nuovo ma certamente molto diffuso; lo scopo di questa tesi è quello di descrivere e mettere a confronto diversi tipi di algoritmi al fine di individuare la scelta ottimale per contesti differenti. I suddetti algoritmi si differenziano tra loro sotto diversi punti di vista, in particolare per l'approccio utilizzato e la famiglia di appartenenza.

Ai fini di una migliore comprensione della valutazione delle performance negli ambiti proposti nel capitolo conclusivo, si può trovare nei primi capitoli una descrizione sufficientemente dettagliata del concetto di Collaborative Filtering (Capitolo 2) e delle due famiglie di algoritmi che esso comprende (Capitolo 4). Nonostante questo documento risulti abbastanza dettagliato nelle definizioni, ci sono alcune conoscenze che vengono sottintese.

- **Python** La conoscenza del linguaggio di programmazione Python è rilevante, in quanto tutti gli algoritmi citati sono implementati utilizzando questo linguaggio.
- **Numpy** La conoscenza della libreria matematica Numpy di Python può risultare utile nella comprensione di alcune funzioni presenti nell'implementazione della fattorizzazione delle matrici.

- **Operazioni tra matrici** Le matrici e le operazioni che si possono eseguire utilizzandole sono molto utili ai fini della comprensione degli algoritmi Model-Based.
- **Conoscenze geometrico-matematiche** Utili per poter comprendere le formule utilizzate nell'implementazione degli algoritmi Memory-Based.

Capitolo 2

Collaborative Filtering

Nel presente capitolo vengono definiti i concetti fondamentali affrontati all'interno dell'intero documento, essenziali per comprendere al meglio l'analisi delle performance che rappresenta la conclusione della tesi.

2.1 Premesse

Negli ultimi tempi, Internet è diventato un nuovo metodo di approccio al mondo del commercio e del marketing, mettendo a disposizione piattaforme, gratuite o a pagamento, che forniscono servizi di diverso genere; esempi rilevanti sono quelli di Amazon.com e Netflix.com.

L'aspetto fondamentale riguarda certamente l'enorme differenza con i corrispettivi negozi fisici in materia di quantità di opzioni disponibili; per questo motivo, per incontrare le esigenze degli utenti, si è resa necessaria l'introduzione di sistemi di filtering delle informazioni sulla base degli interessi degli utenti stessi: i **Recommender Systems**.

Questi ultimi possono essere costruiti utilizzando due approcci differenti: il **Collaborative Filtering** e il **Content-Based Filtering**. Lo scopo di queste due tecniche è similare, ovvero ottenere suggerimenti per l'utente in modo che esso possa reperire in maniera semplice e veloce oggetti o servizi di suo interesse, trascorrendo,

di conseguenza, più tempo sulle piattaforme coinvolte. La differenza tra i due approcci invece, si trova nel procedimento; nel primo caso si predicono le preferenze dell'utente sulla base delle sue interazioni sulla piattaforma, mentre nel secondo caso si calcolano sulla base di azioni, informazioni fornite dall'utente stesso o feedback espliciti.
Nel documento che segue verrà analizzato unicamente l'approccio del Collaborative Filtering.

2.2 Definizione di Collaborative Filtering

La supposizione implicita su cui si basa l'approccio del Collaborative Filtering è la seguente:

> *Se un utente A ha la stessa opinione di un utente B su un certo argomento, è più plausibile che A abbia la stessa opinione di B su un differente argomento rispetto ad un utente scelto in maniera casuale.* [8]

Come si evince da questa definizione, il termine "collaborative" dipende da una collaborazione sotto più punti di vista, che avviene utilizzando le caratteristiche di users o items simili; il termine "filtering" invece, fa riferimento al fine per cui avviene questa collaborazione, cioè quello di riuscire a predire in maniera automatica le preferenze di un certo utente.

Risulta evidente il fatto che ai fini di una predizione più accurata sia necessario essere in possesso di quanti più dati possibili, in questo modo sarà più semplice trovare item o user molto simili tra loro.
Essendo questo un ambiente così ampio e complesso, esistono diversi tipi di algoritmi, basati su approcci differenti tra loro, per poter applicare le tecniche sopra descritte; in particolar modo, distinguiamo due famiglie di algoritmi: **Memory-Based** e **Model-Based**.

Come evidenziato nello schema che segue (Figura 2.1), il primo approccio ricerca affinità tra utenti o tra items grazie all'utilizzo di operazioni aritmetiche all'interno

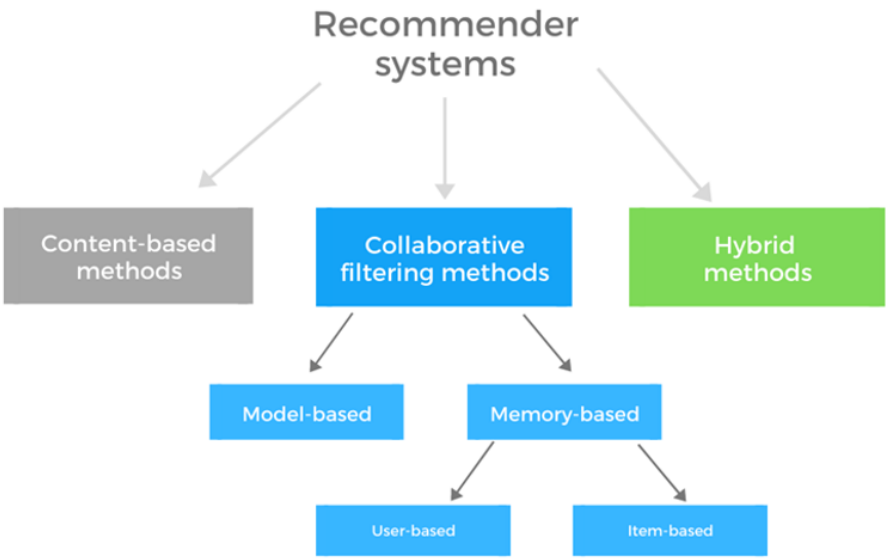

Figura 2.1. Diagramma esplicativo sulle famiglie di algoritmi del CF

di formule specifiche; nel secondo caso invece ci si serve del machine learning per la memorizzazione di parametri inerenti al processo di training dell'algoritmo, per permettere a quest'ultimo una maggiore accuratezza nei calcoli delle predizioni.

Per quanto riguarda la scelta di utilizzo dell'uno o dell'altro approccio, risulta necessario comprendere il contesto di applicazione, poiché entrambi presentano vantaggi e svantaggi. In generale, il Memory-Based si presta molto più alla facilità di creazione e esposizione dei risultati, a discapito delle prestazioni nel caso di pochi dati; il Model-Based invece, risulta molto più complesso a livello di applicazione e comprensione dei risultati ma è certamente più efficiente ed efficace, in particolare quando si lavora con pochi dati.

Proprio per queste caratteristiche, molto spesso vengono proposti degli ibridi tra i due approcci, per ottenere il meglio da entrambi. Si inizia con un training dell'algoritmo tramite un Model-Based, poiché nella prima fase non si hanno molti dati a disposizione, in secondo luogo si introduce un Memory-Based per favorire la velocità delle operazioni e, successivamente, la leggibilità dei risultati.

