UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria dell'Informazione

Un modello di deep learning per la raccomandazione di film: progettazione e implementazione per dispositivi Android mediante TensorFlow Lite

RELATORE:
Carlo Fantozzi

LAUREANDO:
Giacomo Visentin

ANNO ACCADEMICO 2022/2023

DATA DI LAUREA 27/09/2023

Abstract

Molte applicazioni moderne, sia per computer che per dispositivi mobili, utilizzano sistemi di raccomandazione per fornire suggerimenti personalizzati agli utenti migliorando la loro esperienza all'interno dell'applicazione stessa. Questi sistemi, che si basano su algoritmi di Machine Learning, sono ampiamente impiegati in contesti che vanno dall'intrattenimento ai social media, fino alle piattaforme di vendita online.

Lo scopo di questa tesi è stato lo sviluppo di un semplice sistema di raccomandazione basato sul collaborative filtering e realizzato in Python con le librerie Keras e TensorFlow. Il modello ottenuto è in grado di fornire all'utente una lista di dieci raccomandazioni di film, basandosi sulle valutazioni che l'utente ha espresso per altri film e confrontandole con le recensioni di altre persone. Il modello è stato successivamente convertito con TensorFlow Lite, rendendolo quindi adatto all'implementazione su dispositivi Android e iOS che, presentando molte più limitazioni rispetto ai computer, necessitano di una versione adatta ad essi.

È stata successivamente sviluppata un'app utilizzando il framework Flutter e il linguaggio di programmazione Dart per implementare il modello creato e consentire agli utenti di visualizzare i suggerimenti dei film direttamente dai dispositivi mobili.

Indice

Capitolo 1

Cenni teorici

1.1 Machine Learning

1.1.1 Definizione

"Il Machine Learning è il campo di studio che dà ai computer la capacità di imparare senza essere programmati esplicitamente". Questa è la definizione di Machine Learning secondo Arthur Samuel, informatico statunitense pioniere dell'intelligenza artificiale che nel luglio 1959 coniò il termine Machine learning usandolo per la prima volta in una sua pubblicazione.[21]

Una definizione più moderna di è quella data da Aurélien Geron, informatico e autore di alcuni dei libri più famosi e venduti sull'Intelligenza artificiale: "Il Machine Learning è la scienza (e arte) di programmare i computer in modo che possano imparare dai dati".

Il Machine Learning è quindi la pratica di aiutare il software ad eseguire un'attività senza regole esplicite e senza essere programmata. Nella programmazione tradizionale, uno sviluppatore deve specificare con precisione tutto l'algoritmo e quindi la sequenza di passi necessaria per trasformare un input nell'output desiderato. Il machine Learning ci permette di raggiungere lo stesso obiettivo ma i programmatori in questo caso non specificano l'algoritmo: forniscono invece una serie di esempi che il computer utilizza successivamente per apprendere.[8]

1.1.2 L'utilità del Machine Learning

La programmazione tradizionale non è scomparsa con la sempre più diffusa espansione dell'intelligenza artificiale; questo perché il Machine Learning non è la soluzione ad ogni tipo di problema. Sorge spontaneo chiedersi allora quando utilizzare il Machine Learning. Supponiamo di trovarci di uno scenario in cui non è possibile codificare le regole con precisione, come ad esempio se si volesse scrivere un programma in grado di riconoscere se una mail è spam oppure no. Per cercare di scrivere un codice in grado di risolvere tale problema dovremmo fare una lista di regole complesse, magari ricercando la presenza di alcune parole chiave come "carta di credito", "gratis" e "prova" o pattern nella struttura delle mail. Basterebbe però cambiare questi pattern o le parole utilizzate per sfuggire al controllo dell'algoritmo tradizionale. Un modello di Machine Learning sarebbe invece non solo più capace di scovare pattern nascosti e quindi migliore nel riconoscimento, ma anche in grado di adattarsi ai cambiamenti delle mail risultando quindi la scelta migliore per risolvere questa tipologia di problemi. In generale il Machine Learning è utile per:

- risolvere problemi che necessitano lunghe sequenze di regole o che non hanno una soluzione valida usando tecniche tradizionali;
- risolvere problemi in cui le condizioni o le variabili possono cambiare in modo imprevedibile e la cui soluzione deve essere in grado di adattarsi a tali cambiamenti;
- ottenere intuizioni su problemi complessi e grandi quantità di dati data la sua capacità di scovare pattern nascosti e difficilmente intuibili da un essere umano[8, 26].

1.2 Deep Learning

Il Deep Learning è una sottoarea del Machine Learning, la quale si fonda sull'utilizzo di algoritmi basati su reti neurali artificiali progettate per emulare il funzionamento dell'apparato cerebrale umano.[2]

É stato proprio il cervello umano che ispirò la creazione delle Reti Neurali Artificiali, conosciute anche con il termine inglese "Artificial Neural Network" (ANN), anche se

la similitudine tra il nostro cervello e una rete neurale è andata con tempo sempre più perduta.[9] Le ANN non sono affatto una novità: sono state inventate nel 1943 dal neurofisiologo Warren McCulloch e dal matematico Walter Pitts, mostrando un loro possibile utilizzo assieme a dei neuroni biologici di animali per eseguire calcoli complessi. Fu la prima volta in cui venne presentata un architettura di rete neurale. Anche se il successo iniziale delle reti neurali artificiali portò una diffusa convinzione di poter presto interagire con macchine intelligenti, rapidamente l'ottimismo svanì. I finanziamenti cominciarono a essere indirizzati verso altre direzioni e le reti neurali rimasero in sospeso per decenni. Nel frattempo altre tecniche di Machine Learning vennero create e sembravano offrire risultati migliori. Ora però c'è un rinato interesse per le ANN e ci sono diversi motivi per credere che le reti neurali non verranno facilmente accantonate nuovamente:

- C'è un'enorme quantità di dati per poter allenare i modelli e spesso se i problemi sono molto grandi e complessi le reti neurali superano altre tecniche di ML;

- La potenza di calcolo è cresciuta in modo esponenziale negli ultimi decenni[1] e permette di allenare modelli molto più velocemente;

- Sono coinvolte in un circolo virtuoso: straordinari prodotti derivati da esse trovano spazio nei media, catturando molta attenzione e generando di conseguenza consistenti finanziamenti. Questi finanziamenti a loro volta alimentano il progresso di tali reti, portando a nuovi miglioramenti. Alcuni esempi di tali prodotti sono le automobili a guida autonoma, il riconoscimento facciale o assistenti vocali come Siri o Alexa.[8]

[1]Nel 1965 Gordon Moore fece una previsione ora conosciuta come legge di Moore. Affermò che il numero di transistor all'interno di un circuito integrato (microchip) sarebbe raddoppiato approssimativamente ogni diciotto mesi, con un conseguente aumento della potenza di calcolo. La previsione è stata generalmente rispettata per diverse decadi e la potenza di calcolo attualmente disponibile è quindi cresciuta esponenzialmente[14]

1.2.1 Perceptron

Con il termine percettrone, perceptron in inglese, ci si riferisce sia ad una rete neurale composta da singolo neurone artificiale sia ad una rete composta da un singolo livello di neuroni. D'ora in poi chiamerò percettrone il singolo neurone artificiale. Il percettrone, inventato da Frank Rosenblatt nel 1957, è uno dei più semplici modelli di rete neurale nonché un algoritmo di classificazione binario utile quindi come separatore di due classi di dati. Prende come input una una serie di valori numerici $\mathbf{x}$, a cui sono associati dei pesi $\mathbf{w}$, e ne calcola la combinazione lineare z:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{x}^T \mathbf{w}.$$

L'output y è quindi ottenuto calcolando la funzione gradino di z:

$$y = h_{\mathbf{w}}(\mathbf{x}) = heaviside(z).$$

La funzione gradino conosciuta in inglese anche come funzione di Heaviside è:

$$heaviside(z) = \begin{cases} 0 & se \quad z \leq 0, \\ 1 & se \quad z > 0. \end{cases}$$

L'output di un percettore è un valore binario.

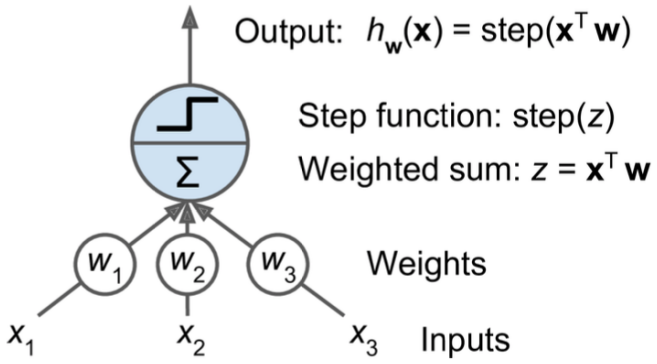

Figura 1.1: Percettrone.

L'esempio introdotto precedentemente (riconoscere se una mail è spam o non spam)

può essere risolto usando un percettrone: durante l'addestramento il percettrone impara i pesi ottimi per le caratteristiche delle mail con lo scopo di minimizzare l'errore tra il suo output che rappresenta la sua previsione e l'etichetta corretta fornita in fase d'addestramento. Con i pesi ottenuti in questa fase può quindi classificare nuove mail. É bene notare come in realtà il percettrone abbia una struttura troppo semplice per un problema di questo tipo; infatti le email spam cercano spesso di eludere i filtri anti-spam usando tecniche avanzate quindi per questa tipologia di problema sono più utili soluzioni più sofisticate.

Reti neurali più complesse sono formate connettendo tra loro molti livelli di percettroni. Il primo livello è chiamato livello di input, l'ultimo è il livello di output mentre i livelli intermedi sono chiamati *hidden layers* (letteralmente livelli nascosti). Ogni neurone riceve degli input dai neuroni del livello precedente, li elabora e produce un output che può essere trasmesso come input ai neuroni del livello successivo.

Le reti neurali possono variare in complessità e struttura in base al problema che devono risolvere. Variano da semplici percettroni multi-strato a architetture più avanzate come le reti neurali convoluzionali (CNN) per l'elaborazione delle immagini oppure le reti neurali ricorrenti (RNN) utili per il trattamento del linguaggio naturale.[8]

Una delle possibili applicazioni delle reti neurali è la creazione di sistemi di raccomandazione.

1.3 Sistemi di Raccomandazione

Nella quotidianità, ci affidiamo spesso ad applicazioni d'intrattenimento che possono includere social network come Twitter o Instagram, così come piattaforme di streaming come Netflix o YouTube. Chiunque abbia utilizzato almeno una volta una di queste applicazioni avrà notato come vengano suggeriti video o film da guardare in modo personalizzato, oppure post con contenuti simili a quelli che ci sono piaciuti in precedenza. Tutti questi suggerimenti sono il risultato di complessi sistemi di raccomandazione, il cui impiego si estende ampiamente oltre il campo dell'intrattenimento come nel caso di piattaforme

di vendita digitali come Amazon o Zalando che utilizzano questi modelli per consigliare prodotti da vendere.

