POLITECNICO DI TORINO

DIPARTIMENTO DI AUTOMATICA E INFORMATICA

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo di un sistema di raccomandazione basato su adaptive learning e tecnologie semantiche

Politecnico di Torino

Relatori
Prof. Antonio Vetrò
Dr. Giovanni Garifo

Candidato
Antonio Casto

Anno Accademico 2021-2022

Alla mia Famiglia,
Sostegno nella mia
Vita.

Sommario

Nel Web moderno le piattaforme di e-commerce e i servizi di contenuti in streaming sono caratterizzati da vasti cataloghi di contenuti e prodotti a disposizione degli utenti iscritti. Nasce da ciò la necessità, per queste piattaforme, di moduli software che consentano loro di proporre nuovi contenuti o prodotti massimizzando le interazioni con gli utenti: i Sistemi di Raccomandazione. Questi generano raccomandazioni di contenuti o prodotti per l'utente, migliorando l'esplorazione del catalogo proposto.

In questa tesi si presenta il lavoro di ricerca e sviluppo eseguito al fine di sviluppare un Sistema di Raccomandazione per il progetto (europeo) Erasmus+ *CLIKC - Content and Language Integrated learning for Key Competences*. Lo sviluppo del progetto è ancora *in corso* ed è iniziato a Giugno 2021 con scadenza fissata a Maggio 2023.

Si presenta inizialmente una panoramica sulla teoria esistente che sta alla base dei raccomandatori. Si espongono i concetti di utente, item e feature e le difficoltà che vengono spesso affrontate, come il problema del Cold Start. Segue poi il concetto di feedback, l'importanza dell'attività di Feature Engineering e i principali approcci su cui si basano i Sistemi di Raccomandazione, come quelli Basati sul Contenuto, sul Filtraggio Collaborativo e l'Approccio Ibrido. Si espongono inoltre i passaggi fondamentali per ottenere le raccomandazioni, tra cui l'addestramento del modello e la successiva valutazione attraverso alcuni metodi. Segue una panoramica su alcuni dei framework e delle librerie esistenti per sviluppare Sistemi di Raccomandazione.

Vengono poi esposte caratteristiche, motivazioni e obiettivi che stanno alla base del progetto CLIKC, ovvero la riduzione dei tempi di disoccupazione tramite lo sviluppo di una piattaforma Web innovativa per la formazione, a cui il Centro Nexa su Internet & Società - Politecnico di Torino contribuisce con lo sviluppo del Sistema di Raccomandazione, oggetto

di questa Tesi.

Si descrivono poi dataset utilizzati, struttura e funzionalità dei due applicativi prototipali di raccomandatori sviluppati, basati sui framework Collie e LightFM. Segue inoltre un'analisi sulle raccomandazioni ottenute tramite l'applicativo prototipale, basato su LightFM, utilizzando dataset caratterizzati da features più frequenti, tra gli item, rispetto ad altre.

Infine, in questa esposizione si illustrano i framework e librerie impiegate e lo schema dell'architettura software prodotta per il progetto, consistente di due microservizi sviluppati con il framework FastAPI e disposti in cascata, uno di interfaccia verso l'esterno ed uno che gestisce il modello del raccomandatore basato sul framework LightFM; a ciò si aggiunge una descrizione dei contenuti prodotti dai partner didattici di progetto, i quali saranno utilizzati per la costruzione del dataset del raccomandatore.

Ringraziamenti

In questa poche righe vorrei esprimere la mia gratitudine verso coloro che mi sono stati accanto nel mio percorso universitario.

Vorrei in particolare ringraziare la mia famiglia: mio padre Enzo, mia madre Paola e mio fratello Emanuele per aver creduto ogni singolo istante in me e per avermi sostenuto nei momenti più difficili di questo percorso.

Ringrazio Michela per la sua profonda capacità di ascolto, comprensione e sostegno nel mio ultimo anno univesitario.

Ringrazio i miei zii, Domenico ed Angela, per avermi sempre spinto ad osare e ad aspirare di più.

Vorrei ringraziare inoltre lo staff del Centro Nexa su Internet & Società - Politecnico di Torino per avermi dato l'opportunità di svolgere questa Tesi; un ringraziamento particolare va ai miei relatori, il Prof. Antonio Vetrò e il Dr. Giovanni Garifo, i quali mi hanno supportato nella stesura e nel progetto a cui questa tesi fa riferimento.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

I Sistemi di Raccomandazione

1.1 Introduzione

In termini molto semplici, riprendendo anche le considerazioni del riferimento [2], un Sistema di Raccomandazione è un *modulo software* che, tramite un algoritmo apposito, permette di ricevere suggerimenti; dunque esso è in grado di aiutare l'utente nelle sue scelte relativamente a ciò che in linguaggio tecnico vengono definiti ***item***, ovvero tracce musicali, film, articoli, ecc...

Tali suggerimenti sugli item possono essere derivati, ad esempio, dalla storia passata delle *interazioni* dell' ***utente*** con item aventi simili caratteristiche (ad esempio per un film: genere, anno, regista, ecc..), in linguaggio tecnico dette ***features***, oppure dalle interazioni eseguite da utenti aventi features simili (genere, età, nazionalità, ecc..)[1].

Sempre riferendoci a [2], a partire dagli anni '70 alla Duke University si è deciso di incentivare la ricerca in tale campo; successivamente si è arrivati all'effettivo sviluppo del primo sistema di raccomandazione, chiamato Tapestry, presso gli uffici dello Xerox Palo Alto Research Center.

Al giorno d'oggi l'impiego dei Sistemi di Raccomandazione è estremamente frequente tra le piattaforme web, basti pensare a Facebook, Amazon, Netflix, Google, YouTube; essi hanno progressivamente assunto un

[1] Le features saranno meglio descritte nella sezione 1.3.2

ruolo sempre più centrale, in ottica business, poichè capaci di aumentare significamente i guadagni delle aziende. A ciò si aggiunge la possibilità di portare all'utente finale un esperienza d'uso migliore. Per rimarcare maggiormente l'importanza dei sistemi di raccomandazione si osservi che nel 2006 Netflix ha indetto una competizione da 1,000,000$, chiamata "Netflix prize", la cui finalità era la realizzazione di un sistema di raccomandazione migliore di quello che era attualmente utilizzato dalla società statunitense.

1.2 Principi di Funzionamento

Come osservato nel riferimento [3], le raccomandazioni che gli utenti possono ricevere possono essere di due tipi: *personalizzate* o *non personalizzate*. Le raccomandazioni non personalizzate forniscono risultati diversi rispetto alle raccomandazioni personalizzate. Più in generale si tratta di raccomandazioni generiche basate sulla popolarità degli item o sulle valutazioni globali espresse dagli utenti della piattaforma.

Chiaramente una raccomandazione personalizzata porta un maggiore beneficio all'utente finale in quanto gli item che vengono a lui proposti sono quelli che più ne rispecchiano gli interessi.

Secondo il riferimento [3], un sistema di raccomandazione deve risolvere varie problematiche quali il *Cold Start*, la *scarsità dei dati*, la *scalabilità* e la *diversità*:

1. **Cold Start**: Rappresenta il caso iniziale in cui non sono ancora disponibili abbastanza metadati[2] tali da consentire ad un sistema di raccomandazione di fornire suggerimenti utili. Questa mancanza di dati può essere dovuta, ad esempio, al fatto che si voglia fornire raccomandazioni per un nuovo utente che si è appena iscritto ad una piattaforma, e di conseguenza esso non ha ancora ancora fornito informazioni su eventuali interazioni utili alla generazione di raccomandazioni. [3]
2. **Scarsità dei Dati**: Tale problema consiste nella mancanza di dati, ovvero interazioni, tali da completare quella che viene definita ***matrice delle interazioni utente-item***. In tale matrice bidimensionale ogni riga rappresenta un utente e ogni colonna rappresenta un item;

[2]Il concetto verrà approfondito nella sezione 1.3.1

l'intersezione data da una riga i-esima e da una colonna j-esima rappresenta l'interazione[3] avvenuta tra l'utente i-esimo e l'item j-esimo. Sfortunatamente gli utenti spesso non hanno modo di interagire con tutti gli item di una piattaforma ma limitano il loro spazio di azione all' 1% del catologo disponibile. Conseguenza di ciò è una matrice delle interazioni tipicamente *sparsa* con il 99% delle interazioni mancanti. Considerando anche che i sistemi di raccomandazione fanno spesso leva sulla somiglianza tra utenti, sulla base delle interazioni utente-item registrate, ciò che ne deriva è che utenti che non hanno interagito con alcun item ottengono raccomandazioni che spesso non rispecchiano le loro necessità. [3]

3. **Scalabilità**: La *scalabilità* è uno dei tipici problemi che vengono affrontati per qualunque applicativo che debba funzionare in un contesto in cui sono presenti migliaia, se non milioni, di utenti utilizzatori. In tali contesti le prestazioni devono essere ottimizzate. Anche per un sistema di raccomandazione non ci si può esimere dall'affrontare tale problema. Basti pensare ad Amazon, Netflix o Spotify: i sistemi di raccomandazione di tali piattaforme devono essere in grado di fornire suggerimenti in tempo reale.