2.3 Approccio Memory-Based

Il Memory-Based prevede l'applicazione di tecniche statistiche sull'intero dataset per effettuare le predizioni; esso è divenuto molto popolare grazie al suo essere intuitivo e semplice, sia per quanto riguarda l'implementazione, sia per la comprensione immediata dei risultati della raccomandazione.
Questo approccio è costituito da tre componenti fondamentali:

- **Preprocessing dei dati**

  I dati vengono preprocessati per normalizzate i ratings, rimuovendo caratteristiche latenti non controllabili da questo tipo di approccio (ad esempio un'indole critica dell'utente).

- **Selezione del neighborhood**

  Consiste nel selezionare un certo numero di user considerati simili allo user scelto, questo avviene tramite diversi metodi per il calcolo della similarità.

- **Computazione della predizione**

  Essa genera predizioni e aggrega un certo numero di item in maniera decrescente per similarità all'utente di partenza.

2.3.1 Similarità

È necessario sottolineare come la selezione del neighborhood ricopra un ruolo molto significativo, poiché l'accuratezza con cui si determina la similarità permetterà risultati più o meno attendibili; la selezione può avvenire secondo due diversi tipi di interazioni:

- **Similarità user-user**

  Dato un utente, cerca altri utenti simili a lui sulla base di ratings simili, consigliandogli poi items simili a quelli piaciuti agli utenti simili a lui.

- **Similarità item-item**

  Dato un item, piaciuto ad un certo utente, si ottengono gli utenti a cui è

piaciuto l'item in questione, per poi risalire ad altri item piaciuti a questi utenti, in modo da poterli consigliare all'utente iniziale.

2.3.2 Calcolo della similarità

A livello grafico, si tratta di rappresentare diversi punti che costituiscono gli users; le loro coordinate sono determinate dai ratings che hanno attribuito ad ogni item. Per determinare la similarità ci si può basare sulle distanze tra questi punti, ad esempio usando la formula per la distanza euclidea tra due punti, oppure, per essere più accurati, si può calcolare il coseno dell'angolo tra le coppie di punti (più l'angolo è piccolo più la compatibilità e il coseno dell'angolo sono alte e viceversa).

In ogni caso, tramite questi approcci grafici, è possibile ottenere similarità tra due user che hanno registrato ratings molto differenti, si parla perciò di un'indole critica per l'utente che ha assegnato ratings più bassi, definito come **tough rater**. Per l'approccio Memory-Based tuttavia, risulta necessaria la rimozione di questi fattori latenti per poter procedere ad una valutazione della similarità più accurata. Seguono alcuni approcci molto diffusi per il calcolo della similarità [4].

Cosine Similarity

Questa metrica può essere pensata geometricamente se si tratta la riga della matrice dei ratings dell'utente come un vettore e se si tratta allo stesso modo la colonna degli items.

Per lo user-based CF, la similarità tra due utenti è data dal coseno dell'angolo tra i loro vettori.
Per gli utenti *u* e *u'*, la similarità del coseno è calcolata come:

$$sim(u, u') = cos(\theta) = \frac{r_u \cdot r_{u'}}{\|r_u\| \cdot \|r_{u'}\|} = \sum_i \frac{r_{ui} \cdot r_{u'i}}{\sqrt{\sum_i r_{ui}^2} \cdot \sqrt{\sum_i r_{u'i}^2}} \tag{2.1}$$

Per predire il rating dell'utente u riguardo un certo item i possiamo calcolare la sommatoria pesata dei ratings dell'item i ricevuti da tutti gli utenti u', dove il peso attribuito ad ogni rating dipende dalla similarità tra u e u'.
Normalizzando, otteniamo il rating effettivo:

$$\hat{r}_{ui} = \frac{\sum_{u'} sim(u, u') \cdot r_{u'i}}{\sum_{u'} |sim(u, u')|} \tag{2.2}$$

Pearson Similarity

Il Coefficiente di Correlazione di Pearson (PCC) è la misura di similarità più diffusa ed efficace; essa mostra la correlazione lineare tra user/item ed è rappresentata dal rapporto tra la covarianza di due utenti e la deviazione standard tra di essi. La formula per effettuare tale calcolo è la seguente:

$$similarity(A, B) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}}$$

2.3.3 Predizione dei ratings

Una volta determinati gli user simili ad uno user u è necessario predire i ratings che esso attribuirà a determinati items. Per farlo, esistono due approcci molto importanti, matematicamente molto simili ma differenti concettualmente: Item-Based e User-Based.

User-Based

Questa modalità consiste nell'utilizzo della matrice dei ratings per trovare utenti simili a quello di partenza, sulla base dei ratings che essi hanno attribuito; in particolare, la definizione è la seguente:

> Dato un utente U con un set di utenti simili a lui, determinati sulla base di un vettore di ratings attribuiti ad alcuni items, il rating per l'item I, che non è stato valutato dall'utente U, si ricava scegliendo N utenti dalla lista di similarità che hanno valutato l'item I e utilizzando i loro ratings per ottenere il rating dell'utente considerato inizialmente.

Item-Based

A differenza dell'approccio precedente, in questo caso la matrice di ratings viene utilizzata per trovare items simili sulla base dei ratings che essi hanno ricevuto dagli utenti, in maniera più precisa:

> Dato un item I con un set di items simili a lui, determinati sulla base di un vettore di ratings ricevuti da alcuni users, il rating dell'user U, che non ha valutato l'item I, si ricava scegliendo N items dalla lista di similarità che hanno ricevuto la valutazione dall'user U e utilizzando i loro ratings per ottenere il rating dell'item considerato inizialmente.

Questo tipo di Collaborative Filtering è stato introdotto da Amazon; infatti in un contesto dove ci sono più users che items, l'item-based risulta un approccio più efficace, in quanto più stabile e veloce, di un user-based.

2.4 Approccio Model-Based

Questa categoria introduce lo step di ridurre o comprimere la matrice user-item, ampia ma poco popolata, utilizzando, nella maggior parte dei casi, algoritmi di machine learning per la generazione dei ratings.
Questi algoritmi dovranno affrontare una fase di training per poter essere automatizzati e contribuire a rendere ottimizzate le predizioni per le quali vengono impiegati.
La suddivisione principale per questo tipo di tecnica riguarda diverse famiglie di algoritmi: Lo schema che segue evidenzia la suddivisione interna all'approccio Model-Based.

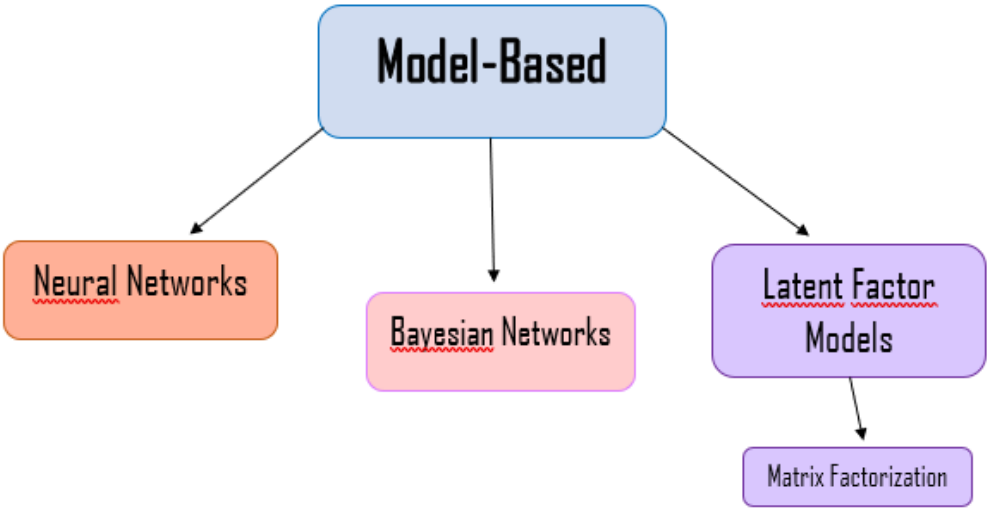

Figura 2.2. Suddivisione famiglie di algoritmi del CF Model-Based

2.4.1 Bayesian Networks

Dal punto di vista probabilistico, i compiti del CF possono essere visti come il calcolo del valore atteso per il rating di un utente su un certo item, date le informazioni che abbiamo sugli altri ratings dati dall'utente.
Nella formula che segue p indica la probabilità, $r\ (k,j)$ costituisce il rating per l'item