Il modo più semplice e immediato per creare un tale sistema consiste nel raccomandare gli elementi più popolari cioè più acquistati o più consultati, ad esempio su YouTube è presente la sezione delle "Tendenze", che si basa proprio su questa concetto elementare, mentre Netflix consiglia i film più visualizzati in raccolte come "I titoli del momento" oppure "Top 10 delle serie in Italia oggi".

Questo tipo di raccomandazione è utile soprattutto per utenti nuovi che non hanno interagito con nessun elemento[2], ma aumentando la complessità del modello si possono creare delle raccomandazioni personalizzate per ogni utente ottenendo dei risultati più precisi ed efficaci.

Possiamo dividere i sistemi di raccomandazione personalizzati in due grandi categorie: sistemi **content based** e sistemi bastati sul **collaborative filtering**.

1.3.1 Content Based

I sistemi content based si basano sulla similarità degli attributi per fornire le raccomandazioni. Se si volesse consigliare un film attraverso questa tipologia di sistema si potrebbe consigliarne uno con lo stesso attore protagonista, diretto dallo stesso regista o più semplicemente appartenente allo stesso genere. Pertanto, basandosi sulle valutazioni di un utente, il modello propone elementi simili a quelli che l'utente ha gradito in passato, creando così raccomandazioni personalizzate e focalizzate su di lui.

Gli svantaggi di questa tipologia di modello non sono trascurabili:

- Necessitano che gli elementi da raccomandare presentino delle informazioni aggiuntive rispetto al solo valore della valutazione. Un dataset che presenta solamente valutazioni non è utile per questa tipologia;

[2]Con il termine "elemento" mi riferisco ai contenuti o agli oggetti che vengono raccomandati agli utenti come film, libri o prodotti

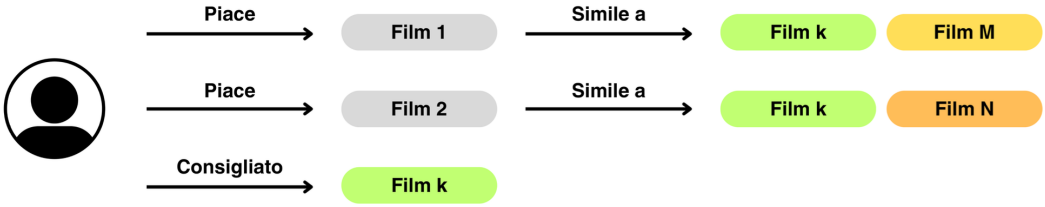

Figura 1.2: Esempio di sistema Content Based

- Molte informazioni devono essere raccolte per creare un modello in grado di eseguire delle ottime raccomandazioni;

- C'è il rischio che si verifichi il fenomeno della "filter bubble". Se un utente legge un libro che aderisce a una specifica ideologia politica, saranno consigliati libri simili, con il rischio che l'utente finisca per leggere esclusivamente opere di quella stessa ideologia. Questa situazione è solo un esempio, ma casi con questo tipo di problematica si riscontrano nei social media, dove individui con opinioni simili interagendo con post della medesima ideologia conducono gli algoritmi dei social a proporre loro esclusivamente contenuti della stessa natura isolandoli in tal modo nella loro bolla ideologica. Questo fenomeno può portare a una polarizzazione delle opinioni, incentivando anche la diffusione di teorie del complotto o di idee estremizzate.[7]

1.3.2 Collaborative Filtering

Se nei sistemi content based le raccomandazioni vengono effettuate basandosi su gli interessi di un singolo utente, i sistemi basati sul collaborative filtering sfruttano invece i gusti di molti utenti. L'idea di base è che se un utente A ha la stessa opinione di B su un determinato argomento, allora è più probabile che A abbia la stessa opinione di B su un altro argomento rispetto ad una persona casuale.

Il collaborative filtering si divide a sua volta in due categorie: il **memory based** e il **model based**.

Memory Based

Questi modelli utilizzano le informazioni presenti nei dataset per cercare similitudini tra i gusti di utenti. Non sono modelli particolarmente complessi da implementare e i loro risultati sono intuitivi. Hanno come aspetto negativo la necessità di utilizzare tutto il dataset per eseguire le prediction, il che può rallentare la raccomandazione.

Si dividono in:

- User based: Questi modelli raccomandano elementi che piacciono ad utenti simili, rispondendo alla frase "A utenti simili a te piace anche...". Ad esempio se all'utente A piacciono i film X e Y mentre a B piacciono X, Y e Z allora ad A può essere raccomandato Z se i due utenti sono ritenuti simili. Quindi partono da un utente per consigliare altri elementi piaciuti a utenti simili;

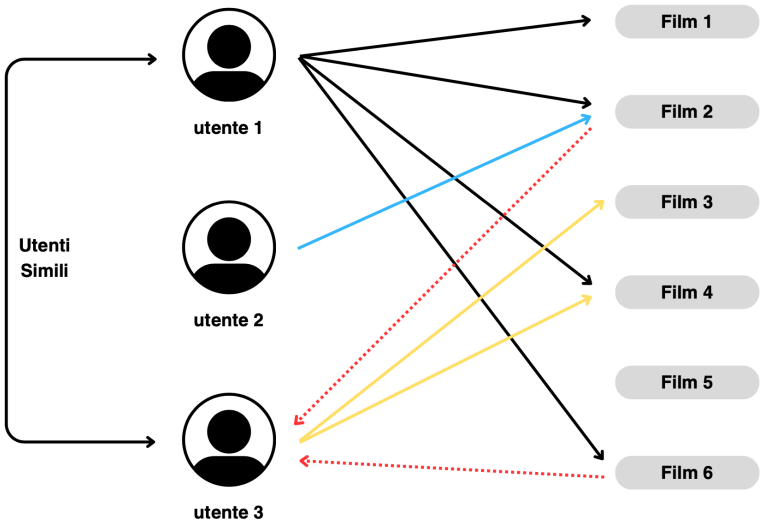

Figura 1.3: Esempio di sistema User Based. Le frecce rosse tratteggiate indicano i film raccomandati

- Item based: Questi modelli rispondono alla frase "A utenti a cui è piaciuto questo elemento piace anche...". Quindi partono da un elemento che è particolarmente piaciuto ad un utente, e gli consigliano degli elementi che sono piaciuti ad altri utenti a cui è piaciuto lo stesso elemento;

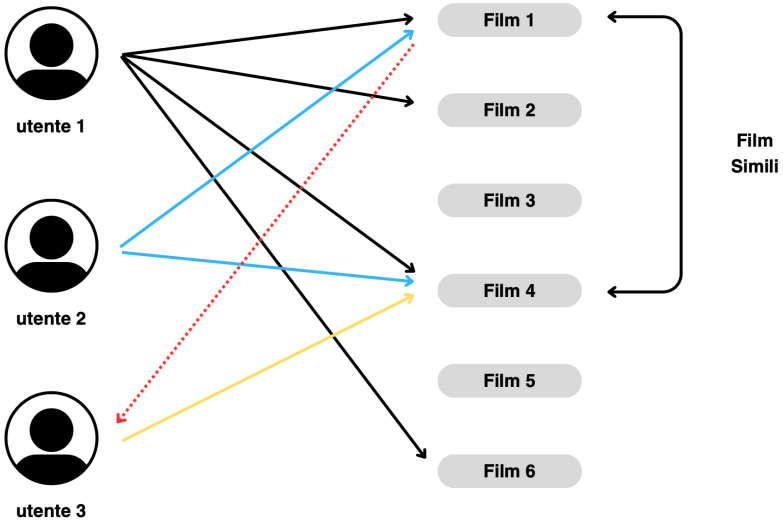

Figura 1.4: Esempio di sistema Item Based. La freccia rossa tratteggiata indica il film raccomandato

Model Based

Questa tipologia di sistema utilizza un approccio diverso: basandosi sulle interazioni tra utenti ed elementi cerca di predire la valutazione di elementi non recensiti. Per ottenere ciò crea un modello matematico che rappresenta le relazioni tra gli utenti e gli elementi del sistema. Le relazioni che il sistema è in grado di apprendere sono spesso nascoste: ad esempio in un modello di raccomandazione di film possono emergere associazioni tra generi (es: utenti a cui piacciono film thriller tendono ad apprezzare film di fantascienza o d'azione), film cult che piacciono ad un determinato gruppo di persone, preferenze per determinati attori o registi o persino differenze regionali. Tuttavia il modello necessita che gli elementi possiedano molte informazioni per essere davvero efficace.

Svantaggi Collaborative filtering

Alcuni svantaggi del collaborative filtering sono:

- È difficile avere un dataset che contenga molta varietà di utenti e opinioni originali e diversificate. Spesso una particolare opinione domina la community rendendo difficile eseguire predizioni precise;

- Il cold start problem: un utente che ha recensito pochi o addirittura nessun elemento, non permette al modello di eseguire raccomandazione specifiche su di esso;

1.3.3 Sistemi Ibridi

Moderni sistemi di raccomandazione non usano solo uno dei metodi descritti precedentemente ma creano modelli ibridi che implementano contemporaneamente più tipologie di raccomandazione per ottenere una risultati migliori. [20, 12]