   Da ciò diventano di fondamentale importanza i concetti di *throughput*[4] e di *latenza*[5]. [3]

4. **Diversità (e Novità)**: La *diversità* è un concetto importante. Con tale concetto vogliamo esprimere la necessità per cui un sistema di raccomandazione non fornisca suggerimenti tali da indurre un utente ad interagire con altri item che siano eccessivamente *simili* a quelli con cui ha interagito in passato, riducendo anche il possibile impatto dovuto all'eccessiva popolarità. Un esempio potrebbe essere, in ambito cinematrografico, la raccomandazione di tutti i film di un'intera trilogia data la visione, da parte dell'utente, di uno solo dei film appartenenti ad essa. Tutto ciò induce l'utente a navigare in uno spazio eccessivamente limitato del catalogo disponibile. A tale contesto si lega il concetto di *serendipità* che esprime la capacità di un

[3] Valutazione numerica o il caso binario in cui un'interazione sia avvenuta o meno.

[4] Inferenze al secondo.

[5] Tempo per un'inferenza.

sistema di raccomandazione di non limitare lo spazio esplorabile di un catalogo. [1, 3]

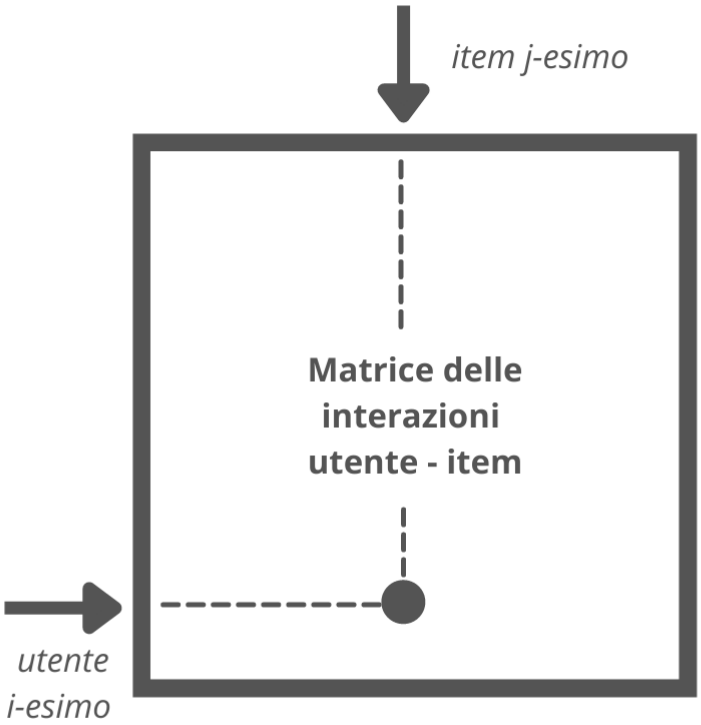

Figura 1.1: Esempio della matrice delle interazioni utente-item. Immagine realizzata come riadattamento di un'immagine tratta dal riferimento [1].

1.3 Componenti Principali

In questa sezione vengono presentati i componenti ed i concetti essenziali che compongono un Sistema di Raccomandazione, in accordo con i riferimenti [1] e [3].

1.3.1 Feedback: Implicito vs. Esplicito

Al fine di fornire raccomandazioni personalizzate rilevanti agli utenti, il compito principale dei sistemi di raccomandazione consiste nel tenere traccia dei dati rilevanti in merito agli utenti, gli item e le loro possibili interazioni. Tali dati vengono principalmente conservati in un *Content Management System (CMS)*, per quanto riguarda le informazioni relative agli item e agli utenti, oppure, riguardo le interazioni, vengono generati a partire da un sistema di *web analytics* (ad esempio Google Analytics) [3].

Come sostenuto nei riferimenti [1,3], al giorno d'oggi i sistemi di raccomandazione adottano 3 tipi di modelli strategici[6]:

1. **Content Based** *(Basato sul Contenuto).*
2. **Collaborative Filtering** *(Filtraggio Collaborativo).*
3. **Hybrid Approach** *(Approccio Ibrido).*

Il concentto di ***feedback*** è centrale nel contesto del *Filtraggio Collaborativo*: in accordo al riferimento [4] tale approccio è basato sul comportamento dell'utente, osservabile nella storia passata delle sue interazioni con la piattaforma; tali informazioni costituiscono i metadati raccolti. Di conseguenza le *features* demografiche relative all'utente *affiancano* quelle relative ai feedback. Il feedback verso un item viene definito ***esplicito*** se fornito appunto *esplicitamente* dall'utente, ad esempio nella forma di una valutazione numerica di un item: si pensi alle valutazioni assegnate ad un film su una scala che va da 1 a 5 stelle. Diversamente, il feedback ***implicito*** verso un item viene inteso come l'insieme delle azioni compiute dall'utente per interagire con gli item di una piattaforma; si pensi ad esempio all'atto di aggiungere un prodotto al carrello per poi effettuarne l'acquisto: l'insieme degli acquisti costituisce una fonte di feedback implicito. [4]

È bene ora rimarcare alcune differenze semantiche e tecniche tra feedback *implcito* ed *esplicito*. Al contrario del feedback esplicito, dove tramite una valutazione numerica è possibile esprimere un feedback negativo, con il feedback implcito è complicato rilevare un *feedback negativo*: ad esempio, un utente che non ha interagito con un item potrebbe non averlo

[6] Verranno approfonditi nella sezione 1.3.3.

fatto poichè non lo ha gradito oppure perchè non è stato in grado di interagirvi. Tuttavia, a differenza del feedback esplicito, il feedback implicito da importanza ai dati sulle interazioni mancanti in quanto questi possono contenere informazioni su eventuali feedback negativi. Si noti, inoltre, che il feedback implcito è per natura affetto da *rumore*; ciò è dovuto al fatto che un'interazione potrebbe essere stata compiuta dall'utente per motivi non strettamente connessi ai suoi gusti personali: ad esempio, l'acquisto di un prodotto su Amazon potrebbe essere un regalo destinato a qualcun altro. Un'altra caratteristica da notare è che, se l'interazione conservata come feedback implicito è espressa tramite valore numerico, il feedback esprime allora una certa *confidence (o fiducia)* sul fatto che l'utente gradisca un item o meno. Un valore numerico di confidence *grande* verso un item, ad esempio un evento che ricorre più volte, a differenza del feedback esplicito, non esprime in termini assoluti il grado di preferenza di un utente bensì una maggiore possibilità di gradimento da parte dell'utente. Infine, i sistemi di raccomandazione basati su feedback implicito devono fare uso di metodi di valutazione particolari, tenendo conto di diversi fattori: essi devono tener conto della disponibilità di un item, della sua contesa con altri (ad esempio, concomitanza nel caso di trasmissioni televisive) e di eventuale feedback ripetuto nel tempo. [4]

1.3.2 Feature Engineering

Nella sezione 1.1 abbiamo anticipato che item ed utenti sono contrassegnati da caratteristiche che li definiscono, denominate *features*.

In accordo con il riferimento [3], una delle attività principali che vengono svolte contestualmente allo sviluppo di un sistema di raccomandazione è quella di ***Feature Engineering***.

Tale attività, che è nota per la sua lunga durata, si svolge essenzialmente durante la fase di preparazione dei dati ed ha lo scopo di estrarre dai dati, precedentemente raccolti, le *features* che saranno più rilevanti ai fini delle raccomandazioni. L'estrazione è strettamente connessa al contesto applicativo in cui ci trova. Per quanto riguarda gli item, si vogliono estrarre delle features che siano in grado di descrivere l'item e che che dipendano fortemente dal contesto: nel caso di un servizio musicale, ad esempio, per ogni brano vogliamo delle features per *titolo, autore, genere, durata, ecc...* Per quanto riguarda gli utenti, invece, la fase di Feature Engineering si

può focalizzare anche sull'estrazione di features inerenti ai dati demografici degli utenti come, ad esempio, genere, età, nazionalità. A questi si aggiungono i metadati relatvi al feedback che, come già accennato, può essere implicito o esplicito. [3]

1.3.3 I Modelli

Come anticipato in 1.3.1, secondo i riferimenti [1, 3], vengono impiegati tre tipologie di modelli su cui si basano i sistemi di raccomandazione:

1. **Content Based** *(Basati sul Contenuto).*
2. **Collaborative Filtering** *(Filtraggio Collaborativo).*
3. **Hybrid Approach** *(Approccio Ibrido).*

I sistemi di raccomandazione *Basati sul Contenuto* pongono maggiore enfasi sulle *features* fornite esplicitamente che descrivono utenti e item a supporto delle interazioni. Infatti le raccomandazioni sono fornite sulla base di un modello che, date le feature inerenti agli utenti e/o agli item e i risultati delle interazioni, è in grado di generare nuovi suggerimenti. A sua volta tale approccio può essere incentrato sugli item (un modello per item e uso di feature degli utenti) oppure sugli utenti (un modello per utente e uso di feature degli item); alternativamente è anche possibile un approccio che faccia utilizzo di entrambe le tipologie di feature esplicite. Tali modelli inoltre soffrono molto meno il problema del *Cold Start*, che invece affligge i modelli basati sul *Filtraggio Collaborativo*; infatti eventuali nuovi contenuti o utenti aggiunti sono descritti tramite le features che consenteno di fornire suggerimenti utili. Nel caso venga aggiunto un item con features *mai introdotte*, solo allora tale tipologia di modello potrebbe soffrire del Cold Start; tuttavia questo problema verrà superato con l'invecchiamento del sistema di raccomandazione. [1]