j, dati i precedenti ratings *r(k)*, *x*, infine, costituisce il valore del rating scelto in un intervallo *r(min)*,*r(max)*.

$$\nu_{k,j} = \sum_{s} p(r_{k,j} = x | r_k) \cdot x \tag{2.3}$$

Breese, Heckerman e Kadie [5] introduco due differenti modelli probabilistici per la computazione delle informazioni. Nel primo algoritmo, gli users sono raggruppati utilizzando la probabilità condizionata di Bayes, basata sull'idea che esistono determinati gruppi di utenti che condividono preferenze; questa computazione viene solitamente effettuata da un algoritmo di massimizzazione delle aspettative (EM algorithm).

Per quanto riguarda il secondo invece, esso si basa sul modello di reti di Bayes, dove ogni item nel database è modellato come un nodo avente stati corrispondenti ai ratings che ha ricevuto.

2.4.2 Latent Factor Models

I modelli a fattore latente costituiscono l'approccio più diffuso ed utilizzato per effettuare predizioni; in particolar modo, la tecnica della fattorizzazione delle matrici risulta essere la più esplicativa per una comprensione generale di questo tipo di algoritmi.

Matrix Factorization

Le sfide maggiori per gli algoritmi di Collaborative Filtering riguardano in particolare la scalabilità e la carenza di dati; per ovviare a questi problemi è stato introdotto l'approccio della fattorizzazione delle matrici.
Esso implica la suddivisione di una matrice, grande e poco popolata, in sottomatrici, con una dimensione minore e quindi una maggiore densità di dati, un esempio potrebbe essere la suddivisione in una matrice per gli users e una per gli items. In

altre parole, la scomposizione delle matrici può essere riformulata come un problema di ottimizzazione con perdita di vincoli e di funzioni [3].

L'idea dietro questo modello è che l'attitudine o le preferenze di un utente possano essere determinate da un piccolo numero di fattori latenti, detti ***embeddings***. Questi ultimi sono dei vettori, solitamente a 5 dimensioni (la dimensione dipende dalla complessità), che rappresentano delle entità (es. un utente o un item). I numeri che compongono questo vettore descrivono le caratteristiche nascoste delle entità e sono detti fattori latenti (es. si può immaginare che il primo numero del vettore di un utente indichi quanto gli piacciano i film fantasy, mentre nei film questo numero sarà relativo a quanto esso si avvicini all'essere fantasy); se i numeri di due vettori sono simili tra loro, allora ci sono due possibilità: o due user sono simili o ad un user piacerà un item dell'altro vettore.

Un altro concetto molto importante è il fatto che esista una natura intrinseca dell'entità, che è indipendente dalle sue interazioni con gli altri, questa informazione è contenuta all'interno del ***bias***. In altre parole alcuni users sono più critici se messi a confronto con altri, così come alcuni items sono giudicati in maniera positiva dalla maggior parte degli utenti perché considerati universalmente eccellenti (es. i classici nei film o nei libri).

In generale, ciò che si ottiene dalla fattorizzazione delle matrici è quanto un utente sia allineato con un certo set di caratteristiche latenti e quanto esse siano adattabili ad un certo item [6].

Algoritmi Uno degli algoritmi più diffusi per implementare la fattorizzazione delle matrici è il Single Value Decomposition (SVD), il quale si è diffuso in particolar modo dopo aver dimostrato il suo potenziale con Netflix; altri algoritmi sono il Principal Component Analysis (PCA) e le sue variazioni, come il Non-Negative Matrix Factorization (NMF). Nei capitoli che seguono verrà spiegato in maniera più accurata unicamente il SVD.

2.4.3 Neural Networks

Possiamo pensare questa tecnica come un'estensione del metodo della fattorizzazione delle matrici, dove il prodotto puntuale user-item viene rimpiazzato con un'architettura neurale.

Nonostante l'efficacia della fattorizzazione delle matrici per il CF, le sue performance sono ostacolate dalla scelta di utilizzo del prodotto puntuale e delle funzioni di interazione user-item; le performance potrebbero migliorare includendo i termini bias user-item nella funzione di interazione, poiché il semplice prodotto puntale non riesce a catturare interamente la complessa struttura dei dati di interazione.
Per affrontare queste problematiche di performance, entra in gioco il Neural Collaborative Filtering (NFC) [7], che risolve i problemi riscontrati in due modi:

- **Modellando le interazioni user-item tramite un'architettura di reti neurali.** Esso utilizza un Multi-Layer Perceptron (MLP) per memorizzare le interazioni user-item; questo costituisce un upgrade rispetto al MF, poiché il MLP può memorizzare qualsiasi funzione continua e ha un alto livello di non linearità (grazie ai molteplici layer).

- **Generalizzando e esprimendo il MF come un caso particolare del NCF.**

Per quanto riguarda l'aspetto implementativo, nel caso di SVD o PCA, che prevedono la scomposizione della matrice originale (con pochi dati) nel prodotto tra due matrici ortogonali di rango basso, nelle reti neurali esse non devono essere ortogonali, poiché lo scopo è far si che il modello impari in autonomia il valore della matrice di embeddings.

Capitolo 3

Premesse e dettagli sull'implementazione

In questo capitolo verranno introdotti alcuni dettagli e concetti utili al fine della comprensione dell'implementazione suggerita nel capitolo successivo. In particolare verranno approfondite la definizione di *dataset*, le librerie utili per il Collaborative Filtering e alcune tecniche di verifica dell'accuratezza delle predizioni.

3.1 Dataset

Per sperimentare con gli algoritmi è necessario utilizzare dei Dataset, ovvero dati che contengono set di items o di user che hanno avuto reazioni per determinati item. La reazione può essere di due tipi:

- **implicita**, riflette indirettamente le preferenze dell'utente tramite azioni come la visualizzazione di items o l'aggiunta al carrello;
- **esplicita**, consiste nei ratings e nelle valutazioni ed è un feedback diretto.

Per lavorare con questo tipo di dati si fa spesso riferimento alle matrici formate da colonne di item (che contengono i ratings ricevuti dagli utenti) e righe di user (con i ratings per ogni item); è raro che tutte le celle della matrice siano piene perché

di solito gli user reagiscono solo ad alcuni items, non a tutti, e per questo ci sono due tipi di matrici:
sparse (molte celle vuote) e **dense** (quasi tutte le celle occupate).
Se si utilizza un approccio basato unicamente sul collaborative filtering la similarità non è calcolata sulla base di fattori come l'età, il tipo di item o altri dati, ma unicamente sulla base delle valutazioni, implicite o esplicite.