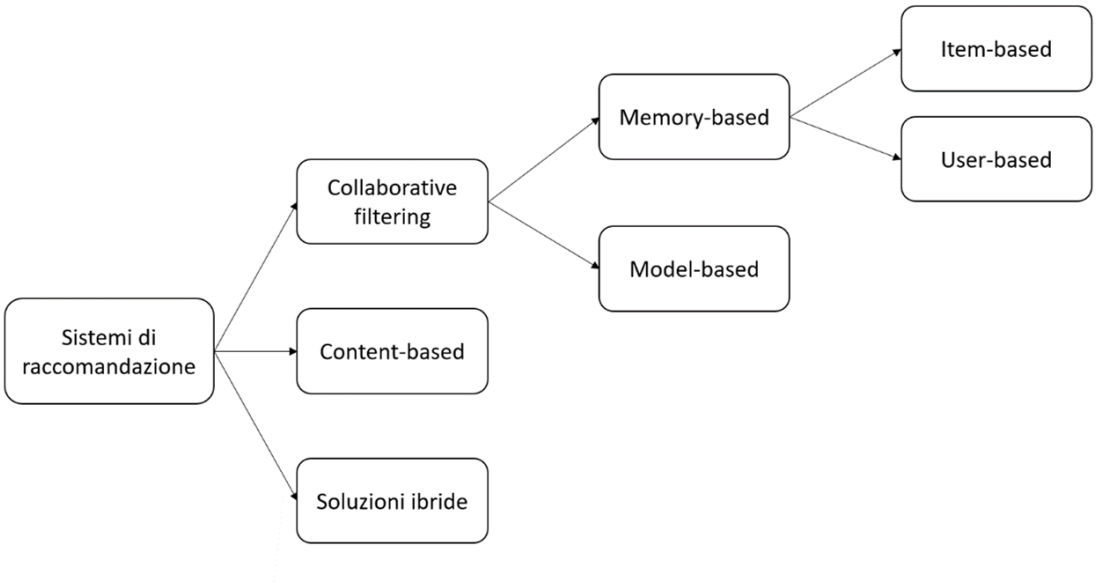

Figura 1.5: Schema riassuntivo dei sistemi di raccomandazione

Capitolo 2

Strumenti utilizzati

2.1 Durante la creazione del modello

2.1.1 Google Colab

Per creare un modello di Machine Learning è necessaria molta potenza di calcolo che potrebbe non essere disponibile nel proprio computer. Infatti specialmente l'addestramento di reti neurali con dataset di grandi dimensioni diventa molto oneroso in termini di risorse sia di calcolo che temporali. Per risolvere questo problema esistono molti servizi cloud che forniscono potenza di calcolo, solitamente a pagamento o con varie restrizioni. In alternativa ad essi c'è Google Colab, una piattaforma che, seppur con alcune limitazioni, permette di eseguire codice direttamente sul cloud sfruttando la potenza di calcolo fornita da Google. Colab si basa sul progetto Open Source Jupyter, è gratuito e permette di scrivere ed eseguire codice Python direttamente da un browser.[10]

2.1.2 TensorFlow

TensorFlow è una complessa libreria di Machine Learning che permette di addestrare ed eseguire grandi modelli di reti neurali in modo efficiente, distribuendo i calcoli su centinaia

di server multi-GPU [1].[8]

2.1.3 TensorFlow Lite

TensorFlow Lite è un set di strumenti che abilita l'apprendimento automatico sul dispositivo aiutando gli sviluppatori a eseguire i loro modelli su dispositivi mobili, embedded ed edge. Le caratteristiche principali di TensorFlow Lite sono:

- Ottimizzazione per l'apprendimento automatico sui dispositivi mobile o edge;
- Supporto per più piattaforme: dai dispositivi con sistema operativo Android, iOS o Linux Embedded ai microcontrollori;
- Supporto per diversi linguaggi di programmazione tra i quali Java, Swift, Objective-C, C++ e Python;
- Alte prestazioni grazie all'utilizzo di accelerazioni hardware e ottimizzazione del modello;

TensorFlow Lite risolve alcune complicazioni che si riscontrano quando si cerca di eseguire modelli di Machine Learning su dispositivi mobili, tra le quali:

- Eliminazione della latenza con i server, ovvero il ritardo tra l'invio dei dati al server e il ricevimento della risposta. Non essendoci la necessità di inviare i dati ad un server esterno (dato che l'apprendimento e l'elaborazione dei dati avvengono direttamente sul dispositivo) il tempo necessario per ottenere le prediction viene ridotto drasticamente;
- Privacy: siccome i dati non lasciano mai il dispositivo dell'utente, la privacy è massimizzata. Garantisce che tutti i dati sensibili o personali vengano elaborati internamente senza essere condivisi con server remoti.

[1] Tipologia di server dedicato all'elaborazione di compiti che richiedono elevate prestazioni di calcolo parallelo

- Connettività: l'assenza della necessità di una connessione a Internet è un aspetto fondamentale di questa modalità di apprendimento automatico. Infatti in alcune situazioni, potrebbe non essere possibile o pratico avere una connessione stabile. Eseguire gli algoritmi di Machine Learning sul dispositivo consente di svolgere attività di apprendimento e inferenza anche in assenza di connessione.

- Dimensioni del modello: la riduzione delle sue dimensioni è essenziale per consentire il suo utilizzo su dispositivi con risorse limitate, come dispositivi mobili o embedded, dove generalmente sono minori rispetto ai computer. Un modello che occupa meno spazio di archiviazione e richiede meno risorse computazionali per l'elaborazione, consente di essere implementato su più dispositivi, anche in quelli con più restrizioni di spazio.

- Consumo energetico: l'efficienza energetica è importante per garantire una durata della batteria accettabile nei dispositivi mobili. Il Machine Learning quando eseguito su dispositivo deve cercare di ridurre al minimo il consumo energetico durante l'elaborazione dei dati e l'inferenza del modello. Inoltre eliminando la necessità di trasmettere dati, si riduce ulteriormente il consumo energetico.[24]

Quantizzazione

Una delle modalità con cui TensorFlow riesce a diminuire la dimensione del modello è la **quantizzazione** a discapito in alcuni casi di un peggioramento delle prestazioni. La quantizzazione arrotonda o rappresenta con meno bit i pesi del modello (coefficienti tra i neuroni) occupando quindi meno memoria. Permette anche la riduzione della latenza, intesa in questo caso come la quantità di tempo necessaria per eseguire una singola inferenza con un determinato modello, e ciò ha un impatto positivo sul consumo energetico. Ovviamente è importante bilanciare la quantizzazione in modo da mantenere un equilibrio tra efficienza e accuratezza.[19]

2.1.4 Keras

Keras é una libreria ad alto livello per il Deep Learning, rende molto semplice addestrare ed eseguire modelli di reti neurali. Sviluppata da François Chollet e rilasciata come progetto open source nel marzo del 2015, ottenne subito grande popolarità per la semplicità d'uso e flessibilità. Keras copre ogni fase del flusso di lavoro di machine learning, dall'elaborazione dei dati all'ottimizzazione degli iperparametri fino alla distribuzione.[13] Per eseguire la grande quantità di calcoli richiesta nell'addestramento delle reti neurali si affida come backend a TensorFlow o altre librerie di calcolo.

Attualmente TensorFlow viene direttamente fornito con la propria implementazione di Keras, chiamata tf.keras. La libreria tf.keras utilizza TensorFlow come backend con il vantaggio di alcune funzionalità aggiuntive come ad esempio l'API Data TensorFlow che rende semplice caricare e preprocessare i dati. L'implementazione tf.keras all'interno di TensorFlow unisce quindi la semplicità e la flessibilità di Keras con le potenzialità di TensorFlow, semplificando lo sviluppo, l'addestramento e l'ottimizzazione di modelli di machine learning.[8]

2.2 Durante la sviluppo dell'applicazione

2.2.1 Flutter

Flutter è un framework open source di Google rilasciato nel maggio 2017. Permette di creare applicazioni native sia per Android che per iOS scrivendo un unico codice.[16] Caratteristica peculiare di Flutter è l'*hot reload* che permette di verificare gli effetti dei cambiamenti del codice senza ricompilarlo preservando quindi lo stato. L'hot reload è reso possibile attraverso l'uso di una Virtual Machine.[11]

Per utilizzare Flutter è necessario conoscere il linguaggio di programmazione Dart, presentato da Google nel 2011.[5, 6]

Ho optato per l'uso di Flutter poiché avevo già familiarità con questo framework. Questa scelta mi ha consentito di velocizzare il processo di sviluppo dell'applicazione, permettendomi di concentrarmi maggiormente sulla creazione del modello.

2.2.2 Android Studio

Android studio è un famoso ambiente di sviluppo integrato[2] (IDE) rilasciato nel maggio 2013 da Google e Jetbrains. Siccome per lo sviluppo dell'applicazione ho deciso di utilizzare Flutter, ho scelto Android Studio come IDE essendo quello raccomandato da Google.[1]

2.3 Dataset

I dataset svolgono un ruolo cruciale nell'ambito del Machine Learning poiché costituiscono il fondamento su cui vengono costruiti e addestrati i modelli di apprendimento automatico La qualità di un dataset riveste importanza vitale per la creazione di modelli affidabili e precisi.

Uno dei motivi che mi ha spinto a sviluppare un modello basato sul collaborative filtering per la raccomandazione di film è la disponibilità di ottimi dataset, tra i quali il database MovieLens[17]. Questo database è stato appositamente creato da un gruppo di ricerca dell'Università del Minnesota con l'obiettivo di potenziare e perfezionare i sistemi di raccomandazione. Inoltre, il database MovieLens è soggetto a costanti aggiornamenti e si presenta in diverse versioni che si differenziano principalmente per le dimensioni dei dati in esse contenuti.[18]

Per il mio progetto ho scelto di adottare la versione più recente disponibile (settembre 2018) che avesse un numero limitato di film e recensioni. Include 100,000 valutazioni di film effettuate da circa 600 utenti e contente un catalogo di 9,000 film.

Scaricando il MovieLens database si ottengono quattro tabelle diverse:

- links.csv che contiene anche informazioni di IMDb[3];

[2]Applicazione software che offre un insieme di strumenti e funzionalità integrati per semplificare e agevolare il processo di sviluppo software. Un IDE fornisce un ambiente completo in cui gli sviluppatori possono scrivere, testare, eseguire la fase di debug e distribuire il loro codice.

[3]Database online che raccoglie informazioni dettagliate su film, programmi televisivi, attori, registi e molto altro legato all'industria cinematografica e televisiva. Fondato nel 1990, IMDb è diventata una risorsa popolare per gli appassionati di cinema e per coloro che cercano informazioni su film e serie TV.

- movies.csv contente i film;
- ratings.csv con le valutazioni;
- tags.csv che contiene alcuni tag per i film come ad esempio la presenza di un attore o resta famoso o un'argomento;

I dataset "links" e "tags" non sono stati utilizzati. Nonostante il dataset "tags" possa servire per migliorare la precisione dei sistemi di raccomandazione ho voluto focalizzarmi sulla creazione di un modello basato esclusivamente sulle recensioni degli utenti.

Il dataset movies contiene 3 colonne:

- movieId: numero intero che rappresenta il film. Da evidenziare che i numeri non seguono un ordine continuo;
- title: stringa contente il titolo del film;
- genres: stringa con i generi del film. Questa colonna non è servita per la realizzazione del modello per lo stesso motivo per cui non è servito il dataset "tags".

Alcune righe del dataset movies sono presenti nella Tabella 2.1. Ogni film può contenere più di un genere ma non tutti sono visualizzabili nella tabella.