I modelli dei sistemi di raccomandazione basati sul *Filtraggio Collaborativo* si basano sulle interazioni precedentemente registrate tra utenti e item. Da tale *storico* vengono fornite nuove raccomandazioni riuscendo a definire eventuali similarità tra utenti (raccomandazioni basate sugli utenti) e tra item (raccomandazioni basate sugli item) e predicendo possibili interazioni che possano avvenire con una certa probabilità. Come anticipato precedentemente, lo storico delle interazioni è rappresentato dalla ***matrice delle interazioni utente-item***. [1]

A loro volta, secondo il riferimento [1], i sistemi di raccomandazione basati sul Filtraggio Collaborativo possono essere suddivisi in due sottocategorie che rappresentano due approcci differenti:

1. **Model based (basati su un Modello)**:
   L'approccio *Model based* presuppone l'esistenza di un ***modello*** in grado di interpretare ed elaborare la *matrice delle interazioni utente-item* così da predire nuove possibili interazioni utente-item che vengono poi proposte all'utente sotto forma di raccomandazioni. Esso sfrutta il concetto di ***Fattorizzazione di Matrice***, tramite cui è possibile scomporre la matrice delle interazioni utente-item, inizialmente *sparsa*, nel prodotto di due matrici più piccole e dense: una contiene una rappresentazione degli utenti ed una contiene quella degli item. Tutto ciò consente di operare in uno spazio dimensionale delle features ridotto, dove queste non sono più esplicite ma derivate dall'apprendimento. [1]
2. **Memory based (basati sulla Memoria)**:
   L'approccio *Memory based* utilizza esclusivamente le interazioni precedentemente salvate. Non viene utilizzato alcun *modello* e, per l'approccio *user-user*, viene sfruttata la ricerca tra i *k-nearest neighbours*, ovvero tra i *vicini più vicini*: ad esempio, per gli utenti più *vicini* ad un utente, in termini di interazioni, consigliare all'utente i film o i brani musicali più popolari. Per l'approccio *item-item* si considera invece una ricerca di similarità, in termini di interazioni da parte di tutti gli utenti, tra gli item più simili a quelli con cui un utente ha avuto un'interazione positiva; si scelgono poi i *k-nearest neighbours*. Lo svantaggio di tale approccio è che ha una bassa *scalabilità* se non gestito correttamente: basti pensare al caso di milioni di utenti e all'uso dell'algoritmo K-Nearest Neighbours (avente complessità O(uik)[7]). [1]

Riassumendo, i sistemi di raccomandazione basati sul *Filtraggio Collaborativo* necessitano delle interazioni utente-item. Ciò li rende estremamente versatili e sempre più *precisi* man mano che le interazioni utente-item aumentano. Nelle fasi inziali però si osserva il fenomeno già citato del *Cold Start*: non avendo a disposizione uno storico delle interazioni precedente, le raccomandazioni iniziali si possono basare su item e utenti

[7]Dove u è il numero di utenti, i il numero di item e k il numero di *vicini*.

scelti casualmente, sugli item più popolari al momento oppure sulla proposta di nuovi item a vari utenti e viceversa. Infine, almeno per le fasi iniziali, è bene osservare che è possibile utilizzare modelli di sistemi di raccomandazione ***non*** basati sul *Filtraggio Collaborativo*. [1]

Quest'ultima affermazione ci porta a parlare di modelli di sistemi di raccomandazione ad ***Approccio Ibrido***. In tale contesto, per ovviare al problema del *Cold Start*, si possono combinare i due approcci, ovvero quello basato sul *Filtraggio Collaborativo* e quello *Basato sul Contenuto*; il secondo metodo, sfruttando le *features* degli item e degli utenti definite in fase di inserimento, potrebbe essere utilizzato nelle fase iniziali e, successivamente, con l'aumentare dei dati a disposizione sulle interazioni, vi si potrebbe affiancare il *Filtraggio Collaborativo*. Tutto ciò permette di sfruttare i punti di forza di entrambi gli approcci. [1, 3]

1.4 Le Implementazioni

In questa sezione si vuole presentare una *panoramica* su alcuni dei *framework* tramite cui è possibile sviluppare un sistema di raccomandazione. Con un *framework*, inteso come *impalcatura software*, è possibile utilizzare le funzionalità messe a disposizione, evitando allo sviluppatore di riscriverne il relativo codice.

Alcuni dei framework per sistemi di raccomandazione che andremo ad analizzare sono, a loro volta, basati sul framework open source di *machine learning* ***PyTorch***; dunque vogliamo presentare un'introduzione a tale framework nella sottosezione 1.4.1.

Successivamente, nella sottosezione 1.4.2, introdurremo *criteri* e *metodologie* tramite cui è possibile valutare un sistema di raccomandazione.

Infine, nella sottosezione 1.4.3, si illustreranno i framework scoperti durante la fase di ricerca ed esplorazione svoltasi nell'ambito del progetto europeo *CLIKC*. Collateralmente verrà fornita una comparativa tra essi.

1.4.1 Introduzione a PyTorch

PyTorch[8] è un framework open source sviluppato da Facebook e scritto in linguaggio Python. Esso viene principalmente utilizzato per applicazioni

[8] https://pytorch.org

Figura 1.2: Schema dei modelli di Sistemi di Raccomandazione esistenti. Immagine realizzata come riadattamento di un'immagine tratta dal riferimento [1].

di ***Deep Learning***, il quale, come suggerito dal riferimento [5], costituisce un campo del machine learning che implementa tutta una serie di algoritmi che si ispirano al funzionamento ed alla struttura del cervello. Tali algoritmi vengono chiamati ***Reti Neurali Artificiali***.

PyTorch, osservandone la documentazione online presente al riferimento [6], fornisce tutta una serie di *Classi* e *Metodi di Classi* tramite cui è possibile *costruire* una rete neurale a più *strati*, da addestrare attraverso una fase detta *training*, al fine di ottenere un sistema di raccomandazione.

Come vedremo nella sezione 1.4.3, è possibile costruire un sistema di raccomandazione utilizzando dei framework sviluppati al di sopra di PyTorch, o altre librerie Python, evitando di riscrivere di volta in volta tutto il codice inerente ad una rete neurale. Sarà pertanto possibile dedicarsi, una volta scelto il modello per un certo framework, all'attività di ottimizzazione dei suoi *parametri* al fine di ottenere delle migliori raccomandazioni.

1.4.2 Valutazione di un Sistema di Raccomandazione

Prima di introdurre le metodologie di valutazione di un sistema di raccomandazione è bene sottolineare, per un framework qualunque[9], quali sono le fasi principali al fine di ottenere un sistema di raccomandazione *pronto all'uso*; a tal fine faremo riferimento al framework LightFM[10] (vedi riferimento [7]) ed al codice di esempio tratto dalla relativa documentazione [8]:

1. **Ottenimento dei dati:**
   Solitamente è la prima attività che viene svolta. Preso un framework di riferimento, si vuole effettuare il reperimento dei dati che verranno forniti in pasto al *modello* scelto. Alcuni framework forniscono delle funzioni che si occupano di fare ciò. Ad esempio per LightFM e il Dataset Movielens[11]:

```
1  import numpy as np
2  from lightfm.datasets import fetch_movielens
3
4  movielens = fetch_movielens()
5
6  train = movielens['train']
7  test = movielens['test']
```

   I dati ottenuti vengono poi suddivisi in un sottoinsieme di *train*, che servirà per addestrare il modello, ed in un sottoinsieme di *test* che

[9] Ciascun framework presenta dei passaggi simili.

[10] https://github.com/lyst/lightfm

[11] https://grouplens.org/datasets/movielens/100k/

servirà per mettere alla prova l'*accuratezza* del sistema appena addestrato.

2. **Addestramento del modello:**
   Si passa, in seconda fase, all'*addestramento* del modello. In tale contesto va scelto il modello di riferimento e vengono stabiliti i parametri per l'addestramento (o *training* in lingua inglese). Tra i parametri più rilevanti troviamo il parametro *epochs* [9], che definisce il *numero di volte* per cui ciascun campione di dato di un dataset può essere utilizzato, durante l'addestramento, al fine di *aggiornare* i parametri interni del modello. Successivamente abbiamo la *funzione di costo (loss function in inglese)* [10], che è indicatore dell'errore che cerchiamo di *minimizzare*, ed infine il *tasso di apprendimento (learning rate)* [11], ovvero una *risposta* all'errore stimato su quanto i parametri del modello debbano cambiare al fine di ottenere una soluzione che converga.

```
1    from lightfm import LightFM
2    from lightfm.evaluation import precision_at_k
3    from lightfm.evaluation import auc_score
4
5    model = LightFM(learning_rate=0.05, loss='bpr')
6    model.fit(train, epochs=10)
```

3. **Valutazione dell'accuratezza del modello:**
   Si passa poi ad una fase di valutazione sull'accuratezza del modello addestrato. Qui vengono definite delle *metriche* e delle *metodologie* che ci permettono di effettuare dei confronti tra modelli, framework e configurazioni di parametri di modello differenti. Nel caso di LightFM, se utilizziamo il metodo ***P@k (precision at k)***, l'ottenimento dei punteggi di valutazione si traduce nel seguente codice:

```
1    train_precision = precision_at_k(model, train, k=10).mean()
2    test_precision = precision_at_k(model, test, k=10).mean()
3
4    train_auc = auc_score(model, train).mean()
5    test_auc = auc_score(model, test).mean()
6
7    print('Precision: train %.2f, test %.2f.' % (train_precision,
        test_precision))
8    print('AUC: train %.2f, test %.2f.' % (train_auc, test_auc))
```

e fornisce il seguente risultato:

```
1    Precision: train 0.59, test 0.10.
2    AUC: train 0.90, test 0.86.
```

4. **Predizione (generazione delle raccomandazioni):**
   In tale fase, tramite il metodo opportuno del modello scelto, *predict* nel caso di LightFM, si ottengno le raccomandazioni da fornire all'utente finale. Tali raccomandazioni sono fornite con un punteggio che esprime il livello di gradimento di un item.