In generale, è fondamentale lavorare utilizzando un dataset di dimensioni adeguate al contesto in cui si sta operando, più dati conterrà il dataset stesso, più semplice sarà fornire suggerimenti agli utenti in maniera accurata ed efficace.

3.2 Accuratezza predizioni

Per misurare l'accuratezza delle predizioni e le performance esistono diverse metriche rilevanti, in questo documento verranno proposte le più utilizzate e rilevanti, in particolare per gli scenari proposti nella valutazione delle performance. Esse sono le seguenti:

- **Root Mean Square Error (RMSE)**

  Si predicono i ratings per un dataset di prova di cui si conoscono già i valori, a questo punto la differenza tra i valori effettivi e quelli predetti sarà l'errore; in altre parole, il risultato ottenuto rappresenta di quanto si discosta il rating stimato dall'algoritmo rispetto al rating effettivo inserito dall'utente.

  Questi risultati sono facilmente calcolabili tramite la formula che segue:

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(\frac{d_i - f_i}{\sigma_i}\right)^2}$$

- **Mean Absolute Error (MAE)**

  Questa metrica si occupa di calcolare il valore medio tra tutti gli errori assoluti, essendo questi ultimi definiti come il valore di errore commesso nella misurazione di una singola predizione.

  La formula necessaria ad ottenere questo risultato è la seguente:

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|e_t|$$

3.3 Librerie

Un altro aspetto essenziale da introdurre prima di passare all'implementazione sono le librerie utilizzate nell'ambito del Collaborative Filtering. Esse sono molteplici ma per la comprensione degli algoritmi dell'unità successiva ne saranno necessarie solo tre.

3.3.1 Surprise

La libreria Surprise contiene 3 classi, in particolare, molto utili per gli algoritmi che si vedranno:

- **Dataset**, usata per dividere in train e testers, sia direttamente che tramite un iteratore di controllo incrociato.
- **Trainset**, usato come parametro nel metodo `fit` di un modello.
- **TestSet**, usato come parametro nel metodo `test` di un modello.

Innanzitutto dividiamo i dati in testset e trainset. Il primo può contenere ratings user-item selezionati in maniera random oppure può essere riempito con tutti i dati e poi essere soggetto ad un controllo incrociato per il testing. Il secondo è il nostro caso e avviene con il metodo `build_full_trainset.`

Una volta creati gli oggetti Trainset e Testset, surprise si occupa di convertire i `raw_id`, ovvero gli identificatori degli item relativi agli user che sono nel file csv, in `inner_id`, ossia una serie di interi che parte da 0 che si usano per creare nuovi identificatori nell'algoritmo. Entrambi i tipi di identificatori potrebbero risultare utili, perciò la libreria Surprise fornisce metodi sia per la conversione da `raw_id` a `inner_id`, sia il viceversa, rispettivamente `to_inner_iid` e `to_raw_iid`.

Metriche di similarità

Le metriche di similarità che Surprise utilizza sono tre, delle quali le prime due sono state approfondite sufficientemente nel capitolo precedente:

Cosine Similarity, Pearson Similarity e Metric Standard Deviation.
Nonostante non ci siano scelte migliori di altre tra le metriche citate, l'ultima (MSD) non è molto utilizzata, per questo motivo ci si limiterà a citarne la formula, a titolo informativo.

$$msd_sim(i,j) = \frac{1}{msd(i,j)+1} \tag{3.1}$$

dove *msd(i,j)*:

$$msd(i,j) = \frac{1}{|U_{ij}|} \cdot \sum_{u \in U_{ij}} (r_{ui} - r_{uj})^2 \tag{3.2}$$

Modelli kNN

Il k-Nearest Neighbor, che verrà approfondito nel capitolo successivo, è un modello utilizzato in particolare nella famiglia Memory-Based come concetto di base per l'implementazione degli algoritmi. In Surprise esistono diverse varianti di questo modello, delle quali alcune risultano importanti da citare:

- **kNNBasic**, rappresenta il calcolo della media pesata dei ratings che gli user danno agli item simili, dove questi ultimi sono pesati per similarità.

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} sim(i,j)} \tag{3.3}$$

- **kNNWithMeans**, si differenzia dal modello precedente poiché effettua la media dei ratings degli items.

$$\hat{r}_{ui} = \mu_i + \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - \mu_j)}{\sum_{j \in N_u^k(i)} sim(i,j)} \tag{3.4}$$

- **kNNWithZScore**, si differenzia dal modello precedente poiché considera la deviazione standard dei ratings.

$$\hat{r}_{ui} = \mu_i + \sigma_i \cdot \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - \mu_j)/\sigma_j}{\sum_{j \in N_u^k(i)} sim(i,j)} \tag{3.5}$$

Testing

Le metriche di performance sono contenute all'interno del modulo di accuratezza di Surprise. Ci sono, in particolare, quattro metriche disponibili (RMSE, FCP, MAE, MSE), ma la più utilizzata è la Root Mean Squared Error, analizzata nel paragrafo precedente.

3.3.2 Pandas

Questa libreria è stata sviluppata per la manipolazione e l'analisi dei dati in Python; in particolare, rivolta alle strutture dati e alle operazioni per la gestione delle tabelle numeriche e le serie temporali.
La caratteristica più utile di questa libreria, che ritorna utile nell'analisi degli algoritmi che verrà presentata in seguito, è che Pandas si occupa di trasformare file di tipo CSV o TSV in oggetti Python con righe e colonne, chiamati DataFrame (df); per convertire oggetti Python, come dictionary o list, il comando da utilizzare è:

```
pd.DataFrame()
```

Solitamente essa viene utilizzata assieme alla libreria Numpy, che fornisce molti strumenti matematici utili.

3.3.3 Tensorflow.Keras

Keras costituisce la libreria di Python utilizzata per il Deep Learning e per le Reti Neurali. [1]
In particolare, i metodi più utili presenti in questa libreria, ai fini del lavoro esposto in questo documento sono i metodi dell'API `Model`, relativi all'allenamento dei modelli per i Model-Based.

Il metodo `compile` definisce la funzione di perdita, l'ottimizzatore e la metrica, al fine di poter allenare il modello su cui viene eseguito grazie all'utilizzo di un altro metodo, `fit`, necessario per adattare il modello ai parametri ricevuti in input. Essi verranno spiegati nel dettaglio nel capitolo successivo.

Capitolo 4

Algoritmi valutati

In questo capitolo verranno approfonditi gli algoritmi utilizzati per la valutazione delle performance del capitolo conclusivo, introducendo e spiegando alcuni elementi significativi dell'implementazione.

4.1 User-Based

Come spiegato in precedenza, l'approccio User-Based si incentra sulla similarità tra utenti per predire quali items saranno d'interesse per l'utente considerato; questa tecnica ci consente perciò di generare una lista di utenti simili a quello dato e, sulla base delle recensioni degli utenti in lista, sarà poi possibile predire i ratings per quest'ultimo.

Le tecniche utilizzate per implementare il modulo User-Based sono in particolare due: **k-Nearest Neighbor**, per costruire un insieme di utenti simili a quello dato sulla base delle loro recensioni, e la **Correlazione di Pearson**, per analizzare le coppie user/user e user/item al fine di applicare il kNN per costruire il neighborhood dell'utente considerato.