Tabella 2.1: Alcune righe del dataset movies

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation
2	Jumanji (1995)	Adventure|Children
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama
...		
1617	L.A. Confidential (1997)	Crime|Film-Noir
1619	Seven Years in Tibet (1997)	Adventure|Drama|War
...		
193585	Flint (2017)	Drama
193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

Il dataset ratings contiene 4 colonne:

- userId: numero intero che identifica in modo univoco un utente del dataset. Gli userId sono sequenziali e variano da 1 a 610;
- movieId: identificativo del film identico a quello presente nel dataset "movies";
- rating: la valutazione assegnata da un utente a un film. varia nell'intervallo da 0.5 a 5;
- timestamp: numero intero che rappresenta univocamente la recensione, inutile ai fini del modello di raccomandazione che avevo intenzione di creare;

Alcune righe del dataset rating sono presenti nella Tabella 2.2.

Tabella 2.2: Alcune righe del dataset rating

userId	movieId	rating	timestamp
1	1	4.0	964982703
1	3	4.0	964981247
1	6	4.0	964982224
1	47	5.0	964983815
...			
342	3016	2.0	1042821991
342	3157	3.0	1042822234
...			
610	168250	5.0 1494273047	
610	168252	5.0	1493846352
610	170875	3.0	1493846415

Capitolo 3

Realizzazione del modello

La realizzazione del modello di raccomandazione è stata suddivisa in diverse fasi. Inizialmente, è stata eseguita la fase di preprocessing dei dati, in cui sono state applicate varie operazioni e trasformazioni ai dati grezzi presenti nei dataset per renderli adatti all'utilizzo da parte del modello. Successivamente è stata sviluppata l'architettura[1], la quale è stata poi sottoposta alla fase di addestramento e di valutazione. La fase di progettazione dell'architettura è strettamente connessa con le fasi di addestramento e di valutazione del modello, infatti solamente dopo aver completato questi tre passaggi si è in grado di determinare se l'architettura scelta permette di ottenere dei risultati soddisfacenti oppure se necessita di ulteriori ottimizzazioni.

3.1 Preprocessing

Dopo aver importato le librerie necessarie, nella fase iniziale del codice i dataset sono stati elaborati per ottenere un input conforme alle specifiche del modello. I dataset "ratings" e "movies" sono stati caricati all'interno di un notebook di Colab e del primo è stata eliminata la colonna `timestamp` dato che, come già precedentemente spiegato, tale attributo

[1]L'architettura di un modello nel contesto del Machine Learning si riferisce alla sua struttura, cioè agli strati che compongono la rete neurale, e agli iperparametri, ovvero i valori selezionati prima dell'addestramento che influenzano il processo di apprendimento stesso.

non è rilevante per il modello da realizzare. I due dataset sono stati salvati rispettivamente nelle variabili `rating` e `movies` attraverso la funzione di Pandas[2] `read_csv` che crea un DataFrame[3] a partire da un file csv.

```
import pandas as pd
import numpy as np
import tensorflow as tf

rating = pd.read_csv("/content/ratings.csv",
            names=["userId","movieId","rating","timestamp"],skiprows=1)
rating = rating.drop('timestamp', axis=1)

movies = pd.read_csv("/content/movies.csv",
            names=["movieId","title","genres"], skiprows=1)
```

In seguito, i due dataset sono stati fusi in un unico dataset combinando le informazioni in base alla colonna `movieId`. Attraverso questa procedura ogni recensione (che corrisponde ad una riga del dataset `rating`) è stata arricchita con le colonne del dataset `movies` relative al film con lo stesso `movieId`. Questo passaggio può apparire insignificante ma in realtà consente di individuare e rimuovere alcuni film che non hanno mai ricevuto recensioni da parte degli utenti. Quei film non avendo recensioni non possono essere considerati in un modello basato sul collaborative filtering e perciò devono essere esclusi.

```
data = pd.merge(rating, movies, on='movieId')
```

Attraverso questa semplice procedura sono stati eliminati i film presenti nel dataset `movies` che non presenti nel dataset `rating` e il numero totale di film utilizzati nel modello è quindi passato da 9742 a 9724.

Non si poteva escludere a priori la situazione opposta in cui alcuni film recensiti dagli utenti non fossero presenti nel dataset `movies`. In tal caso sarebbe stato necessario

[2]Pandas è una libreria open-source per la manipolazione e l'analisi dei dati in Python. Offre strutture dati flessibili e potenti, come i DataFrame e grazie a questa libreria è possibile caricare, trasformare, filtrare e aggregare dati in modo intuitivo e efficiente.

[3]Struttura di dati bidimensionale che permette di rappresentare i dati in modo simile a una tabella con righe e colonne o a un array di due dimensioni

eliminare la recensione poiché non avremmo avuto tutte le informazioni relative al film (come titolo, genere e anno di pubblicazione). Tuttavia il numero di recensioni è rimasto costante così come quello degli utenti (610) e questo implica che tutti i film recensiti erano presenti nel dataset `movies`.

L'accesso a film specifici nel dataset non era però agevole poiché i `movieId` assegnati erano numeri casuali e non seguivano un ordine sequenziale. È stato quindi necessario mapparli in una sequenza da 0 a 9723 e per farlo è stata utilizzata la libreria LabelEncoder[23] di scikit-learn. Dopo questo passaggio è stata introdotta una nuova colonna chiamata `movie` nel dataset contenente i nuovi ID sequenziali per i film. Gli `userId` erano invece numeri sequenziali da 1 a 610 ma siccome labelEncoder mappa il primo elemento in 0 ci sarebbero stati due diversi indici per accedere al primo elemento per le due colonne di ID. Per ovviare a ciò, la stessa procedura di label encoding è stata applicata anche agli `userId`, al fine di ottenere un array di numeri interi zero-based, assicurando coerenza con la numerazione dei film. In questo caso la colonna aggiuntiva ottenuta è `user` che banalmente si discosta di uno da `userId`.

```
from sklearn.preprocessing import LabelEncoder

user_encoder = LabelEncoder()
movie_encoder = LabelEncoder()

data['user'] = user_encoder.fit_transform(data['userId'])
data['movie'] = movie_encoder.fit_transform(data['movieId'])
```

Alcune righe della tabella `data` così ottenuta dopo il preprocessing dei dati sono presenti nella Tabella 3.1. Da notare come nella colonna `genres` è visualizzabile solo il primo genere del film nonostante ogni film possa averne anche più di uno.

Il modello necessita di un dataset per la fase di addestramento e di un altro separato per la fase di test. Viene quindi utilizzata la funzione `train_test_split`[22] della libreria scikit-learn per suddividere il dataset `data` in un insieme di addestramento `train` e un insieme di test `test`. Questa suddivisione è fondamentale nel processo di addestramento e valutazione dei modelli di Machine Learning perché è importante verificare che il modello funzioni correttamente anche con dati diversi da quelli usati durante la fase d'addestra-

Tabella 3.1: Alcune righe del dataset Data

userId	movieId	rating	title	genres	user	movie
1	1	4.0	Toy Story (1995)	Adventure	0	0
5	1	4.0	Toy Story (1995)	Adventure	4	0
...						
380	6	5.0	Heat (1995)	Action	379	5
385	6	3.0	Heat (1995)	Action	384	5
386	6	3.0	Heat (1995)	Action	385	5
...						
610	160836	3.0	Hazard (2005)	Action	609	9324
610	163937	3.5	Blair Witch (2016)	Horror	609	9371
610	163981	3.5	31 (2016)	Horror	609	9372

mento. In assenza di questa procedura il modello non sarebbe in grado di apprendere in modo efficace.

Il parametro `test_size` indica che vengono utilizzati l'80% delle righe per la fase di addestramento del modello e il 20% per la valutazione.

```
1 from sklearn.model_selection import train_test_split
2 train, test = train_test_split(data, test_size=0.2, random_state=42)
```

3.2 Creazione del modello

La scelta dell'architettura del modello è un aspetto cruciale per ottenere prestazioni ottimali: per tale motivo ho preferito utilizzare un modello esistente anziché crearne uno completamente da zero. L'architettura del modello è stata ricavata da un articolo pubblicato su Medium[4] [4] il quale presentava un modello di raccomandazione per dei luoghi da visitare in Indonesia ed è stato adattato per la raccomandazione di film. Il seguente codice è l'architettura del modello originale adattata:

[4]Medium è una piattaforma editoriale online americana sviluppata da Evan Williams e lanciata nell'agosto 2012

```
import tensorflow as tf
from keras.layers import Input, Embedding, Flatten, Dense, Concatenate
from keras.models import Model

movies_input = Input(shape=[1], name="movies_input")
movies_embedding = Embedding(n_movies+1, 5,
                        name="movies_embedding")(movies_input)
movies_vec = Flatten(name="flatten_movies")(movies_embedding)
# creating user embedding path
user_input = Input(shape=[1], name="user_input")
user_embedding = Embedding(n_users+1, 5,
                        name="user_embedding")(user_input)
user_vec = Flatten(name="flatten_users")(user_embedding)
# concatenate features
conc = Concatenate()([movies_vec, user_vec])
# add fully-connected-layers
fc1 = Dense(256, activation='relu')(conc)
fc2 = Dense(128, activation='relu')(fc1)
fc3 = Dense(128, activation='relu')(fc2)
out = Dense(1)(fc3)
```

Analizzando il modello possiamo vedere la presenza di diversi livelli nella rete neurale:

- `Input` è utilizzato per definire l'input della rete neurale ed è il primo livello della rete. Viene utilizzato per specificare la forma e il tipo di dati che la rete dovrà ricevere in ingresso attraverso il parametro `shape`. In questo caso essendo `shape=[1]` l'input è unidimensionale e ha un singolo valore per ciascun elemento (ID di un film o di un dell'utente);

- `Embedding` permette di convertire le informazioni categoriche dei dati, come parole, ID di utenti o elementi, in vettori numerici. Questi vettori sono appresi durante il processo di addestramento del modello e catturano relazioni implicite tra i dati. I primi due parametri rappresentano rispettivamente la dimensione di input e la dimensione del vettore di output per ogni input: in questo caso il livello prende gli ID come input e li trasforma in vettori numerici a 5 dimensioni;

- `Flatten` serve a trasformare un tensore multidimensionale in un tensore unidimensionale. Viene utilizzato per convertire gli embedding degli utenti (che potrebbero

essere in una rappresentazione multidimensionale) in un vettore unidimensionale. In questo caso li appiattisce da 5 a 1;

- `Concatenate` utilizzato per unire più tensori in un unico tensore lungo una dimensione specifica. In questo caso il tensore conterrà informazioni sia sui film che sugli utenti, consentendo al modello di apprendere le relazioni tra i due per effettuare raccomandazioni personalizzate basate sulle preferenze degli utenti e potenzialmente sulle caratteristiche dei film ma in questo modello non vengono utilizzate.

- `Dense` un livello nascosto contente con un numero di neuroni pari al valore del primo parametro e funzione di attivazione ReLU[5]. Prende in input o il tensore risultante dall'operazione di concatenazione degli Embedding dei film e degli utenti o il tensore di output del livello Dense precedente. Questo livello aggiunge ulteriori elaborazioni e complessità al modello, permettendo al modello di apprendere rappresentazioni più sofisticate e fare raccomandazioni più accurate.