Possiamo ora approfondire le *metriche* e le *metodologie* per la valutazione di un sistema di raccomandazione. Per la trattazione faremo riferimento a [12].

Per la valutazione dell'accuratezza ci può riferire sostanzialmente alle seguenti metodologie:

1. **Metodi basati sull'Accuratezza e sull'Errore**.
   Secondo il riferimento [12], questi a sua volta si suddividono in:

   (a) **Errore Assoluto Medio**:
   Considerato un item, è la media calcolata sulle differenze tra i punteggi sulle predizioni che il raccomandatore ha fornito per ogni utente e i punteggi reali forniti, in termini di valutazioni, da ogni utente. Di questo se ne prende il valore assoluto. Più il valore è vicino allo zero, più accurata sarà la predizione ottenuta.
   Di seguito si riporta la formula[12] dell'Errore Assoluto Medio:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n} \tag{1.1}$$

   (b) **Errore Quadratico Medio**:
   Si differenzia dal metodo precedente per via del fatto che l'errore medio non viene più calcolato con il valore assoluto bensì facendone il quadrato. Quindi, se il valore ottenuto è molto vicino allo zero, vorrà dire che il raccomandatore ha fornito un'ottima predizione.

[12] https://en.wikipedia.org/w/index.php?title=Mean_absolute_error&oldid=1053388699

(c) **Radice Quadrata dell'Errore Quadratico Medio**:
Il problema dell'Errore Quadratico Medio è che tende ad amplificare gli errori ponendoli su una scala differente rispetto a quella dei punteggi reali sulle valutazioni ottenute.
L'utilizzo della *Radice Quadrata dell'Errore Quadratico Medio* risulta utile per andare a risolvere tale problema; con questa operazione si va infatti a *normalizzare* il risultato ottenuto con l'Errore Quadratico Medio.

2. **Metodi di Supporto Decisionale**.
Tali metodi, secondo il riferimento [12], forniscono un *indicatore* in merito all'utilità di un raccomandatore nel consentire agli utenti di prendere delle decisioni ottimali sugli item che risultano essere più pertinenti:

(a) **Precisione**:
Consiste nella percentuale relativa a quanti item *opportuni* vengono selezionati rispetto al totale di tutti quelli raccomandati. Se ad esempio si vogliono raccomandare 7 item e di questi abbiamo solo un totale di 5 item veramente *opportuni*, allora la *precisione* sarà circa del 71%. È utile per rendersi conto di quanto le raccomandazioni sia accurate.

(b) **Richiamo**:
Consiste nella percentuale relativa a quanti item *opportuni* vengono selezionati rispetto al totale degli item *opportuni* disponibili. Se ci sono 7 item opportuni in totale e il raccomandatore ne consiglia solamente 2 allora il *richiamo* sarà circa del 29%. È utile per rendersi conto di quanti item utili si sta tralasciando.

(c) **Curva ROC**:
Il nostro sistema di raccomandazione è in grado di fornire in output un certo numero N di raccomandazioni. Tra i risultati ottenuti possiamo osservare una certa percentuale di item correttamente raccomandati (e rilevanti) ed una certa percentuale di essi non correttamente raccomandati (e dunque non rilevanti). La scelta di N influisce direttamente sulle quantità precedentemente citate.
Tramite la curva ROC e la massimizzazione dell'area sottostante a questa è possibile selezionare una *soglia* ben precisa per N tale da massimizzare il numero di item significativi tra le raccomandazioni ottenute in output.

3. **Metodi Basati sulla Classifica**.
   Secondo il riferimento [12], i metodi precedentemnte analizzati non forniscono alcuna indicazione sull'ordinamento, magari in termini rilevanza, con cui dovrebbero essere suggeriti gli item.

   Il riferimento [12] elenca altri metodi che vanno ad affrontare proprio tale problematica:

   (a) **nDCG (Normalized Discounted Cumulative Gain)**:
   Ad ogni item suggerito si vuole assegnare un punteggio che ne indica l'importanza o rilevanza. Definiamo con il **Cumulative Gain (CG) o Guadagno Cumulativo** (vedi formula 1.2) la somma di tutti i punteggi di rilevanza, assegnati precedentemente, di tutti gli item forniti nell'insieme dei suggerimenti dal raccomandatore. Il *CG* non tiene però conto dell'importanza dell'ordine con cui vengono suggeriti tali item; con il **Discounted Cumulative Gain (DCG)** (vedi formula 1.3) si vuole risolvere tale problema, dividendo ogni singolo punteggio per un valore, ottenuto attraverso una funzione logaritmica su base 2, che cresce man mano che si considera l'i-esima posizione dell'item a seguire. Un raccomandatore si potrà così ritenere non ottimale se andrà a posizionare item abbastanza rilevanti tra gli utilimi posti.
   Di seguito le formule[13] 1.2 e 1.3:

$$CG_p = \sum_{i=1}^{p} rel_i \tag{1.2}$$

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} \tag{1.3}$$

   Al *DCG* si vuole applicare adesso una normalizzione tramite un valore opportuno: si ottiene allora l'**nDCG** (vedi formula 1.4). Tale normalizzazione consiste nel dividere il DCG calcolato all'i-esimo item per l'*IDCG* calcolato all'i-esimo item. IDCG sta per **Ideal Discounted Cumulative Gain (DCG ideale)** e si ottiene ordinando *preventivamente* gli item di un set di suggerimenti, in

[13] https://en.wikipedia.org/w/index.php?title=Discounted_cumulative_gain&oldid=1069780045

ordine discendente di rilevanza, e calcolandone il DCG fino all'item i-esimo.
Di seguito la formula[13] 1.4:

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{1.4}$$

(b) **Mean Reciprocal Rank**:
Consiste nel localizzare dove si trova il primo item di rilevanza tra gli item proposti nell'insieme di raccomandazioni. Il punteggio MRR per un dato insieme di raccomandazioni avente il primo item rilevante tra le prime posizioni sarà maggiore del punteggio ottenuto per un insieme di raccomandazioni avente il primo item rilevante tra le ultime posizioni. L'MRR per un insieme di raccomandazione si può ottenere sommando gli inversi dei numeri degli indici delle posizioni, a partire da 1, degli item considerati rilevanti ai fini delle raccomandazioni.

(c) **Precisione Media**:
Per una lista di 5 item raccomandati, si considerano solo gli item rilevanti. Per ognuno di essi, si calcola la *precisione* come rapporto tra il numero di raccomandazioni rilevanti fino all'item i-esimo considerato e il numero di raccomandazioni totali fino all'item i-esimo considerato. Con tali valori ne viene fatta la media. Tale metodo tiene quindi conto anche del posizionamento degli item nella lista delle raccomandazioni fornite.

(d) **Valutazione tramite Correlazione del Rango di Spearman**:
Tramite la correlazione del rango di Spearman è possibile calcolare la differenza, per ogni item, tra come il modello del raccomandatore ha classificato gli item ed una classificazione ritenuta ideale. Ciò diviene possibile mettendo a confronto le precedenti classifiche tramite una numerazione esplicita delle posizioni degli item. Successivamente, per ogni differenza D che viene ottenuta e per N item, si calcola il coefficiente, inteso come punteggio finale, tramite l'equazione[14] 1.5.

[14] https://it.wikipedia.org/w/index.php?title=Coefficiente_di_correlazione_per_ranghi_di_
122301008

$$\rho_s = 1 - \frac{6 \sum_i D_i^2}{N(N^2 - 1)} \tag{1.5}$$

I valori ottenuti variano tra 1 e -1; se risultano essere negativi devono intendersi come una classifica nel verso opposto.

4. **Altri Metodi**:
   Vogliamo infine analizzare altri metodi presentati dal riferimento [12] in grado di fornire ulteriori tipologie di valutazione.

   (a) **Copertura**:
   Indica il numero di item raccomandati sul totale di quelli disponibili nel catalogo. Se vengono raccomandati 20 item su 100 dispobili la *copertura* sarà del 20%.

   (b) **Popolarità**:
   Gli item che ricorrono maggiormente tra le preferenze degli utenti vengono definiti *popolari*; è corretto supporre che essi vengano suggeriti con maggiore frequenza rispetto ad item meno popolari. Questa scelta è comunque in mano allo sviluppatore. Con tale metrica è possibile intuire quanto il nostro raccomandatore stia suggerendo item popolari.

   (c) **Novità**:
   In alcuni contesti è necessario che un sistema di raccomandazione fornisca anche delle novità in termini di raccomandazione così da portare un contributo maggiore, in termini di scoperta, all' utente finale. Con tale metrica è possibile definire quanto un sistema di raccomandazione stia effettivamente proponendo nuovi item all'utente.

   (d) **Diversità**:
   Tale metrica è utile allo sviluppatore per definire quanto *diversi* debbano essere i contenuti proposti, di volta in volta, dal sistema di raccomandazione.