4.1.1 Pearson Similarity

Il primo step dell'implementazione consiste nel trovare i rapporti sia tra ogni utente e gli items che ha recensito, sia tra l'utente inizialmente analizzato e tutti gli altri utenti.

L'approccio che si è scelto di implementare all'interno della funzione `pearson_correlation` è quello di calcolare la media pesata per ognuno dei due utenti che si considerano in prima analisi, per poi verificare la presenza di items in comune tra i due.
A questo punto sarà possibile ottenere una stima numerica di quanto effettivamente i due utenti considerati siano simili, tramite un semplice ciclo, al cui interno viene sviluppata la formula della similarità di Pearson citata nel Capitolo 2.

```
for item in sxy:              #con sxy lista di item in comune
    rxs = user1_data[item]
    rys = user2_data[item]
    top_result += (rxs - rx_avg) * (rys - ry_avg)
    bottom_left_result += pow((rxs - rx_avg), 2)
    bottom_right_result += pow((rys - ry_avg), 2)
bottom_left_result = math.sqrt(bottom_left_result)
bottom_right_result = math.sqrt(bottom_right_result)
result = top_result / (bottom_left_result * bottom_right_result)
```

4.1.2 k-Nearest Neighbor

Una volta ottenuti i valori di similarità tra gli users del dataset considerato, è possibile generare una lista di users simili, sulla base delle similarità di Pearson calcolate precedentemente, ad un user fornito in input.

L'algoritmo k-Nearest Neighbor si basa sul concetto che gli items simili siano vicini tra loro, perciò si affida al calcolo della *distanza Euclidea* per determinare la distanza tra di essi. Gli step per l'implementazione di questo algoritmo sono, solitamente, i seguenti:

- importare i dati
- inizializzare k con il numero di vicini che si vogliono ottenere
- per ogni item considerato dal dataset si deve:
  - calcolare la distanza tra l'item fornito come input e quello attuale
  - aggiungere la distanza e l'indice dell'item attuale ad una collection ordinata
- ordinare la collection dove sono le distanza in ordine crescente
- selezionare i primi k items e prendere le loro labels:

- se si ottiene un problema di classificazione, ovvero l'output è costituito da valori discreti (interi), allora si ritorna il modulo delle k labels.
- se si ottiene un problema di regressione, ovvero l'output è costituito da valori reali (con la virgola), allora si ritorna la media delle k labels.

4.1.3 Predizioni

Una volta determinato il Neighborhood, tramite una semplice media pesata dei ratings attribuiti dai vicini ad un certo item, sarà possibile determinare il rating dell'utente considerato per l'item selezionato.

$$predizione = \frac{\sum W(i,1) \cdot rating(i, item)}{\sum W(i,1)} \tag{4.1}$$

dove W(i,1) è la similarità dell'utente i con l'utente 1 dei k nearest neighbors.

L'algoritmo descritto si occuperà, perciò, di generare in output la lista dei nearest neighbor e la predizione del rating dell'utente, sulla base dei soli vicini in lista che abbiano già recensito l'item su cui si sta effettuando la predizione.

4.2 Item-Based

A differenza dell'User-Based, questo approccio è finalizzato alla ricerca di similarità tra items per poter consigliare all'utente selezionato un item simile a quello che ha già dimostrato di apprezzare.

L'implementazione di questo algoritmo, al cui interno è stata utilizzata la già citata libreria *Surprise*, può essere suddivisa in 4 step fondamentali.

4.2.1 Importazione dei dati e costruzione del trainset

L'importazione dei dati in maniera corretta avviene grazie ad un oggetto `Reader`, costituito dai seguenti attributi:

- `line_format`: deve essere riempito con i nomi delle colonne del file di input e si assicura che l'ordine sia lo stesso presente in quest'ultimo;
- `sep`: costituisce il separatore settato tra gli elementi del file, ad es. una virgola se si lavora con un csv o un tab nel caso di tsv;
- `rating_scale`: una tupla con i minori e maggiori range possibili. (in caso di dati binari il range è 0 - 1)

Una volta inserito il dataset nell'oggetto `data`, si utilizza la funzione di *surprise* per la creazione del trainset.

```
trainsetfull = data.build_full_trainset()
```

4.2.2 Implementazione di kNNWithMeans e Cross-Validation

Utilizzando l'approccio kNN si hanno a disposizione due iperparametri da regolare: il parametro k e l'opzione di similarità.

Il **parametro k** ricopre un ruolo analogo al funzionamento che ha in generale nel kNN: è il limite superiore degli item simili che vogliamo l'algoritmo consideri (es. se ho ratings per 20 items e setto k a 10, con l'algoritmo prendo in considerazione solo i primi 10).

È possibile inoltre settare un min_k quando non si hanno a disposizione abbastanza dati; in questo caso si userà la media globale degli stessi per effettuare le stime (di default la media è 1).

Il **parametro opzione della similarità**, invece, permette di definire come calcolare la distanza tra gli items e viene implementato tramite un dictionary, chiamato `sim_option`, nel modo che segue.

```
sim_option = { 'name': 'pearson', 'user_based': False }
```

Le chiavi effettive di questo dictionary sono però in realtà 4:

- `Shrinkage`: utile unicamente nel caso di modelli kNNBaseline, perciò non in questo caso.
- `User_based`: ci sono due diversi percorsi quando si vogliono stimare le similarità; o si calcola quanto gli items sono simili tra loro o si fa lo stesso con gli user.
- `Min_support`: il minimo numero di punti comuni sotto il quale la similarità andrà settata a 0 (es. se il min_support è 10 e ci sono due items, per cui solo 9 user hanno attribuito ratings ad entrambi, allora la similarità dei due items sarà 0, indipendentemente dai ratings).
- `Name`: il tipo di formula per la similarità che si intende utilizzare.

Tutte le funzioni di similarità torneranno un numero tra 0 e 1 a una specifica coppia di item; se è 1 allora vuol dire che i ratings sono perfettamente allineati, se è 0 vuol dire che non c'è connessione tra i due items.

Infine, per quanto riguarda la **Cross-Validation**, viene utilizzato il metodo `cross_validate` definito nella libreria *Surprise*, il quale esegue una procedura di controllo incrociato per l'algoritmo dato, riportando misure di accuratezza e il tempo di computazione.
Il valore di ritorno è costituito da un dictionary con le seguenti chiavi:

- **algo**: contiene l'algoritmo da valutare;
- **data**: contiene il dataset su cui valutare l'algoritmo;
- **measures**: definisce il tipo di misure di performance da computare;
- **cv**: determina come il parametro `data` verrà diviso, cioè il tipo di *folds*;
- **return_train_measures**: indica se computare le misure di performance sul trainset, di default è FALSE;
- **n_jobs**: il numero massimo di *folds* valutabili in parallelo.

Nel nostro caso:

```
results = cross_validate(
algo=algo, data=data, measures=['RMSE'],
cv=5, return_train_measures=True)
```

4.2.3 Model-Fitting e Predizioni

Nello step riguardante il Model-Fitting ci si occupa di allenare l'algoritmo sul training set dato; il ruolo del metodo utilizzato, e citato sotto, è quello di inizializzare delle strutture interne e settare l'attributo `self.trainset`.

```
algo.fit(trainsetfull)
```

Per fare una predizione, invece, si può usare il raw id, che si ricorda essere l'indice effettivo dell'item nel dataset, all'interno del metodo `predict`, quest'ultimo ritornerà un dictionary di questo tipo:

```
Prediction(uid='TestUser1', iid='161936',
r_ui=None, est=6.647051644687803,
details={'actual_k': 4, 'was_impossible': False})
```

R_ui è None perché non si ha un rating effettivo per quell'item.