3.2.1 Scelta della Loss Function

La **loss function**, o funzione di perdita, è una misura che quantifica l'errore tra le previsioni del modello e i valori target di addestramento. L'obiettivo durante la creazione di un modello è la minimizzazione della loss function poiché implica il miglioramento delle prestazioni.

Tra le molte funzioni tra cui poter scegliere, ho deciso di utilizzare **L'errore quadratico medio**, in inglese mean squared errore (MSE), definito dalla seguente formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$$

dove x rappresenta la valutazione dell'utente, y l'output del modello e n il numero di recensioni.

[5]funzione di attivazione che converte i valori negativi in zero e lascia inalterati i valori positivi

La scelta di utilizzare come loss function il mean squared error è dovuta al fatto che penalizza gli errori maggiori più pesantemente rispetto agli errori minori. Ciò implica che le prediction "sbagliate" (ovvero quelle che hanno un grande errore rispetto ai valori target) hanno un impatto maggiore sull'MSE rispetto a quelle con errori minori.

Siccome il mio scopo era quello di creare un sistema di raccomandazione che consigliasse dieci film, è facile intuire che l'obiettivo era ottenere previsioni che fossero sufficientemente accurate per identificare se un film potesse piacere o meno a un utente, piuttosto che mirare a singole previsioni estremamente precise a scapito di altre ampiamente errate. L'errore quadratico ha permesso quindi di minimizzare gli errori più grandi. Un altro pregio del MSE è la sua facile interpretazione: più piccolo è, più precise sono in media le prediction.

Ho provato comunque ad allenare e valutare il parametro anche con una funzione di perdita diversa come **l'errore assoluto medio**, meglio conosciuto con il termine inglese mean absolute error (MAE), ma le prediction risultanti erano simili perciò ho preferito utilizzare il MSE per le motivazioni precedentemente elencate.

3.2.2 Addestramento del modello

Viene utilizzata una variabile nominata `history` per salvare l'addestramento del modello e poter visualizzare in un secondo momento il grafico della loss function. Per allenare il modello viene invocata la funzione `fit` di Keras. Come parametri della funzione servono tre array della stessa grandezza dato che una recensione è composta da un utente, un film e una valutazione. Uso quindi il dataset `train` ottenuto precedentemente per questo scopo e seleziono le colonne `user`, `movie` e `rating`. Più precisamente la coppia `user-movie` rappresenta l'input mentre `rating` rappresenta l'output desiderato, il target. All'inizio dell'addestramento i pesi delle connessioni tra i neuroni della rete neurale vengono inizializzati casualmente. Ad ogni iterazione si utilizza la loss function per calcolare l'errore tra la previsioni del modello e l'output desiderato e i pesi delle connessioni vengono aggiornati in base all'errore calcolato e al metodo di ottimizzazione utilizzato. Il procedimento di calcolo della loss function e aggiornamento dei pesi viene ripetuto per un numero di

iterazioni pari al parametro `epochs`.

```
history = model.fit([train['user'], train['movie']], train['rating'],
                        epochs=10, batch_size=8)
```

Con l'addestramento del modello salvato in `history` è stato possibile usare la libreria matplotlib[15] per disegnare il grafico della loss function in funzione delle epoche.

```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.xlabel("Epochs")
plt.ylabel("Training Error")
```

Il grafico del precedente codice può essere visualizzato nella Figura 3.1.

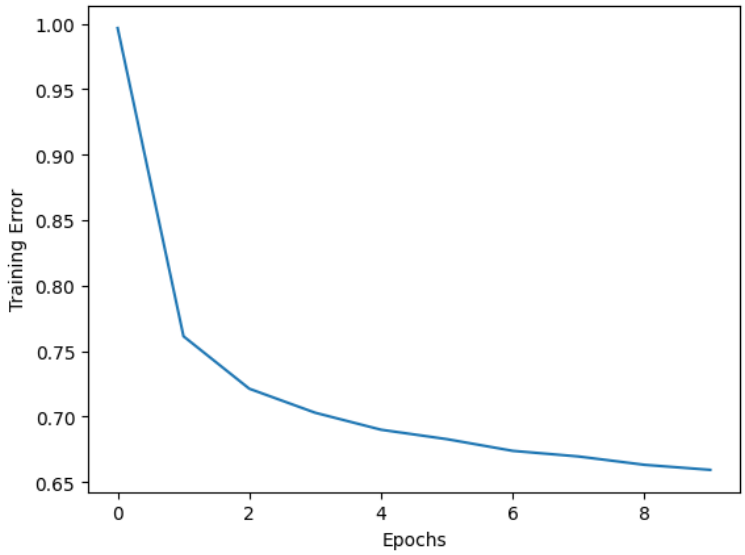

Figura 3.1: Loss function durante l'addestramento del modello in funzione delle epoche

3.2.3 Valutazione del modello

Per valutare le prestazioni del modello è stata utilizzata la funzione `evaluate` di Keras. Come parametri servono tre array della stessa tipologia usata per l'addestramento ma i valori in questo caso sono quelli del dataset di test. L'output di tale funzione è il valore della loss function.

```
model.evaluate([test["user"], test["movie"]], test["rating"])
```

La loss però non è l'unica metrica che ho utilizzato per valutare il modello, durante la sua fase di creazione ho tenuto in considerazione altri parametri come ad esempio il numero di prediction appartenenti al corretto range delle valutazioni. Il modello di raccomandazione creato permette di ottenere come output il rating previsto di un utente per un determinato film usando la funzione `predict` che verrà analizzata in seguito. Eseguendo molte previsioni e quindi ottenendo una possibile valutazione di un film (si ricorda che i valori delle recensioni sono compresi da 0.5 a 5 estremi inclusi) l'output di alcuni modelli non apparteneva al corretto intervallo delle valutazioni. Per l'architettura del modello utilizzata, gli output dovevano essere nello stesso range delle recensioni degli utenti ma alcune prediction erano persino maggiori di 6 o negative. Tali prediction sbagliate che da ora poi saranno chiamate outliers erano un chiaro segnale di malfunzionamento del sistema. Modelli con un alto numero di outliers venivano scartati a priori anche nel caso in cui il valore della loro loss function sull'insieme di test fosse buona. In generale comunque un alto numero di outliers era correlato ad un pessimo valore della loss sul test e viceversa modelli un numero basso o nullo di outliers avevano generalmente una loss migliore.

3.2.4 Il problema dell'overfitting

L'overfitting è un problema comune nel Machine Learning. Si verifica quando un modello ha prestazioni molto buone (un basso valore della loss) nei dati di addestramento ma non è in grado di fornire previsioni accurate per nuovi dati.[3]

Durante la fase di test il modello presentato precedentemente ha dimostrato di essere estremamente inefficiente: è soggetto ad un forte overfitting al punto da risultare completamente inutile nel predire le valutazioni per nuovi film. La loss function nell'insieme di train è inferiore a 0.08, mentre in quello di test supera addirittura 1. Fondamentalmente, il modello memorizza le recensioni incontrate durante l'addestramento, ma fallisce nel riconoscere le relazioni implicite tra i dati necessarie per effettuare previsioni accurate su nuovi film.

Per questa ragione è stato necessario apportare modifiche all'architettura del modello la cui versione finale è stata ottenuta eseguendo iterativamente le tre fasi di scelta dell'architettura, addestramento e valutazione del modello.

3.3 Cambiamenti nell'architettura

Il numero di iperparametri del modello è 104,033, molto alto per le sole 100836 recensioni utilizzate, quindi per prima cosa quindi ho cercato di ridurre tale numero semplificando di conseguenza il modello. Circa la metà degli iperparametri risiede nei due livelli di `Embedding`, pertanto ho iniziato a ridurre la loro dimensione di output decrementandola progressivamente di un'unità per tentativo. I risultati sperimentali hanno dimostrato una chiara correlazione tra la presenza dell'overfitting e la dimensione di output degli `Embedding`. Diminuendo la dimensione di output da 5 a 1 la differenza di tra il valore della loss sul test e sul train è crollata. I loro valori sono diventati rispettivamente circa 0.5687 e 0.8210 rispetto ai precedenti 0.08 e 1.2.

È evidente che raggiungere un valore di loss sul set di addestramento simile a quello ottenuto dal modello iniziale non è possibile dato che quel modello soffre di overfitting e non può quindi essere considerato come un punto di riferimento. L'ideale sarebbe ottenere un modello che performi quasi allo stesso modo con dati di train e di test.

Con il primo passaggio di semplificazione del modello i valori delle due loss si sono notevolmente avvicinati ma la loro differenza rimane considerevole e l'overfitting persiste, anche se in misura notevolmente inferiore rispetto al modello iniziale.

Tralasciando tentativi che non hanno portato a risultati, un'altra strategia di semplificazione è stata la riduzione dei livelli `Dense` e del numero di neuroni all'interno di ciascun livello. Con l'eliminazione di un livello `Dense` i valori delle loss sono scesi a circa 0.5729 per l'insieme di test e 0.8198 per quello di addestramento, un miglioramento che seppur minimo ha contribuito all'ottimizzazione del modello. La ragione per cui è stato rimosso un solo livello `Dense` così come la scelta dei relativi parametri, è perché sperimentalmente questa configurazione ha dimostrato di ottenere il valore della loss più basso. Questa è la

versione finale del codice contenente solamente 3 livelli `Dense` e con parametri di molto minori rispetto a quelli iniziali.

```
fc1 = Dense(32, activation='relu')(conc)
fc2 = Dense(16, activation='relu')(fc1)
out = Dense(1)(fc2)
```

Un altro perfezionamento significativo è stato ottenuto cambiando i parametri della funzione `fit` che si occupa dell'addestramento del modello. Il primo parametro modificato è stato `epochs` il quale determina il numero di iterazioni dell'addestramento. Un numero maggiore di epoche implica un addestramento più approfondito che potenzialmente potrebbe ridurre entrambe le loss, a patto che il modello non inizi a soffrire di overfitting. Dopo vari tentativi, 15 è risultato sperimentalmente il valore migliore per la diminuzione del valore della loss sul test e contemporaneamente per la riduzione della differenza tra le due loss. I valori risultanti della loss sull'insieme di test con 100, 50, 30, 20 e 15 epoche sono rispettivamente: 0.820, 0.782, 0.779, 0.774 e 0.769. Diminuendo ulteriormente il valore delle epoche il valore della loss ricominciava leggermente a salire.