   (e) **Valutazione Temporale**:
   È un dato di fatto che con il passare del tempo le valutazioni degli utenti nei confronti degli item possano subire una variazione. Ciò può essere dovuto a diversi motivi, come ad esempio un cambiamento di gusti o la semplice maturazione di un utente. Può pertanto essere utile considerare delle componenti temporali per

quanto riguarda le valutazioni. Ad esempio, le valutazioni assegnate di recente da un gruppo di utenti potrebbero avere un impatto maggiore sulla valutazione complessiva di un item.

1.4.3 Panoramica e Confronto sulle Implementazioni Esistenti

In questa sottosezione si vuole presentare lo stato dell'arte, al giorno d'oggi, di alcuni dei framework più popolari basati su Python e/o PyTorch[15] tramite cui è possibile sviluppare e confrontare sistemi di raccomandazione.

I criteri utilizzati per valutare i framework presentati consistono sostanzialmente nella loro *estendibilità*, nelle *tipologie di modelli* implementabili, nelle *casistiche di utilizzo* e nelle *API* fornite a corredo.

1. **LightFM**:
   LightFM[16] [7] consiste in una libreria Python, sviluppato da Lyst, che sostanzialmente mira allo sviluppo di sistemi di raccomandazione in grado di risolvere il problema del *Cold Start*. Come osservato nel riferimento [7], i modelli di raccomandatori *Basati su Contenuto* non sono adatti, dal punto di vista prestazionale, a risolvere tale problema poichè, non beneficiando di metadati sulle interazioni, richiedono una grande quantità di metadati per gli utenti e per gli item. Inoltre tale tipologia di modello rende complicata la correlazione tra i modelli stimati per ogni utente.

   Per risolvere il problema del *Cold Start*, i ricercatori di Lyst hanno proposto un modello ibrido Basato sul Contenuto e sul Filtraggio Collaborativo. In particolare si osserva che LightFM offre raccomandazioni rilevanti già dalle prime fasi del Cold Start, quando i dati sulle interazioni sono scarsi, e nelle fasi successive in cui questi diventano abbondanti. Infine, utenti ed item sono rappresentati come *vettori latenti (embeddings)*, ovvero dati che si trovano in uno spazio in cui dati simili sono collocati molto vicini; tali vettori latenti sono in grado di inglobare delle informazioni semantiche riguardo le *features* e possono

[15] Alcuni di essi si presentano come librerie utili nei processi di sviluppo

[16] https://github.com/lyst/lightfm

essere sfruttati per generare raccomandazioni più accurate: si pensi ad esempio alle raccomandazioni basate sui tag.

2. **Collie**:
   Collie[17], secondo la documentazione del riferimento [13], è una libreria Python molto utile per lo sviluppo di sistemi di raccomanazione *ibridi*; essa è stata sviluppata da ShopRunner. Tale libreria è particolarmente indicata nel caso si disponga di metadati su interazioni utente-item registrate come *feedback impliciti*; è in grado di gestire anche situazioni in cui siano presenti metadati registrati come *feedback espliciti*. La peculiarità di Collie consiste nel fornire allo sviluppatore delle API facili da utilizzare per il trattamento del dataset, come ad esempio la suddivisione tra i dati di *train* e i dati di *test*. Collie infine, grazie alla sua flessibilità, si presta per una veloce prototipazione e per eventuali sperimentazioni.
3. **Beta-RecSys**:
   Beta-RecSys[18], secondo il riferimento [14] e secondo la relativa documentazione presente al riferimento [15], è un framework open source basato su PyTorch che si può utilizzare per sviluppare, valutare e raffinare un sistema raccomandazione. Tramite degli strumenti standardadizzati che il framework fornisce è possibile manipolare i dati da dare in ingresso al raccomandatore da sviluppare al fine di costruire il dataset. Beta-RecSys è equipaggiato con tutta una serie di *modelli* pronti all'uso (9 con esattezza) corredati da una serie di strumenti utili all'apprendimento e alla validazione del raccomandatore sviluppato. Le peculiarità di Beta-RecSys consistono nella sua modularità, estensibilità e nella possibilità di implementarlo usando Docker[19].
4. **RecBole**:
   RecBole[20] [21], secondo l'abstract presente al riferimento [16], viene presentato come un *framework unificato*, utilizzabile a fini accademici, che offre una vasta libreria di modelli di raccomandazione. Con tale libreria è possibile implementare fino a 73 tipologie di modelli di

[17] https://github.com/ShopRunner/collie

[18] https://github.com/beta-team/beta-recsys

[19] https://www.docker.com

[20] https://github.com/RUCAIBox/RecBole

[21] https://recbole.io

raccomandazione utilizzando fino a 28 diversi tipi di dataset per valutarne le prestazioni, al fine di sviluppare nuovi modelli e algoritmi per generare raccomandazioni. RecBole viene anch'esso implementato in PyTorch ed offre elevate prestazioni di esecuzione in ambienti accelerati da GPU. Infine, il framework fornisce tutta una serie di strumenti come strutture dati estendibili e dataset da utilizzare con i modelli come benchmark.

Tabella 1.1: Tabella riassuntiva dei Sistemi di Raccomandazione trovati durante l'attività di ricerca.

	Estendibilità	**Modelli**	**Casi d'uso notevoli**	**API di rilievo**
LightFM	In termini di metodologie di apprendimento.	Unico modello di tipo Ibrido.	Adatto per scopi commerciali.	Per *fetch* dei Dataset noti e per trasformazione in matrici con formato COO.
Collie	Possibile creare modelli custom.	Modelli di tipo Ibrido e non Ibrido.	Adatto per prototipazione veloce e per sperimentazioni.	Helper functions per customizzare un model e per il trattamenteo del dataset.
Beta-RecSys	Possibile creare modelli custom.	Diversi modelli pronti all'uso.	Adatto per sviluppo end-to-end di raccomandatori con possibile uso di Docker.	Strumenti utili per la manipolazione dei dati e per la validazione di un raccomandatore.
RecBole	Strutture dati estensibili per l'uso con vari dataset.	Più di 70 modelli disponibili.	Ad esclusivo uso per fini accademici. Per implementazione e valutazione modelli di raccomandazione.	Strumenti per rendere più semplice l'utilizzo di tale framework.

Capitolo 2

CLIKC - Content and Language Integrated learning for Key Competences

2.1 Il Contesto del Progetto

Introduciamo adesso, con informazioni tratte dal riferimento [17], il contesto del progetto. L'Unione Europea negli ultimi anni ha attenzionato il problema della disoccupazione; si è stimato infatti che nel 2019 il *7,5%* dei potenziali lavoratori dell'area EU-19 fosse disoccupato, considerando invece il *6,7%* per l'area EU-27. Inoltre si è anche stimato che tra questi il *44%* si trovasse senza impiego per un periodo di oltre *12 mesi*.

È bene sottolineare che, secondo uno studio effettuato nel 2012 dal *FEIS (Fondo Europeo per gli Investimenti Strategici)*, si stima che il costo, nel corso di un anno, per ogni persona disoccupata è pari all'incirca alla cifra di *25.000 euro*; solo una percentuale pari al *25,2%* dei disoccupati beneficia di aiuti economici pubblici ma, nonostante ciò, la disoccupazione comporta un costo complessivo annuale pari a circa ***90 miliardi di euro***.

La *lunghezza* di tali periodi di disoccupazione è vista con grande preoccupazione per la *sostenibilità* del sistema dal punto di vista economico-sociale. Per tale ragione, diviene di fondamentale importanza un intervento efficace che punti a fornire tempestivamente un supporto ai disoccupati con *servizi di collocamento e carriera*.

Tra gli interventi a supporto dell'impiego più efficaci abbiamo la ***formazione***. Tramite questa è possibile potenziare le *skills (abilità)* di un disoccupato: esse costituiscono l'asset principale in grado di qualificare l'offerta di lavoro. L'impiego è *enormemente influenzato* da una particolare sottocategoria di skills dette ***skills trasversali***: la Raccomandazione del Consiglio UE 189/2018 definisce come competenze chiave le *skill personali, relazionali e di apprendimento*.

2.2 Gli Obiettivi del Progetto

Sempre secondo il riferimento [17], il progetto CLIKC si pone come obiettivo quello di sperimentare la *formazione* dei disoccupati attraverso l'introduzione di una modalità *innovativa*: l'obiettivo finale è quello di ridurre la durata del periodo di disoccupazione e di canalizzare le attenzioni verso le categorie più deboli.

Dunque, per raggiungere tale obiettivo si vuole puntare, tramite questo progetto, alla *creazione e test* di un ***catalogo di contenuti didattici*** da combinarsi con uno ***strumento di formazione a distanza***.

Il catalogo dei contenuti didattici finalizzato alla formazione sarà incentrato su *4 skills* ben precise che rientrano nel campo delle abilità base personali e relazionali.

Per concludere, il progetto CLIKC si focalizza in particolare su *4 innovazioni*:

- 2 innovazioni sono relative alla metodologia di insegnamento definita come **Content Language Integrated Learning and Micro-Learning** (Contenuti con Apprendimento Integrato della Lingua e Micro-Apprendimento).
- 2 innovazioni sono inerenti allo strumento ICT di intermediazione, ovvero alle modalità di fruizione dei contenuti del catalogo: ***l'Adaptive Learning e l'Advanced Analytics*** (Apprendimento Adattivo e Analisi Avanzate).