Per ottenere le migliori N raccomandazioni per un certo user gli step di base sono i seguenti:

- allenare un modello su trainsetfull
- creare un anti testset con il metodo `build_anti_testset` (rappresentante il complementare del dataset originario)
- eseguire le predizioni sull'anti testset con il metodo `test` (con questo step abbiamo una stima dei ratings di tutte le coppie user-item che risultavano mancanti)
- ordinare i ratings stimati per ogni user, con gli N items con ratings stimati più alti.

4.3 Matrix Factorization

L'approccio Matrix Factorization si differenzia completamente dai due precedenti, in quanto utilizza la tecnica Model-Based, per cui anche la valutazione effettuata si discosterà da quanto visto fin'ora per introdurre un confronto tra la Matrix Factorization Standard e quella con l'utilizzo delle Reti Neurali. L'analisi dell'algoritmo implementato è, come nei paragrafi precedenti, suddivisa in diverse sezioni.

4.3.1 Preparazione dei dati

La prima fase consiste nel ricavare i dati riguardanti gli item e gli user dal database utilizzato e farli convergere in un'unica matrice (Utility Matrix) contenente items, users e i ratings dati dagli users agli items.
Per generare questa matrice è necessario codificare le colonne e le righe, dopodiché possiamo proseguire nel generare la Utility Matrix, combinando le informazioni presenti nelle due matrici di users e items.
Vale la pena sottolineare che in questo modo si otterrà un matrice con NaN nelle posizioni corrispondenti ai ratings che non esistono, ovvero gli items che l'utente non ha ancora valutato.

L'ultimo step prima di procedere all'applicazione del MF è quello di costruire dei Trainset e Validation Set.

```
userid2idx = {o:i for i,o in enumerate(users)}
movieid2idx = {o:i for i,o in enumerate(movies)}
df['userId'] = df['userId'].apply(lambda x: userid2idx[x])
df['movieId'] = df['movieId'].apply(lambda x: movieid2idx[x])
split = np.random.rand(len(df)) < 0.8
train = df[split], valid = df[split]
```

4.3.2 Matrix Factorization standard

A questo punto si passa alla suddivisione della Utility Matrix in due sottomatrici di rango minore, gli Embeddings; ovvero si utilizza il metodo del **Low Rank Matrix Factorization** creando Embeddings sia per gli users che per gli items.

I numeri delle dimensioni costituiscono i **Fattori Latenti** degli Embeddings, ovvero gli iperparametri necessari per l'implementazione del Collaborative Filtering. L'approccio descritto ricorre all'utilizzo della libreria `tensorflow.keras` per generare gli Embeddings layers.

```
user_input= Input(shape=(1,),name='user_input',dtype='int64')
user_embedding=Embedding(n_users, n_latent_factors,
    name= 'user_embedding')(user_input)
```

Per definire l'item_input si agisce in maniera esattamente analoga alla definizione dello user_input, dopodiché si effettua il prodotto scalare di entrambi gli embeddings utilizzando il layer `merge` di Keras; questo prodotto costituisce un approccio per il calcolo della similarità che può essere eventualmente sostituito con altre modalità, quali la similarità del coseno.

```
sim= dot([user_vec,item_vec],name='Simalarity-Dot-Product',
 axes=1)
```

A questo punto generiamo un modello Keras con i dettagli specificati, tramite il metodo `models.Model`, e lo compiliamo con la metrica di controllo dell'errore RMSE, utilizzando il metodo `compile`, allo scopo di minimizzare l'errore sul Training set. Prima di proseguire con l'analisi dell'ultimo metodo definiamo il SGD [2], **Stochastic Gradient Descent**, come un metodo iterativo per l'ottimizzazione di funzioni differenziabili, dove, ad ogni iterazione, si sostituisce il valore esatto del gradiente della funzione costo con una stima ottenuta valutando il gradiente solo su un sottoinsieme degli addendi.
L'ultimo step consiste nell'allenamento del Trainset:

```
History= model.fit([train.userId,train.movieId],train.rating,
 batch_size= batch_size,epochs= 50, validation_data=
 ([valid.userId,valid.movieId],valid.rating), verbose = 1)
```

Il Trainset viene allenato tramite il metodo `fit`, necessario per adattare i parametri del modello al fine di fornire dati. Quando questa funzione viene utilizzata i parametri del modello vengono inizializzati con valori random e poi aggiornati iterativamente con SGD per diminuire l'errore, questi update avvengono finché non viene eseguito il numero di `epoch` passato come parametro. Le `epochs` vengono completate ogni volta che l'SGD ha processato un numero sufficiente di piccole combinazioni di dati tali per cui il numero totale di informazioni analizzate risulti equivalente alla dimensione dell'intero training dataset.

4.3.3 Matrix Factorization Neural Network

Per quanto riguarda invece l'applicazione delle tecniche per le Reti Neurali, si parte comunque dalle premesse precedenti e si agisce in maniera molto simile, variando il solo utilizzo di metodi specifici per le NN.

La definizione degli Embeddings resta la stessa del paragrafo precedente, con l'aggiunta del metodo `Dropout`, che rappresenta una tecnica di regolarizzazione per ridurre il sovradimensionamento nelle reti neurali artificiali prevenendo adattamenti sui dati di training, inoltre rappresenta un metodo efficiente per eseguire la media dei modelli con le reti neurali. Il prodotto scalare viene applicato come definito in precedenza.

La differenza sostanziale consiste nella definizione dell'architettura del modello, ovvero i layers.
Il layer più importante è il `Dense`, presente in ogni rete neurale, che costituisce un layer intrinsecamente connesso con il layer che lo precede. Questo vuol dire che il Dense layer riceve l'output da tutti i neuroni del livello che lo precede, per poi far effettuare ai suoi neuroni Dense una moltiplicazione matrice-vettore.

```
nn_inp= Dense(96,activation='relu')(sim)
```

```
nn_inp= Dropout(0.4)(nn_inp)
nn_inp= Dense(1,activation='relu')(nn_inp)
nn_model= keras.models.Model([user_input, movie_input],nn_inp)
```

`Dense` implementa l'operazione *output = activation(dot(input, kernel) + bias)*, nella quale l'elemento activation costituisce una funzione utilizzata per far si che la NN possa imparare la relazione tra inputs e outputs; kernel costituisce la matrice di pesi creata dal layer e il bias è il vettore di bias creato dal layer. I parametri che le vengono passati nel metodo sono activation e units; il primo è settato a 'relu', ossia utilizza la funzione di attivazione dell'unità lineare rettificata (rectified linear unit), mentre il secondo costituisce la dimensione dell'output del Dense layer.

Infine, come nel paragrafo precedente, il modello viene compilato (`compile`) ed allenato (`fit`), ottenendo però, questa volta, risultati migliori, che verranno introdotti e spiegati nel capitolo successivo sul calcolo delle prestazioni.