Discorso analogo per quanto riguarda il parametro `batch_size` che rappresenta il numero di campioni che vengono utilizzati in una singola iterazione durante il processo di addestramento, il cui valore migliore è risultato sperimentalmente essere 8 dopo averlo testato con i valori di 64, 32, 16, 8 e 4. Il miglioramento del valore della loss sul test è stato però molto modesto passando dal valore di 0.7694 a 0.7637.

Un ulteriore leggero miglioramento è stato introdotto successivamente, ed ha permesso di portare le due loss da 0.6072 e 0.7637 a 0.6336 e 0.7550, riducendo contemporaneamente sia la loss sul test che differenza tra le loss. Tale risultato è stato ottenuto aggiungendo dei livelli `Dropout`. I livelli `Dropout` funzionano disattivando casualmente alcune unità (neuroni) all'interno del livello durante ogni passaggio di addestramento e sono stati inseriti tra i livelli `Dense`.

A questo punto, ero soddisfatto dei miglioramenti ottenuti con le modifiche apportate all'architettura. Ho quindi iniziato a valutare manualmente le predizioni del modello, confrontando gli output di diversi film tra vari utenti. Ho selezionato alcuni utenti, eseguito le previsioni per ciascun film e le ho ordinate in ordine decrescente. Dopo aver svolto

questo passaggio ho notato che i film raccomandati agli utenti erano molto simili, se non addirittura identici. Il problema risiedeva probabilmente nella scelta iniziale di impostare la dimensione di output degli embedding pari a 1. Optare per una dimensione di 5 avrebbe permesso a ciascun ID di essere mappato in un array di 5 elementi, consentendo al modello di catturare meglio le relazioni tra gli utenti e migliorare così il sistema di raccomandazione. La riduzione di questa dimensione a 1 era stata essenziale per ridurre significativamente l'overfitting nella fase iniziale dello sviluppo dell'architettura ma a seguito delle successive modifiche, come l'introduzione dei livelli Dropout, la diminuzione del numero dei livelli Dense e soprattutto la modifica del numero di epoche e delle dimensioni del batch nella funzione fit, non era più essenziale mantenere una dimensione di embedding pari a 1. Questo perché i cambiamenti apportati all'architettura hanno permesso di ridurre l'overfitting al punto che eventuali dimensioni di output nei livelli embedding non comportavano più problemi di overfitting. Ho allora scelto di aumentare la dimensione fino al suo valore iniziale pari a 5 risolvendo parzialmente il problema. Tra i film raccomandati ci si può facilmente accorgere che molti sono ancora in comune tra i diversi utenti ma il fatto che le raccomandazioni siano simili non rappresenta necessariamente un problema. Ho preferito abbassare ulteriormente il numero di epoche riducendolo da 15 a 10 perchè con l'ultima modifica eseguita all'architettura era il valore che faceva ottenere risultati migliori.

Questi ultimi cambiamento mi hanno anche permesso di abbassare ulteriormente la loss del test portando il modello finale ad avere dei valori della loss sul train e sul test pari a 0.6230 e 0.7391. Il codice finale dell'architettura è il seguente:

```
from tensorflow.keras.layers import Dropout

# creating book embedding path
movies_input = Input(shape=[1], name="Movies-Input")
movies_embedding = Embedding(n_movies+1, 5,
                        name="Movies-Embedding")(movies_input)
movies_vec = Flatten(name="Flatten-Movies")(movies_embedding)

# creating user embedding path
user_input = Input(shape=[1], name="User-Input")
user_embedding = Embedding(n_users+1, 5,
```

```
                          name="User-Embedding")(user_input)
user_vec = Flatten(name="Flatten-Users")(user_embedding)

# concatenate features
conc = Concatenate()([movies_vec, user_vec])

# add fully-connected-layers
dropout0 = Dropout(0.1)(conc)
fc1 = Dense(32, activation='relu')(dropout0)
dropout1 = Dropout(0.1)(fc1)
fc2 = Dense(8, activation='relu')(dropout1)
out = Dense(1)(fc2)

# Create model and compile it
model = Model([user_input, movies_input], out)
model.compile('adam', 'mean_squared_error')
```

3.4 Risultati

I risultati ottenuti per quanto riguarda la loss function "mean squared error" sono già stati riportati in precedenza: 0.6230 per il train e 0.7391 per il test. Per la loss function "mean absolute error" i risultati sono invece rispettivamente 0.5613 e 0.6466 per train e test.

Tuttavia, è importante sottolineare che il solo valore della loss function non è sufficiente per valutare appieno l'efficacia e la precisione delle raccomandazioni per un utente. Per tale motivo, ho scelto di analizzare in modo approfondito le predizioni dell'utente con `userId` 1. L'obiettivo principale di questa analisi è verificare che i film raccomandati siano coerenti con le recensioni positive precedentemente fornite.

I film recensiti e la loro valutazione sono visibili nella Tabella 3.3, mentre i dieci film più consigliati dal modello e il valore della loro predizioni sono contenuti nella Tabella 3.2.

Osservando i film recensiti possiamo notare la presenza di molti film d'autore tra i quali Kill Bill: Vol. 1, Inglourious Basterds e Django Unchained diretti da Quentin Tarantino e The Departed, Shutter Island e The Wolf of Wall Street di Martin Scorsese. Non

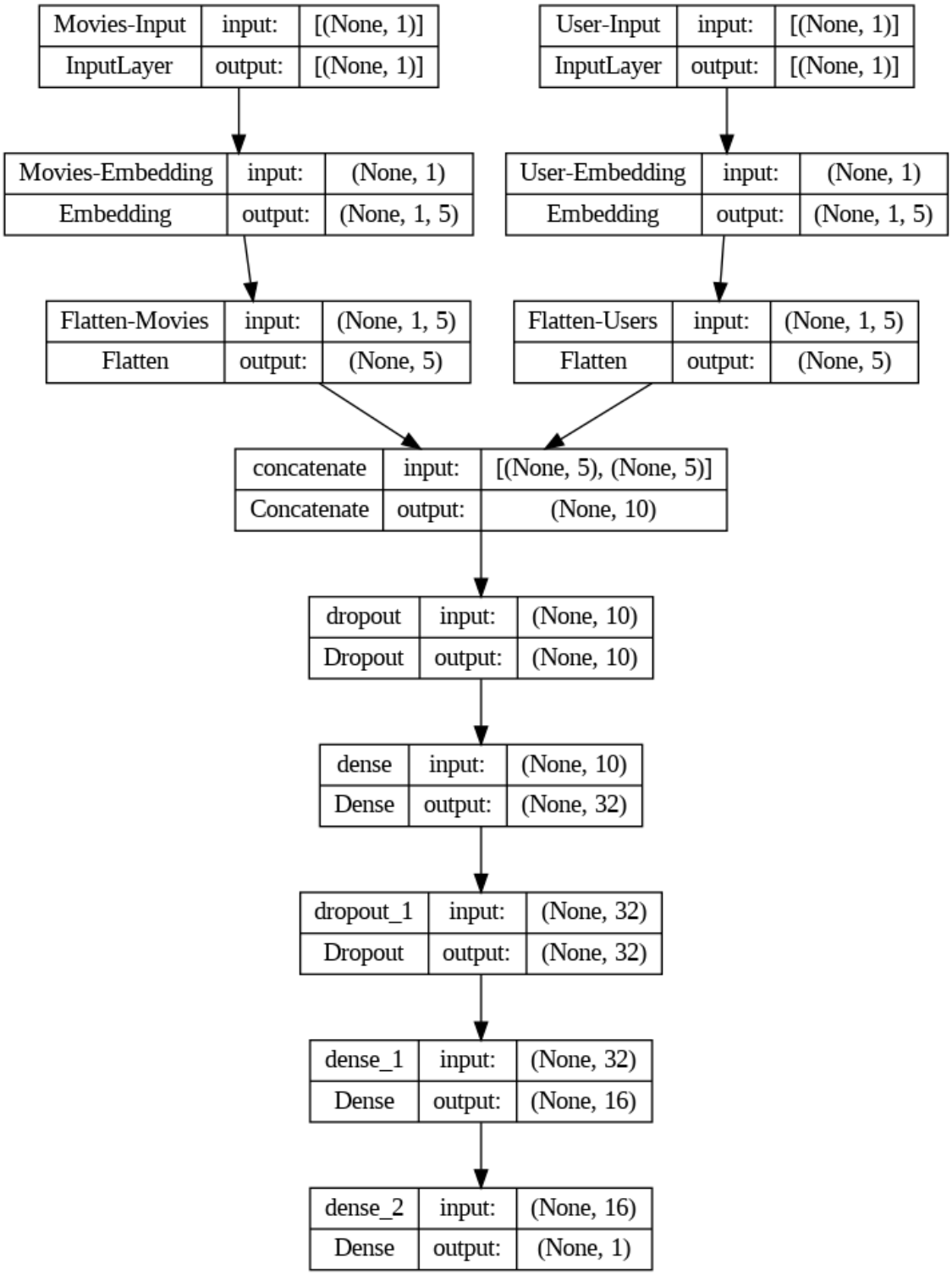

Figura 3.2: Architettura del modello

mancano inoltre opere di registi quali Christopher Nolan, Ridley Scott, Damien Chazelle. L'osservazione dei titoli e dei registi di questi film suggerisce che il modello dovrebbe consigliare opere d'autore molto apprezzate dalla critica cinematografica.

Nell'elenco dei film raccomandati emerge la presenza di pellicole quali "Five Easy Pieces," "Jules and Jim," "Fight Club," e "Il Padrino" parte I e II, entrambi diretti da Francis Ford Coppola, insieme a "Prisoners" di Denis Villeneuve. Quasi tutti questi film sono considerati dei cult oppure hanno ricevuto valutazioni positive dalla critica. Questi suggerimenti risultano quindi in linea con le preferenze dell'utente. È rilevante sottolineare come il modello sia stato in grado di suggerire tali film esclusivamente attraverso l'analisi delle valutazioni fornite dagli utenti e ricordo che non ha accesso ad informazioni quali il regista, valutazioni della critica, genere del film o altre informazioni aggiuntive che sarebbero molto utili per migliorare ulteriormente il sistema di raccomandazione.

Tabella 3.2: I dieci film più raccomandati all'utente 1

title	prediction	id
Guess Who's Coming to Dinner (1967)	4.325036	2579
Jules and Jim (Jules et Jim) (1961)	4.281626	2052
Five Easy Pieces (1970)	4.275655	2409
The Godfather (1972)	4.272176	659
Godfather: Part II, The (1974)	4.2709947	921
Prisoners (2013)	4.2692013	8237
Celebration, The (Festen) (1998)	4.2651424	1761
Sunset Boulevard (1950)	4.2616563	704
Gallipoli (1981)	4.2575107	4040
Fight Club (1999)	4.252494	2224

L'utente appena analizzato tende a dare poche valutazioni di 5 stelle, come si può infatti notare infatti nelle sue recensioni solo 5 dei 29 film recensiti hanno il punteggio massimo. La sua valutazione media è di 3.9 e il modello cerca di fare delle predizioni che si discostino di poco da quel numero. Un modello migliore potrebbe cercare di esporsi maggiormente ma è anche vero che l'utente analizzato ha un numero di recensioni basso. Per l'utente con `userId` 0 che di recensioni ne ha 232, il modello è in grado di capire maggiormente i gusti e per tale motivo alcune prediction raggiungono valori superiori a 4.9, usando quindi un range più ampio.