2.3 Panoramica sul Progetto

Il progetto CLIKC si configura come un progetto europeo finanziato nell'ambito dell'***Erasmus+ Programme*** - *Key Action 2 "Cooperation for innovation and the exchange of good practices". Call 2020 Round 1 - KA226 "Partnerships for Digital Education Readiness"*. Il progetto è ancora *in corso* e per esso è prevista una durata di 2 anni; tale periodo è iniziato a Giugno 2021 e si concluderà a Maggio 2023. Il progetto sarà sviluppato da diversi partner europei: **Veneto Lavoro** (organizzazione leader), Tecum Srl, Acción Laboral, bit Schulungscenter GmbH, ETI Malta (Executive Training Institute Ltd.), Centro Nexa su Internet & Società - Politecnico di Torino, Quantitas Srl, Association Europeenne pour la Formation professionnelle [17].

Il progetto, alla fine, produrrà ***una consegna***, ovvero una ***web app basata su adaptive learning***, la quale sarà in grado di fornire un catalogo di cotenuti formativi di breve durata inerenti alle competenze chiave già citate.

Tra le componenti software da sviluppare per il progetto assume rilevanza, ai fini della presente trattazione, il Sistema di Raccomandazione, che viene sviluppato appositamente e previa fase di ricerca e documentazione, tramite la mia collaborazione con il Centro Nexa su Internet & Società - Politecnico di Torino.

La consegna del progetto è strutturata nella produzione di 3 Intellectual Output (IO); dopo il rilascio dell' *IO1 (Intellectual Output 1)*, che consiste nella produzione del catalogo digitale di micro-learning da parte dei partner didattici, si svolgerà la *fase operativa* inerente allo sviluppo dell'*Intellectual Output 2* che comprende anche lo sviluppo del Sistema di Raccomandazione.

Infine avrà luogo la produzione dell'*Intellectual Output 3* inerente alla produzione delle Linee Guida per la diffusione e l'integrazione dello strumento prodotto all'interno delle politiche attive per l'occupazione.

Lo sviluppo del Sistema di Raccomandazione è anticipato da una fase di rircerca, già ampiamente descritta nel capitolo 1, e viene svolta da me al fine di individuare tecniche, metodologie e framework in grado di fornire tutti gli strumenti necessari allo sviluppo di un raccomandatore che possa fornire risultati ottimali all'utente finale. A questa fase succede l'effettiva

fase di sviluppo che sarà indirizzata, tramite il sistema di raccomandazione, a contribuire alla consegna finale dell'Intellectual Output 2. Infine, il raccomandatore sviluppato dovrà essere integrato nella piattaforma web, sviluppata da Quantitas Srl, e dovrà raggiungere l'obiettivo di fornire all'utente raccomandazioni efficaci sul prossimo contenuto del catalogo da consigliare: ciò verrà fatto in virtù di una prima fase di ***assessment***[1] dell'utente, appena iscritto alla piattaforma, e, successivamente, sulla base dei contenuti fruiti e dei risultati di apprendimento raggiunti.

[1]Questa fase permette di indirizzare l'utente verso i contenuti più opportuni per esso.

Capitolo 3

Sviluppo del Sistema di Raccomandazione

3.1 L'importanza del Dataset

Durante le fasi embrionali inerenti allo sviluppo del Sistema di Raccomandazione ci si è posti il problema della disponibilità di un ***dataset*** da utilizzare per le *fasi di apprendimento del raccomandatore*.

Poichè il Centro Nexa su Internet & Società - Politecnico di Torino ha ritenuto importante che le fasi di svilluppo di un *raccomandatore prototipale* iniziassero più o meno parallelamente alla produzione dell' *Intellectual Output 1*, al fine di ottimizzare i tempi e le successive fasi di sviluppo, si è dovuto compensare alla mancanza del dataset che sarebbe stato prodotto successivamente.

A tal fine sono state vagliate diverse proposte per sopperire a tale mancanza:

- Produzione di un **Dataset Sintetico**:
  Tale opzione prevede l'apposita creazione di un Dataset contenente degli utenti, degli item e delle relative features e interazioni. È possibile ottenere tutto ciò creando appositamente degli script in Python e/o utilizzando dei tool online quali **Mockaroo**[1]. Mockaroo permette di generare dati realistici di test fornendone l'output nei formati CSV, JSON, SQL e Excel.

[1] https://www.mockaroo.com

- Utilizzo di un **Dataset pre-esistente di un progetto esterno**.

Come vedremo nella sezione 3.2, inerente alla fase eplorativa dello sviluppo, sono stati utilizzati entrambi gli approcci proposti.

3.2 La Fase Esplorativa

La primissima *fase esplorativa* dello **sviluppo** del raccomandatore ha visto la messa in campo di due tra i *framework* analizzati durante la *fase di ricerca* (vedi sezione 1.4.3): ***Collie*** e ***LightFM***. Tale scelta è stata dettata dalla loro *versatilità* ed *adattabilità* rispetto al risultato finale che vogliamo ottenere.

3.2.1 Struttura del Prototipo

La struttura del ***prototipo*** iniziale è diversamente caratterizzata da quella che sarà prodotta nel corso dell'*Intellectual Output 2* (un'architettura a **microservizi**): ciò è dovuto al fatto che l'obiettivo di questa fase esplorativa è di stabilire il framework idoneo e definitivo da utilizzare per lo sviluppo. Per stabilire tale idoneità è stato necessario testare come i due framework trattano i dati in input, quali tipologie di raccomandazione forniscono in output e quali metriche forniscono, di default, per valutare i risultati delle raccomandazioni prodotte.

Al fine di eseguire tali valutazioni ho strutturato i due applicativi prodotti, uno per ciascun framework, in vari moduli Python al fine di ottenere una ***suite*** contenente le varie funzionalità che di norma un raccomandatore deve possedere. È bene notare che le due implementazioni presentano una struttura analoga.

Il modulo **main.py** fornisce un'interfaccia da riga di comando che consente allo sviluppatore di eseguire le varie funzionlità attraverso un menù stampato a schermo.

Viene fornito inoltre un modulo Python, di nome **recsys.py** (**collie_recsys.py** per l'applicativo basato su Collie), che si occupa di implementare le varie funzionalità del raccomandatore. Le funzionalità sono poi internamente implementate tramite delle *helper functions* contenute nel *package* **engines**. Questo, a sua volta, contiene vari *moduli* Python, ciascuno dei quali si occupa di fornire funzionalità diverse:

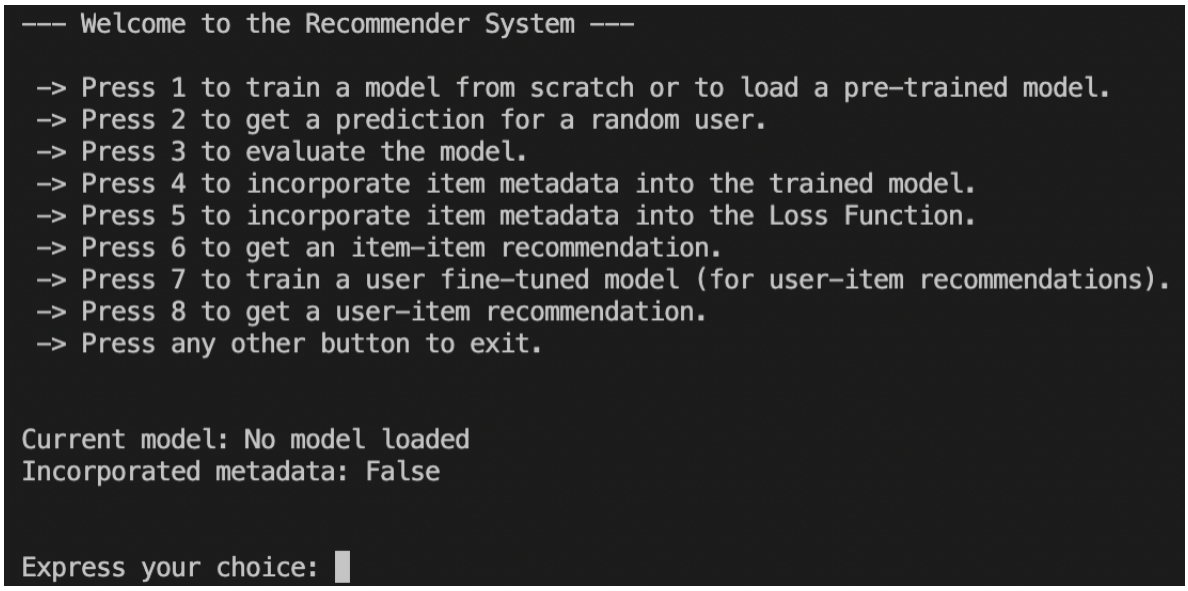

Figura 3.1: Menù principale del raccomandatore basato su Collie.

- **load_store_engine.py**: contiene delle funzioni per caricare e salvare dati da/su disco. Ciò viene fatto per salvare/caricare un modello addestrato, dati relativi al dataset o dati relativi a risulati di valutazioni effettuate.
- **training_engine.py**: contiene delle funzioni tramite cui è possibile addestrare il modello scelto per il raccomandatore.
- **prediction_engine.py**: contiene delle funzioni tramite cui è possible generare delle raccomandazioni per un dato utente. Le raccomandazioni fornite possono essere basate sul puro filtraggio collaborativo oppure sulle features relative agli item e/o agli utenti.
- **evaluation_engine.py**: il modulo implementa delle funzioni che permottono di generare delle valutazioni inerenti al modello addestrato sulla base delle metriche di giudizio che il framework utillizato offre.
- **metadata_engine.py** (solo per applicativo basato su Collie): parallelamente al modulo *training_engine.py*, offre delle funzioni per addestrare un modello integrando dei metadati relativi agli item nel modello stesso ed eventualmente nella *funzione di costo*.
- **mapping_engine.py** (solo per applicativo basato su LightFM): tale modulo viene utilizzato per ottenere delle funzioni che trattano il

mapping degli identificativi degli utenti, degli item e delle relative features. Tale modulo è essenziale in quanto nel framework LightFM, internamente al raccomandatore sviluppato, si ha una rappresentazione interna differente da quella esterna in termini di identificativi utilizzati.