Capitolo 5

Performance e Confronti

In questo capitolo verranno utilizzati gli algoritmi descritti nel capitolo precedente per effettuare una misurazione delle performance per ognuno.
In particolare il confronto sarà separato tra gli algoritmi Memory-Based e quelli Model-based per poter evidenziare al meglio le prestazioni di ognuno, considerando che gli algoritmi appartenenti a famiglie differenti presentano una differenza sostanziale a livello di prestazioni; queste ultime verranno comunque approfondite nella parte finale del capitolo per evidenziarne disuguaglianze.

Ogni algoritmo è stato testato su un laptop con processore Intel Core i3-4010U, frequenza di clock massima 1.70GHz e 8GB di RAM.

5.1 User-based/Item-Based

Il confronto iniziale descritto è tra Item-Based e User-Based; essi verranno in primo luogo analizzati nello specifico ed in seguito confrontati. Il Dataset utilizzato per entrambi è lo stesso, costituito da circa 1500 ratings (compresi tra 0.0 e 5.0) effettuati da circa 40 utenti su circa 100 items (film).

5.1.1 Performance User-Based

Per la valutazione delle performance è possibile approcciarsi all'algoritmo descritto analizzando diversi parametri. Innanzitutto, l'input ne prevede 3: il nome di un film, il nome di un utente e k (il numero di vicini più simili); il risultato in output mostrerà la lista dei k-NN con i rispettivi valori di similarità e il valore predetto per il rating dell'utente sul film inserito.

Ciò che vale la pena sottolineare è che al crescere di k l'affidabilità della predizione potrebbe migliorare, in quanto si avrebbero a disposizione più utenti per il confronto; potrebbe anche accadere però, che gli utenti meno simili a quello preso in input contribuiscano, nella media pesata, ad allontanare la predizione dal rating effettivo. Nel caso in cui si decida di inserire un film per cui l'utente ha già espresso un rating, sarà possibile verificare al meglio l'accuratezza della predizione; nel caso in analisi l'errore è circa ± 0.35.

5.1.2 Performance Item-Based

Come nel caso precedente, la valutazione dipende sempre dal variare degli stessi fattori, considerati gli stessi input; in generale, tutte le affermazioni di premessa relative al paragrafo precedente possono essere ripetute. La differenza risiede nell'output, poiché questa volta la lista generata è costituita dagli items (film) che il dato utente in input potrebbe apprezzare. La predizione generata per lo stesso utente e con lo stesso film del caso precedente risulta migliore, con un errore, rispetto al rating effettivamente attribuito, di circa ± 0.20.

5.1.3 Confronto

Il confronto tra queste due implementazioni del Memory-Based viene effettuato sia sulla base delle tempistiche, sia sulla predizione effettivamente generata; in entrambi i casi le differenze sono piccole.

Il rating generato è stato descritto nei paragrafi precedenti; per quanto riguarda le tempistiche invece, esse variano, anche se in maniera impercettibile, sulla base

del valore di k. Nella tabella che segue si mettono in evidenza le differenze a livello di tempi tra i Memory-Based, al variare del parametro k (bisogna ricordare che gli output differiscono nei due casi, ciò influenza i tempi).

k	time user-based	time item-based
1	0.023s	0.607s
10	0.037s	0.914s
20	0.106s	1.113s
30	0.284s	1.151s

Nel grafico che segue invece, vengono evidenziate le variazioni delle predizioni in casi differenti; sono stati presi in esame diversi item e diversi utenti per valutarne i risultati.

Ciò che risulta chiaro è che l'andamento generale tende a migliorare la predizione nel caso ci sia una disponibilità maggiore di utenti simili all'item che hanno recensito a loro volta; tuttavia in alcuni casi, il numero di vicini compromette la valutazione andando ad intaccare il peso del più simile che, se fosse stato l'unico riferimento, avrebbe probabilmente portato un rating più vicino a quello effettivo. Sull'asse delle ordinate è riportata la differenza tra la valutazione effettiva dell'utente e il valore predetto dai diversi algoritmi (RMSE).

5.2 Matrix Factorization Standard/NN

Questo secondo confronto, sui due algoritmi per il Matrix Factorization, utilizza un dataset di riferimento di dimensioni significativamente maggiori di quello dei casi precedenti; questo per analizzare le prestazioni migliori che gli algoritmi Model-Based possono fornire, in quanto essi tendono a performare meglio con maggiori quantità di dati.

Il Dataset è costituito da circa 100k ratings (compresi tra 0.0 e 5.0), effettuati da circa 1000 utenti su 1700 items (film); il Training Test avviene su 79978 valori, mentre il Validation Test su 20026.

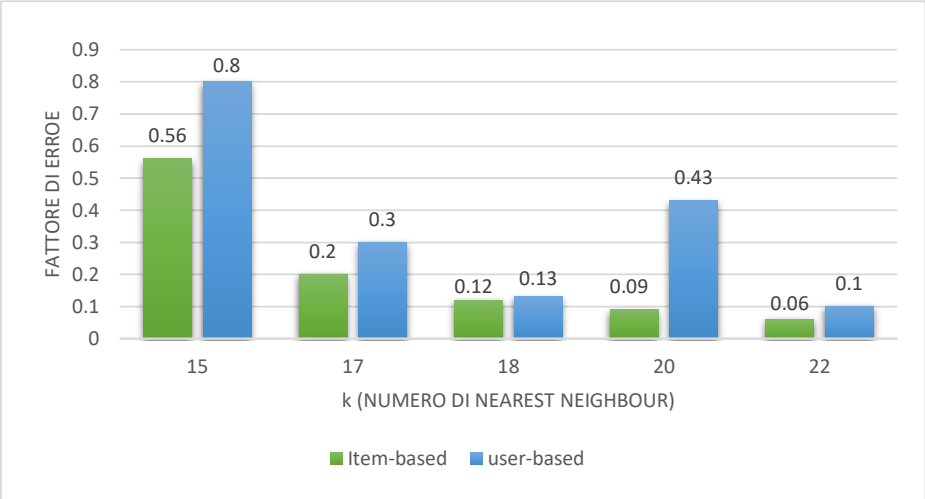

Figura 5.1. Andamento del tasso di errore per le predizioni, al variare di k

5.2.1 Performance Matrix Factorization Standard

In questo primo caso l'esecuzione avviene ripetendo iterativamente il metodo del SGD (come specificato nel capitolo precedente, grazie alla funzione `fit`), è stato necessario settare il valore di epoch, quindi delle iterazioni, a 50 per ottenere risultati accettabili, mentre il bacth_size, quindi la dimensione dei gruppi di dati su cui effettuare i confronti, è stato posto a 128.

Alla fine della 50esima iterazione il validation loss, ossia il valore di perdita risultato della funzione di perdita del RMSE, è sceso fino ad arrivare ad un errore di 1.2452, ottimizzato significativamente rispetto al valore di loss ottenuto nella prima epoch pari a 13.4316.

5.2.2 Performance Matrix Factorization NN

Nel caso delle Reti Neurali l'ottimizzazione diventa ancora più significativa e permette di effettuare poi delle predizioni molto più affidabili rispetto al semplice

MF.

I parametri per il metodo `fit` vengono mantenuti allo stesso modo dell'algoritmo precedente, fatta eccezione per il parametro epoch che sarà sufficiente settare a 20, per questa volta, in quanto sarà possibile ottenere da subito un risultato soddisfacente. Il validation loss di partenza, con la prima epoch, è di 1.1228, mentre il risultato migliore che si riesce ad ottenere con la 20esima iterazione è di 0.8363.