Tabella 3.3: Recensioni dell'utente 1

movieId	title	rating
291	Tommy Boy (1995)	4.0
2670	Gladiator (2000)	4.0
277	Shawshank Redemption, The (1994)	3.0
1283	Good Will Hunting (1997)	4.5
4607	Kill Bill: Vol. 1 (2003)	4.0
5294	Collateral (2004)	3.5
6236	Talladega Nights: The Ballad of Ricky Bobby (2006)	4.0
6298	Departed, The (2006)	4.0
6693	Dark Knight, The (2008)	4.5
6784	Step Brothers (2008)	5.0
6993	Inglourious Basterds (2009)	4.5
7137	Zombieland (2009)	3.0
7241	Shutter Island (2010)	4.0
7306	Exit Through the Gift Shop (2010)	3.0
7355	Inception (2010)	4.0
7398	Town, The (2010)	4.5
7419	Inside Job (2010)	5.0
7572	Louis C.K.: Hilarious (2010)	4.0
7750	Dark Knight Rises, The (2012)	3.5
7758	Girl with the Dragon Tattoo, The (2011)	2.5
8045	Django Unchained (2012)	3.5
8287	Wolf of Wall Street, The (2013)	5.0
8358	Interstellar (2014)	3.0
8448	Whiplash (2014)	4.0
8491	The Drop (2014)	2.0
8532	Ex Machina (2015)	3.5
8663	Mad Max: Fury Road (2015)	5.0
8810	The Jinx: The Life and Deaths of Robert Durst (2015)	5.0

Capitolo 4

Implementazione del modello in Android

4.1 Conversione del modello con TensorFlow Lite

Per poter eseguire un modello di Machine Learning realizzato con TensorFlow su dispositivi mobili è necessario convertirlo in un modello TensorFlow Lite. Per eseguire la conversione sono sufficienti poche linee di codice Python:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Specifica il percorso di salvataggio del modello TFLite
tflite_model_path = 'modelFilm.tflite'

# Salva il modello TFLite nel percorso specificato
with open(tflite_model_path, 'wb') as f:
    f.write(tflite_model)
```

Otteniamo quindi il modello salvato nel file `modelFilm.tflite` da poter utilizzare in Android Studio.

4.2 Controllo delle prediction del modello Lite

Quando il modello viene esportato nella sua versione Lite c'è il rischio che le predizioni siano meno precisione rispetto al modello originale. Perciò prima di utilizzare il modello Lite, ho creato un nuovo notebook di Colab per testarlo e verificare il suo corretto funzionamento.

Ho iniziato importando le librerie necessarie e caricando il modello nella variabile `interpreter`.

```
import numpy as np
import tensorflow as tf

# carico modello
interpreter = tf.lite.Interpreter(model_path='modelFilm.tflite')
interpreter.allocate_tensors()
```

Ho allora selezionato l'ID dell'utente per il quale desideravo ottenere le raccomandazioni; nel codice seguente viene fornito come esempio l'utente con `userId = 1`, in modo da poter comparare i risultati con quelli presentati nel capitolo precedente ottenuti dal modello originale. Vengono salvati nella variabile `filmRecensiti` tutti i film già visti dall'utente e quindi che non possono essere consigliati dal modello.

```
id_user = 1

filmRecensiti = []
for i in range(data.shape[0]):
    if id_user == data.loc[i, 'user']:
        filmRecensiti.append(data.loc[i, 'movie'])
```

L'array `movies_data` contiene numeri interi sequenziali da 0 a 9723 che rappresentano tutti i film disponibili. Successivamente vengono convertiti i due array `movies_data` e `filmRecensiti` in `set` in modo da poter eseguire l'operazione di sottrazione insiemistica. Questo processo permette di ottenere un insieme contenente solamente i film non recensiti dall'utente. l'insieme ottenuto è stato convertito nuovamente in lista e salvato nella variabile `film_da_predire`.

```
user = np.array([[float(id_user)]], dtype=np.float32)
movies_data = np.arange(9724, dtype=np.float32)

# Converti gli array in insiemi per facilitare l'operazione di differenza
elenco_film = set(movies_data)
film_da_rimuovere = set(filmRecensiti)
film_rimasti = elenco_film - film_da_rimuovere
film_da_predire = list(film_rimasti)
```

Ogni prediction rappresenta la valutazione di un film che il modello prevede per uno specifico utente. Il ciclo `for` è necessario per ottenere una prediction per ciascun film, consentendo così di selezionare successivamente i dieci film con la prediction più alta.

```
predictions = []
input_index_user = interpreter.get_input_details()[0]['index']
input_index_movies = interpreter.get_input_details()[1]['index']
output_index = interpreter.get_output_details()[0]['index']

for movie_id in film_da_predire:
    movie_input = np.array([[movie_id]], dtype=np.float32).reshape(1, 1)

    interpreter.set_tensor(input_index_user, user)
    interpreter.set_tensor(input_index_movies, movie_input)

    interpreter.invoke()

    # ottengo i risultati
    prediction = interpreter.get_tensor(output_index)[0][0]
    predictions.append([movie_id, prediction])
```

Le prediction con il valore più alto le otteniamo con il codice seguente:

```
# Ordina l'array in base al secondo elemento di ogni sottolista (prediction)
sorted_predictions = sorted(predictions, key=lambda x: x[1], reverse=True)

# Estrai gli indici dei primi 10 elementi ordinati
recommended_movies = [item for item in sorted_predictions[:10]]

# Iterare attraverso gli indici dei film raccomandati e stampare le informazioni
for movie in recommended_movies:
    print("Movie ID:", movie[0], "- Predicted Score:",movie[1])
```

il cui valore di output è:

```
Movie ID: 2579.0 - Predicted Score: 4.3250356
Movie ID: 2052.0 - Predicted Score: 4.2816267
Movie ID: 2409.0 - Predicted Score: 4.2756553
Movie ID: 659.0 - Predicted Score: 4.2721753
Movie ID: 921.0 - Predicted Score: 4.2709947
Movie ID: 8237.0 - Predicted Score: 4.2692013
Movie ID: 1761.0 - Predicted Score: 4.265142
Movie ID: 704.0 - Predicted Score: 4.2616563
Movie ID: 4040.0 - Predicted Score: 4.257511
Movie ID: 2224.0 - Predicted Score: 4.252494
```

Confrontando i risultati ottenuti con la Tabella 3.2 risulta che i film raccomandati dai due modelli sono gli stessi e persino il valore delle loro predizioni è uguale fino alla sesta cifra decimale.

Per valutare le prestazioni del modello Lite non basta però analizzare un singolo utente. Ho selezionato lo stesso insieme di test utilizzato dal modello originale nella funzione `evaluate` e ho calcolato il valore della loss del modello Lite. Il valore ottenuto è 0.7397, nel modello "non-lite" era 0.7391. Questo significa che la differenza nella precisione tra il modello Lite e quello originale è solamente dello 0.08%. Il fatto che la differenza sia così minima conferma che il processo di conversione utilizzato per creare il modello Lite è efficiente e in linea con le osservazioni empiriche fatte durante l'analisi delle previsioni dell'utente 1.

4.3 Ottenimento delle raccomandazioni in Flutter

Dopo aver verificato il corretto funzionamento del modello Lite ho iniziato lo sviluppo dell'applicazione. Di tutto il codice scritto riporto solamente la funzione che permette di ottenere le raccomandazioni per un determinato utente. I passaggi sono simili a quanto descritto per l'ottenimento delle predizioni in Python con la sola ovvia differenza di sintassi dovuta al diverso linguaggio di programmazione. Nelle righe 35 e 38 del codice vengono utilizzate le funzioni ausiliarie `csv.obtainMovieData` e `csv.obtainRatedMovies` che servono per poter ottenere le informazioni dei film a partire dall'Id. Per poter utiliz-

zare il modello è necessario importare il plugin `tflite_flutter` realizzato dal team di TensorFlow.[25]

```
void performInference(int idUser) {
  List<int> movies = List<int>.generate(9724, (i) => i);
  List<int> filmRecensiti = csv.filmRecensiti(idUser);

  Set<int> elencoFilm = Set.from(movies);
  Set<int> filmDaRimuovere = Set.from(filmRecensiti);

  Set<int> filmRimasti = elencoFilm.difference(filmDaRimuovere);
  List<int> filmDaPredire = filmRimasti.toList();

  Map<int, double> predictions = <int, double>{};
  for (int movieId in filmDaPredire) {
    var prediction = model.runInference(idUser, movieId);
    predictions[movieId] = prediction;
  }

  // Ordina la mappa in base al valore double in ordine crescente
  var sortedKeys = predictions.keys.toList()..sort((a, b) =>
            predictions[b]!.compareTo(predictions[a]!));
  // Crea una nuova mappa ordinata
  Map<int, double> predictionOrdinata = {};

  int count = 0;
  sortedKeys.forEach((key) {
    if (count < 10) {
      predictionOrdinata[key] = predictions[key]!;
    }
    count++;
  });