Si vuole inoltre sottolineare che le costanti di progetto sono state raccolte nel modulo **constants.py** e che entrambi i progetti fanno uso di moduli Python che trattano l'importazione ed eventualmente la generazione dei *dataset* (maggiori informazioni verranno fornite nelle rispettive sottosezioni degli applicativi prodotti).

È importante infine evidenziare come entrambi gli applicativi debbano seguire gli stessi *passi* fondamentali nel medesimo ordine (figura 3.2):

1. **Costruzione o caricamento del dataset**.
2. **Addestramento del modello**.
3. **Generazione delle raccomandazioni (o delle valutazioni del modello addestrato)**.

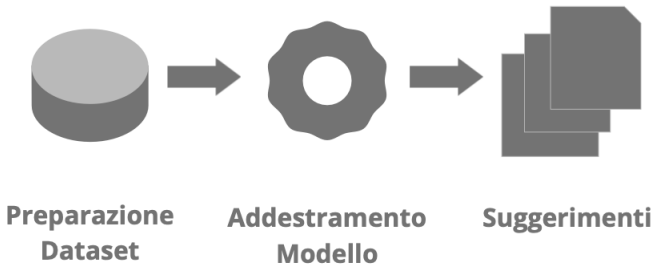

Figura 3.2: Passi fondamentali del prototipo del raccomandatore.

3.2.2 Utilizzo di Collie

La prima via percorsa durante la *fase esplorativa*, ai fini della produzione di un prototipo, è stata quella che ha visto l'applicazione del framework **Collie**. A tale scopo la progettazione è partita dalla scelta del **dataset**

che ha visto, in via definitiva, l'adozione di un dataset *sintetico*. Per la generazione di tale dataset sono stati presi in considerazione due tool online:

- **Mockaroo**.
- **Clearbox AI**[2].

Per lo scopo si è scelto di utilizzare Mockaroo poichè offre la possibilità di scegliere in modo piuttosto semplice le *regole* e le tipologie di dati da produrre. Esso permette infatti di produrre una vasta quantità di tipologie di dati sintetici che fanno riferimento a domini diversi e che, ad esempio, comprendono: nomi dei possibili dipartimenti aziendali, titoli di film, nomi scientifici di animali, indirizzi IP, nomi e cognomi di persone e molto altro ancora.

Il dataset che si è scelto di produrre comprende utenti, film ed eventuali interazioni. Per questi tre elementi si è scelto di produrre dei file JSON che verranno importati in memoria durante l'esecuzione del raccomandatore. Il file JSON degli utenti comprende 1000 utenti e colleziona, per ciascuno, i seguenti attributi:

- *id*.
- *username*.
- *fav_genre* (questa è una lista dei generi di film preferiti dall'utente).

Il file JSON degli item, che in questo caso sono dei film, conta anch'esso 1000 elementi. Ciascun elemento comprende i seguenti campi:

- *id*.
- *title*.
- *genre*.

Infine il file JSON contenente le interazioni tra utenti ed item comprende anch'esso 1000 elementi; ciascuna interazione è rappresentata dalla coppia *user_id-item_id*.

L'importazione del dataset avviene all'avvio del raccomandatore e, successivamente, viene presentato un *menù* interagibile da riga di comando (vedi figura 3.1). Da tale menù è possibile selezionare le seguenti funzionalità (di seguito, in ordine di presentazione):

[2] https://clearbox.ai

1. ***Addestramento di un modello dal principio o caricamento da file di un modello precedentmente addestrato***: Costituisce il primo passo da eseguire prima di poter impartire tutti i comandi successivi. Il modello addestrato viene serializzato e salvato su disco così da poterlo riutilizzare successivamente evitando di ri-addestrare il modello ogni volta. In caso di aggiornamento del dataset risulta opportuno addestrare nuovamente il modello.
2. ***Raccomandazioni per un utente casuale***: Sceglie un utente casuale e ne genera le raccomandazioni.
3. ***Valutazione del modello***: La valutazione viene eseguita tramite la metrica *Mean Average Precision at K (MAP@K)*.
4. ***Incorporare i metadati nel modello addestrato***: Consente di integrare i metadati relativi agli item così da migliorare l'accuratezza del modello addestrato.
5. ***Incorporare i metadati nella Funzione di Costo***: Consente di impartire una penalizzazione inferiore alla Funzione di Costo nel caso in cui essa classifichi con un punteggio maggiore un item senza interazione da parte dell'utente rispetto ad un item con cui l'utente ha interagito. Tutto ciò a patto che abbiano metadati simili (il genere in tal caso).
6. ***Raccomandazioni item-item***: Restituisce raccomandazioni di item simili all'item specificato tramite il suo id.
7. ***Raccomandazioni di item per un nuovo utente***.

È bene sottolineare che le raccomandazioni fornite tramite il *filtraggio collaborativo* sono strettamente legate al dataset e, più nello specifico, alle interazioni tra utenti ed item. Addestrare un modello di raccomandatore utilizzando una grande quantità di dati potrebbe permettere di ottenere delle raccomandazioni *più accurate* e dei *punteggi più elevati* nelle metriche di valutazione.

Infine, a conclusione della prototipazione tramite il framework *Collie*, si è notato che, al fine di generare le raccomandazioni, Collie costruisce un oggetto *Interactions*, comprendente le interazioni utente-item, da dare in input al modello da addestrare. Tale oggetto *Interactions* permette al modello di avere una visione dello spazio degli utenti e degli item che hanno avuto interazioni in precedenza ma esclude, ad esempio, utenti o item che non hanno mai avuto interazioni.

3.2.3 Utilizzo di LightFM

La seconda via percorsa durante la *fase esplorativa*, ai fini della produzione di un prototipo, è stata quella che ha visto l'applicazione del framework **LightFM**.

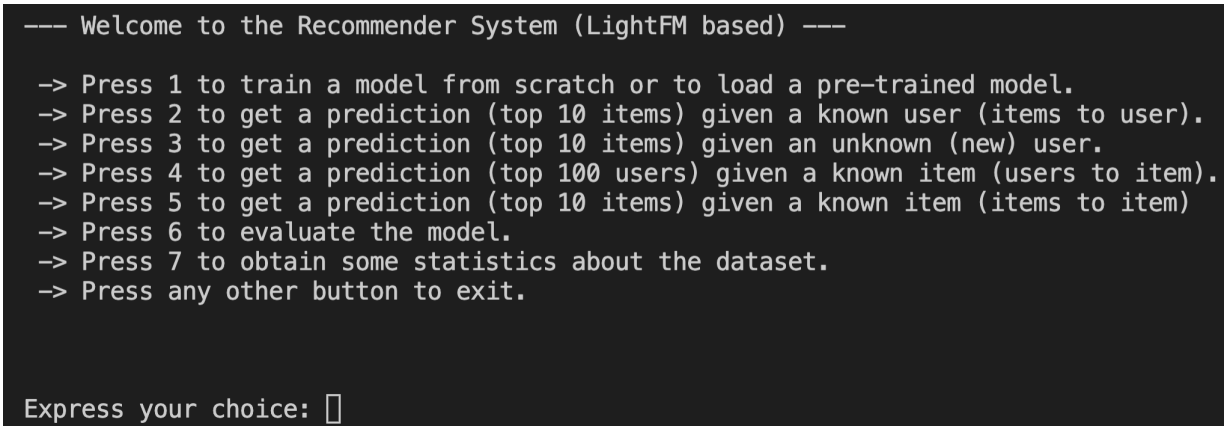

Figura 3.3: Menù principale del raccomandatore basato su LightFM.

Anche per questa implementazione è divenuta di fondamentale importanza la scelta del dataset. In questo caso si è scelta una modalità mista; in particolare, per quanto riguarda gli item, si è scelto di utilizzare due sottoinsiemi di dati già esistenti (circa 20.000 item per il primo insieme e circa 50.000 per il secondo) caratterizzati da tipologie di *features* differenti. Gli utenti invece, così come le interazioni, sono stati generati sinteticamente a partire da un'appropriata funzione di un modulo Python. Tale funzione genera 10.000 utenti ciascuno caratterizzato da interazioni con item casuali, sulla base delle features assegnate, e da features assegnate a partire da un bacino di features estratte dall' opportuno sottoinsieme selezionato.

È bene sottolineare che per il sottoinsieme che consta di 20.000 item si è scelto di generare, se possibile, al massimo 100 interazioni per utente mentre per il sottoinsieme da 50.000 item si è scelto di produrre al massimo 200 interazioni per utente.

Per entrambi i dataset si è scelto di attribuire 6 features per utente.

Utenti ed item sono caratterizzati dunque da alcuni attributi essenziali che comprendono un id e le relative features.