5.2.3 Confronto

Il confronto, in questo caso, avviene sulla base delle iterazioni e dei valori ottenuti a livello di validation loss nei due casi.

Il grafico che segue evidenzia i suddetti parametri nel caso di MF Standard e di MF Neural Network, la differenza rende evidente l'algoritmo migliore nel caso di grandi quantità di dati, nonostante ciò entrambi restano valide opzioni. Importante da ricordare è che il dataset utilizzato è di dimensioni molto trascurabili rispetto ai dataset in uso in contesti reali, in questi ultimi infatti, gli algoritmi rendono in maniera ancora maggiore considerata l'immensa quantità di dati a disposizione.

5.3 Osservazioni conclusive

I confronti sono stati effettuati solamente sulla base della stessa famiglia di algoritmi ed utilizzando dataset diversi, questo per mostrare al meglio le capacità di tutte le tecniche descritte.

Risulta evidente che porre gli algoritmi Memory-Based ad utilizzare dataset delle dimensioni di quello utilizzato per il Model-Based o addirittura di dimensioni maggiori, come avviene nella realtà, porterebbe ad un rallentamento a livello di tempistiche molto evidente.

Allo stesso modo, fornire un dataset di piccole dimensioni ad un algoritmo Model-Based non permetterebbe di ottenere risultati tanto affidabili quanto avverrebbe con un dataset come quello proposto o di dimensioni ancora maggiori.

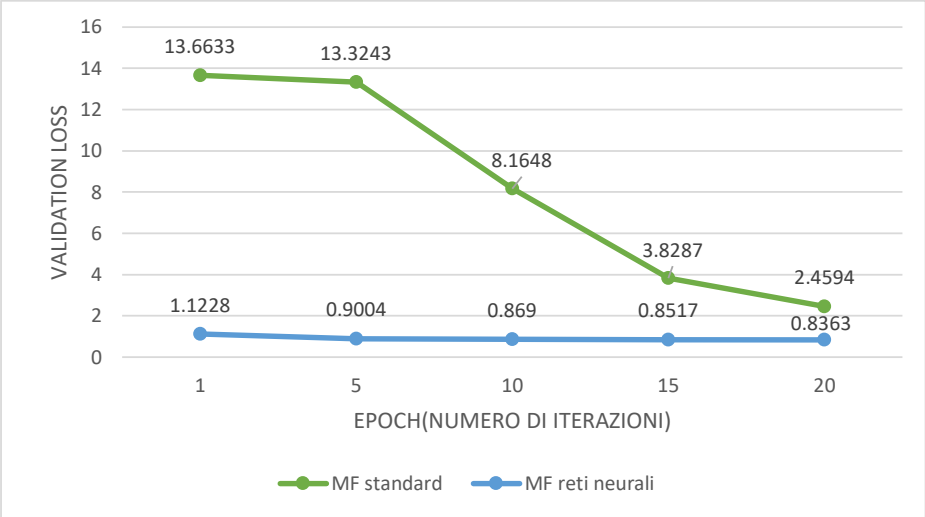

Figura 5.2. Andamento del validation loss, al variare delle iterazioni

In generale, nel contesto attuale, l'uso combinato delle due famiglie risulta la soluzione più affidabile, efficiente ed efficace; nella fase di costruzione del dataset, con pochi ratings user/item, l'alternativa migliore è quella Memory-Based, mentre quando si è riusciti ad ottenere un dataset di dimensioni sufficientemente grandi bisogna ricorrere ad algoritmi Model-Based, per migliorare prestazioni e predizioni.

Capitolo 6

Conclusioni

Gli algoritmi descritti rappresentano certamente una versione significativamente semplificata rispetto a ciò che viene utilizzato nel contesto reale attuale, tuttavia risultano sufficienti per la comprensione della loro importanza; infatti, se si pensa ai risultati accettabili ottenuti con i database di piccole dimensioni citati e algoritmi con prestazioni non ottimali, risulta evidente quanto si possa ottenere da un algoritmo più efficace.

Il concetto del Collaborative Filtering, al giorno d'oggi, è diffuso in moltissime piattaforme di diverso genere; l'esempio migliore presente attualmente riguarda l'algoritmo di raccomandazione utilizzato dal social network TikTok, al fine di suggerire video all'utente che potrebbero interessarlo. Attualmente esso rappresenta l'algoritmo più efficace nell'accuratezza delle predizioni, utilizzando sistemi di Machine Learning, in quanto suggerisce video sulla base di 3 fattori:

- **user interactions**, cioè i video a cui si mette like, si condividono, si guardano fino alla fine o si skippano;
- **video information**, ossia gli argomenti e i personaggi descritti nei video del punto precedente;
- **device and account settings**, quindi le informazioni sul device utilizzato e sulle impostazioni dell'account (lingua, Stato, etc.).

I tipi di algoritmi utilizzati in questo caso riguardano sia il Collaborative Filtering che il Content-Based Filtering, sfruttando le potenzialità di entrambe le famiglie; questo porta l'utente a ricevere sulla sua FYP (For You Page), ossia la schermata su cui appaiono i video consigliati, solo video inerenti ai suoi interessi e alla sua mentalità, testando tuttavia, di tanto in tanto, l'interesse dell'utente a nuovi video al di fuori delle preferenze espresse, al fine di scoprirne di nuove.

Altri esempi molto importanti risultano essere le piattaforme Netflix e Spotify e i social network simili a quello descritto, ossia Instagram e Youtube; in tutti questi casi si ricorre all'utilizzo degli algoritmi descritti. Risulta necessario sottolineare però, che la quantità di dati a disposizione, molto spesso, genera discussioni sulla questione della riservatezza degli stessi, poiché molte di queste piattaforme collezionano dati di ogni genere sugli utenti, permettendo predizioni più accurate a discapito della privacy.

Questa tesi ha avuto lo scopo di descrivere in maniera generale il funzionamento di questi algoritmi così diffusi e di provarne diversi tipi di implementazione, al fine di poterne comprendere e descrivere le funzionalità che forniscono e le motivazioni della loro grande propagazione.

Bibliografia

[1] Alla, S. *Introduction to Keras* (2020). Available from: https://towardsdatascience.com/introduction-to-keras-part-one-data-loading-43b9c015e27c.

[2] Bottou, L. *Stochastic Learning in Advanced Lectures on Machine Learning* (2004).

[3] Grover, P. *Various Implementations of Collaborative Filtering* (2017). Available from: https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0.

[4] Jain, G., Mahara, T., and Tripathi, K. N. *A Survey of Similarity Measures for Collaborative Filtering-Based Recommender System* (2020). Available from: https://www.researchgate.net/publication/339466606_A_Survey_of_Similarity_Measures_for_Collaborative_Filtering-Based_Recommender_System.

[5] Kadie, C., Breese, J., and Heckerman, D. *Empirical analysis of predictive algorithms for collaborative filtering* (1998).

[6] Luo, S. *Introduction to Recommender System* (2018). Available from: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26.

[7] Sharma, A. *Neural Collaborative Filtering* (2019). Available from: https://towardsdatascience.com/neural-collaborative-filtering-96cef1009401.

[8] Terveen, L. and Hill, W. *Beyond Recommender Systems: Helping People Help Each Other* (2012).