  // le trasformo da mappa a Lista
  recommendedMovies = predictionOrdinata.entries.map((entry) =>
            [entry.key, entry.value, "", ""]).toList();
  // ottengo le prediction
  csv.obtainMovieData(recommendedMovies);
  // ottengo i film recensiti dall'utente
  ratedMovies = [];
  csv.obtainRatedMovies(idUser, ratedMovies);
}
```

4.4 L'applicazione realizzata

L'applicazione creata è molto semplice: contiene una Tab Bar[1] con due sezioni "Recommended" e "Rated". Sulla prima è possibile visualizzare i dieci film consigliati ad un utente mentre sulla seconda i film recensiti dall'utente stesso. Per selezionare l'utente è presente una lista di Bottoni sopra la Tab Bar.

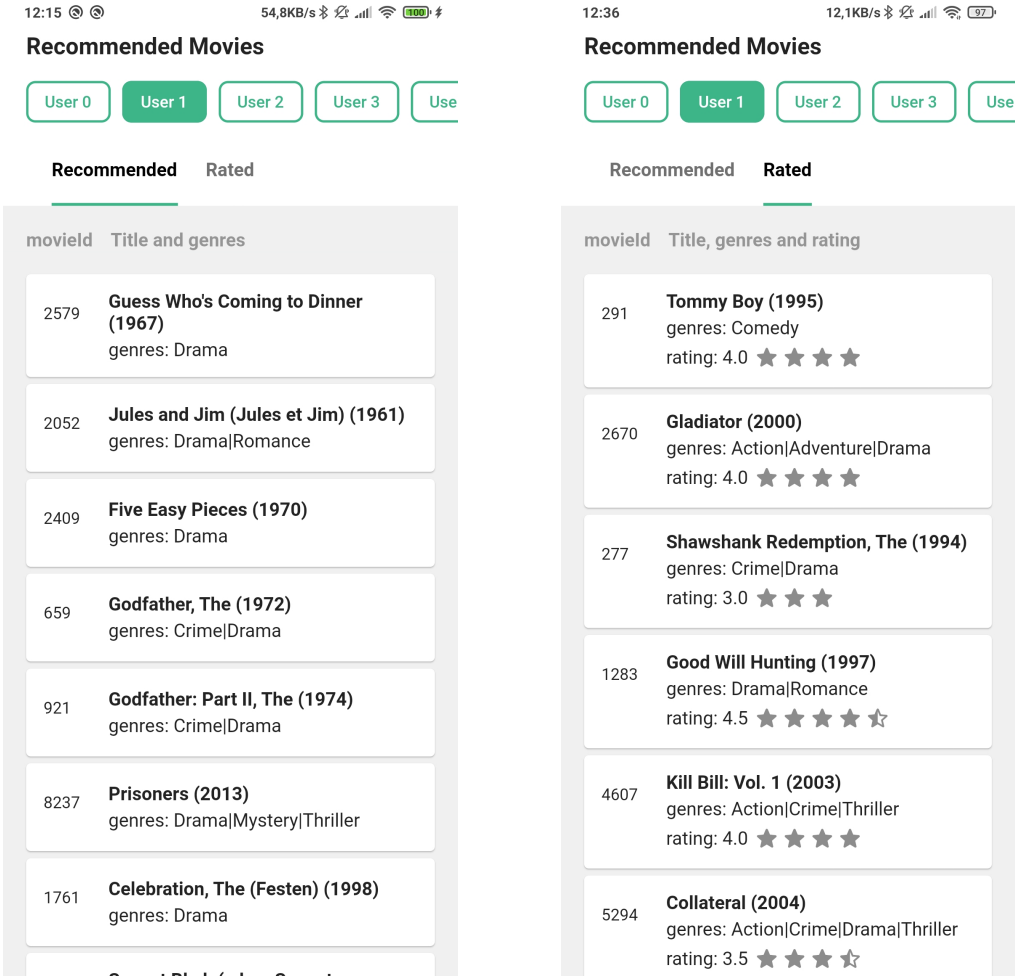

Figura 4.1: Film raccomandati all'utente 1 nell'applicazione

[1]Elemento di navigazione intuitivo presente in molte applicazioni. Con le sue schede facilmente selezionabili, permette agli utenti di accedere rapidamente a diverse sezioni dell'app

Capitolo 5

Conclusioni

Il sistema di raccomandazione creato può essere sicuramente migliorato dato che l'errore quadratico del modello "non-lite" sul test è 0.7391 mentre l'errore assoluto è 0.6466. In particolare il valore del MAE permette di capire che le predizioni ottenute non sono precisissime considerando che il range delle valutazioni è da 0.5 a 5.

Nonostante i valori della loss possano far sembrare il modello un po' approssimativo personalmente mi ritengo molto soddisfatto dei risultati ottenuti dato che la creazione di un sistema di raccomandazione preciso è complicata per molti aspetti.

Prendiamo come esempio un modello di classificazione che si occupa di capire se in una immagine è presente un cane o un gatto. Questo modello viene allenato su delle immagini che hanno come etichette un valore preciso, definito e certo. Il modello di raccomandazione, al contrario, viene addestrato utilizzando le valutazioni degli utenti, le quali sono soggettive e mutevoli. Un singolo utente può manifestare opinioni completamente diverse sullo stesso film, a seconda del momento in cui esprime la valutazione (la valutazione subito dopo la visione potrebbe cambiare da quella espressa un giorno o un mese dopo). Inoltre, l'opinione può variare a seconda dell'umore e delle emozioni del momento, o se si tratta della prima o della seconda volta che guarda il film. Di conseguenza, il voto non rappresenta in modo univoco l'opinione di un individuo su un particolare film, a differenza di un'immagine nella quale è chiaramente identificabile se in essa è presente un cane o un gatto. Inoltre, è importante considerare il metro di valutazione utilizzato nelle recensioni.

Alcune persone potrebbero giudicare un film negativamente se ottiene una valutazione di 3 stelle su 5, mentre altre potrebbero considerarlo negativo solo con valutazioni di 1 o 2 stelle. I sistemi di raccomandazione non posso considerare tutti questi aspetti, cercano di predire le recensioni mediante algoritmi e calcoli matematici che seppur avanzati, presentano limitazioni. Persino il sistema più preciso non potrà mai essere esente da errori perché non potrà tenere in considerazione tutti i fattori elencati precedentemente. Anche il numero di recensioni riveste un ruolo fondamentale. Servono moltissimi dati per creare un modello preciso e quello creato utilizza un database di circa 100 mila recensioni. Sebbene sia un numero significativo, non è sufficiente per creare un sistema basato sul collaborative filtering davvero preciso. Un ulteriore aspetto di cruciale importanza che incide sulla qualità dei risultati è che il sistema di raccomandazione creato non sfrutta informazioni aggiuntive sui film come l'anno di pubblicazione, il genere, il regista o gli attori. Se il sistema utilizzasse tali informazioni insieme alle recensioni, potrebbe essere sviluppato un modello più sofisticato in grado di capire delle relazioni nascoste tra gli utenti e, di conseguenza, diventare più specifico ed efficiente nella sua funzione di raccomandazione.

L'errore assoluto medio di circa 0.65 e quello quadratico medio di 0.74 mettono in luce che il sistema è però capace di determinare se un film potrebbe piacere o meno ad un utente. L'obiettivo principale infatti non era quello di ottenere delle predizioni estremamente precise ma bensì di creare un modello che fosse in grado di consigliare 10 film ad un utilizzatore. Anche nel caso in cui una predizione di un film raccomandato avesse un errore assoluto superiore della media (ad esempio la predizione è 5 mentre la valutazione dell'utente è 4 e quindi l'errore assoluto è pari a 1) il film consigliato sarebbe comunque un film gradito.

È per questo e per i motivi elencati precedentemente che il modello creato mi soddisfa pienamente.

Bibliografia

[1] *Build apps for any screen.* URL: https://developer.android.com/studio/intro. (accessed: 7.08.2023).

[2] *Che cos'è il Deep Learning?* URL: https://www.oracle.com/it/artificial-intelligence/machine-learning/what-is-deep-learning/#:~:text=il%20Deep%20Learning%3F-,Definizione%20di%20Deep%20Learning,da%20grandi%20quantit%C3%A0%20di%20dati.. (accessed: 27.07.2023).

[3] *Che cos'è l'overfitting?* URL: https://aws.amazon.com/it/what-is/overfitting/. (accessed: 25.07.2023).

[4] *Collaborative Filtering Recommendation System Using TensorFlow with Neural Net.* URL: https://python.plainenglish.io/collaborative-filtering-recommendation-system-using-tensorflow-with-neural-net-7f8dba4521da. (accessed: 16.07.2023).

[5] *Dart overview.* URL: https://dart.dev/overview. (accessed: 7.08.2023).

[6] *Dart: a language for structured web programming.* URL: http://googlecode.blogspot.com/2011/10/dart-language-for-structured-web.html. (accessed: 7.08.2023).

[7] *filter bubble.* URL: https://www.treccani.it/vocabolario/filter-bubble_res-b92bdbdc-89c2-11e8-a7cb-00271042e8d9_%28Neologismi%29/. (accessed: 30.07.2023).

[8] Aurelien Geron. *Hands On Machine Learning with Scikit Learn Keras and Tensorflow.* O'Reilly, 2019. ISBN: 9781492032649.

[9] Aurelien Geron. «Hands On Machine Learning with Scikit Learn Keras and Tensorflow». In: O'Reilly, 2019. Cap. 10, p. 277.

[10] *Google Colab per il Machine Learning: cos'è e come si usa.* URL: https://www.html.it/articoli/google-colab-per-il-machine-learning-cose-e-come-si-usa/. (accessed: 6.08.2023).

[11] *Hot reload.* URL: https://docs.flutter.dev/tools/hot-reload. (accessed: 14.08.2023).

[12] *How to Build a Movie Recommendation System.* URL: https://medium.com/towards-data-science/how-to-build-a-movie-recommendation-system-67e321339109. (accessed: 30.07.2023).

[13] *Keras: l'API di alto livello per TensorFlow.* URL: https://www.tensorflow.org/guide/keras?hl=it. (accessed: 5.08.2023).

[14] *legge di Moore.* URL: treccani.it/enciclopedia/legge-di-moore_%28Enciclopedia-della-Scienza-e-della-Tecnica%29/. (accessed: 27.07.2023).

[15] *Matplotlib: Visualization with Python.* URL: https://matplotlib.org. (accessed: 16.07.2023).

[16] *Meet Android Studio.* URL: https://flutter.dev/. (accessed: 7.08.2023).

[17] *MovieLens.* URL: https://grouplens.org/datasets/movielens/. (accessed: 14.08.2023).

[18] *Movielens.* URL: https://movielens.org/. (accessed: 14.08.2023).

[19] *Post-training quantization.* URL: https://www.tensorflow.org/lite/performance/post_training_quantization. (accessed: 5.08.2023).

[20] *Recommender system.* URL: https://en.wikipedia.org/wiki/Recommender_system. (accessed: 30.07.2023).

[21] A. L. Samuel. «Some Studies in Machine Learning Using the Game of Checkers». In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. DOI: 10.1147/rd.33.0210.

[22] *sklearn.model_selection.train_test_split.* URL: https://scikit-learn.org/stable/modules/generated/sklearn.model%5C_selection.train%5C_test%5C_split.html. (accessed: 24.07.2023).

[23] *sklearn.preprocessing.LabelEncoder*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. (accessed: 24.07.2023).

[24] *TensorFlow Lite*. URL: https://www.tensorflow.org/lite/guide?hl=it. (accessed: 5.08.2023).

[25] *tflite_flutter 0.10.1*. URL: https://pub.dev/packages/tflite_flutter. (accessed: 10.07.2023).

[26] *when to use machine learning*. URL: https://docs.aws.amazon.com/it_it/machine-learning/latest/dg/when-to-use-machine-learning.html. (accessed: 27.07.2023).

Elenco delle figure