Nell'applicativo prodotto l'importazione dei dati relativi al file JSON degli item avviene all'avvio così come la generazione degli utenti con le

rispettive interazioni con gli item. Al fine di accelerare la fase di preparazione dell'intero *dataset*, durante l'avvio dell'applicativo, si è scelto di serializzare i dati processati per salvarli successivamente su disco; all'avvio è così possibile caricare e de-serializzare i dati pre-processati.

Come si evince dall figura 3.3, è possibile interagire con il raccomandatore attraverso un menù fruibile da riga di comando. Vengono rese dispobili le seguenti funzioni:

1. ***Addestramento di un modello dal principio o caricamento da file di un modello precedentmente addestrato***:
   Anche qui questo è il primo passo da eseguire prima di poter impartire tutti i comandi successivi. Il modello addestrato viene serializzato e salvato su disco così da poterlo riutilizzare successivamente evitando di ri-addestrare il modello ogni volta. In caso di aggiornamento del dataset risulta opportuno addestrare nuovamente il modello.
2. ***Raccomandazione dei migliori 10 item per un utente***:
   Fornisce una *top 10* ordinata degli item che dovrebbero essere raccomandati ad un dato utente del dataset (selezionando l'id dell'utente scelto).
3. ***Raccomandazione dei migliori 10 item per un utente sconosciuto (nuovo)***:
   Con questa funzionalità è possibile fornire una *top 10* ordinata degli item che dovrebbero essere suggeriti ad un nuovo utente date le features che vengono indicate per esso.
4. ***Raccomandazione dei migliori 100 utenti per un dato item***:
   Dato un item (selezionato tramite il suo id) è possibile ottenere una lista ordinata dei primi 100 utenti a cui l'item dovrebbe essere suggerito.
5. ***Raccomandazione dei migliori 10 item simili ad un dato item***:
   Questa funzionalità permette di ottenere una lista ordinata di item simili, in termini di features, all'item selezionato. Tale funzionalità viene realizzata sfruttando l'operazione di ***cosine similarity (similarità del coseno)***.
6. ***Valutazione del modello addestrato***:
   È possibile valutare il modello addestrato utilizzando le seguenti metriche:

- Mean Average Precision at K (MAP@K) (precisione);
- ROC AUC: Area Under the ROC Curve (curva ROC);
- Recall at K (richiamo);
- Reciprocal Rank (reciprocità);

7. ***Statistiche sul dataset caricato***:
   Tramite tale funzionalità è possibile ottenere delle statistiche, inerenti al dataset attualmente caricato dall'applicativo, quali:
   - Numero di utenti;
   - Numero di item;
   - Numero totale di features (condivise tra utenti ed item);
   - Dimensionalità e numero di interazioni, tra utenti ed item, entrambi estratti dalla ***Matrice delle Interazioni***;
   - Numero di features per utente scelto.

Alla luce di tale sperimentazione è emerso che il framework LightFM non presenta particolari criticità al fine di produrre un raccomandatore per il progetto CLIKC. Inoltre si sottolinea la particolare predisposizione per la risoluzione del problema del *Cold Start*.

Tabella 3.1: Tabella riassuntiva dei dataset utilizzati per LightFM.

	Numero di utenti	**Numero di item**	**Numero totale di features**	**Interazioni totali**	**Features per utente**
Dataset 1	10.000	21.324	40	985.846 (max 100 per utente)	6
Dataset 2	10.000	50.297	40	1.992.321 (max 200 per utente)	6

3.2.4 Analisi dei Risultati Ottenuti

Nella presente sottosezione si vogliono analizzare le raccomandazioni ottenute nella *Fase Esplorativa* del progetto, e generate attraverso l'applicazione prototipale basata sul framework ***LightFM***. Tale *analisi* sarà eseguita

sui due dataset presentati nella sottosezione 3.2.3 e nominati, per convenzione, **Dataset 1** e **Dataset 2**. Per tali dataset verranno presentate delle infografiche che mostrano le caratteristiche dei due dataset così da fornire una ragione alle raccomandazioni ottenute. Si vuole notare inoltre che, per ciascun Dataset, gli item sono contraddistinti da due categorie di features, che chiameremo per convenzione **Categoria 1** e **Categoria 2**; alla Categoria 1 appartengono features come, ad esempio, *acf fiorentina, ambiente e spettacolo*, invece alla Categoria 2 appartengono features come, ad esempio, *national, Roma e Torino*. Le inforgrafiche vengono riportate per entrambe le categorie.

Le raccomandazioni verranno generate per un campione di *3 utenti* (generati sinteticamente) del dataset e per un *nuovo utente* che non abbia ancora interagito con nessun item: con tale presupposto si potranno ottenere delle evidenze su raccomandazioni generate tramite l'approccio ibrido, traendo vantaggio dal *Filtraggio Collaborativo* (vedi utenti che abbiano già interagito) e dall'approccio *Basato sul Contenuto* (vedi la fase di *Cold Start*). Si noti che ai 3 utenti generati sinteticamente vengono assegnate, casualmente, 3 features per ciascuna delle due categorie; inoltre si è scelto, per la Categoria 2, di assegnare sempre la feature *national*. Le interazioni utente-item sono state generate con la condizione che le feature di un item, per entrambe le categorie, combacino con quelle assegnate ad un utente. Per quanto riguarda un *nuovo utente* che non abbia ancora interagito, si è scelto di assegnare 4 features scelte casualmente tra le due categorie di features.

Infine, si vuole sottolineare che le raccomandazioni sono generate con l'ausilio di una *Matrice Utente-Feature* e di una *Matrice Item-Feature*: con esse le raccomandazioni saranno maggiormente personalizzate.

Per quanto riguarda il **Dataset 1** la distribuzione di item per feature è illustrata nell'infografica 3.4.

Da questa infografica emerge come per alcune features appartenenti Categoria 1, vedasi ad esempio *acf fiorentina, cronache e sport*, vi sia un numero maggiore di item; lo stesso accade per la feature *national* inerente alla Categoria 2. Da ciò ne consegue che, essendo le interazioni generate sinteticamente ed in modo casuale secondo le features degli utenti, per questi ultimi si tenderà ad ottenere ***più interazioni*** con gli item caratterizzati dalle features ***più frequenti***, a patto che una o più di esse siano state assegnate agli utenti in fase di generazione. Si presenta di seguito

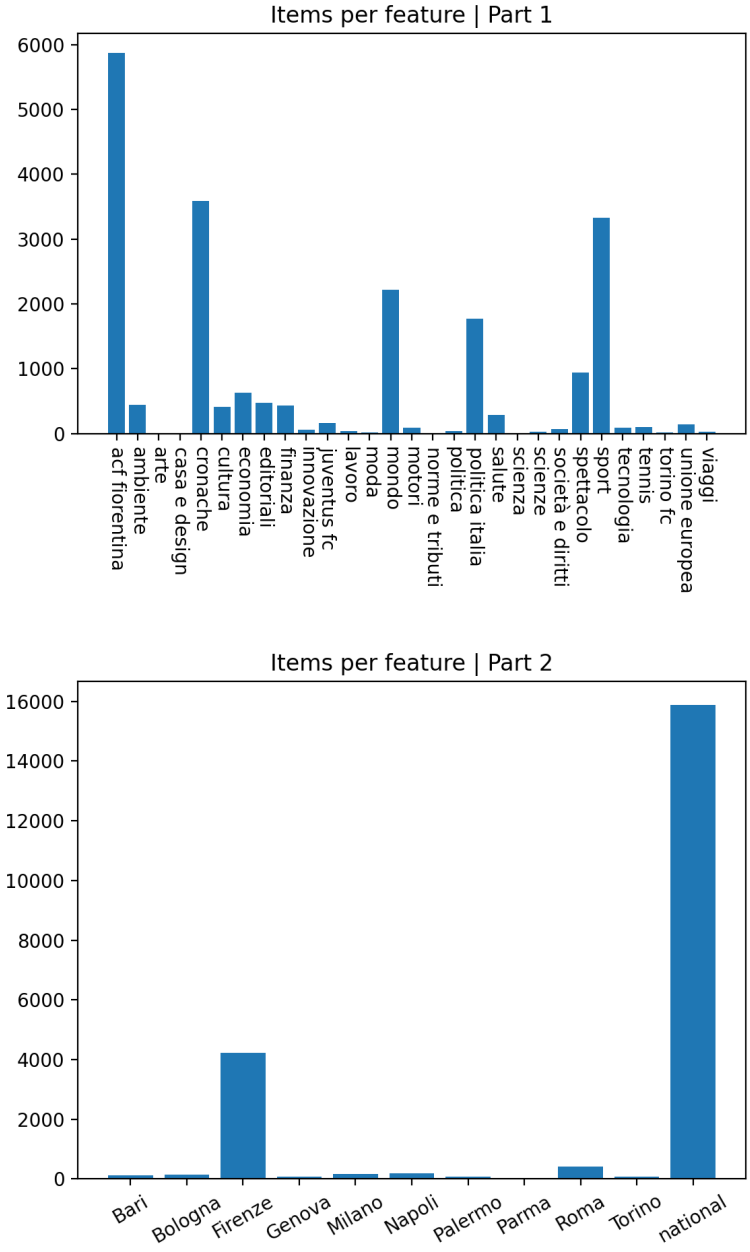

Figura 3.4: Infografica del Dataset 1 su distribuzione item per feature.

l'infografica 3.5, inerente alle interazioni generate, che illustra la distribuzione delle interazioni per feature, secondo le due categorie disponibili. Dall'infografica si può notare una correlazione tra il numero di interazioni per feature e la frequenza di queste tra gli item.

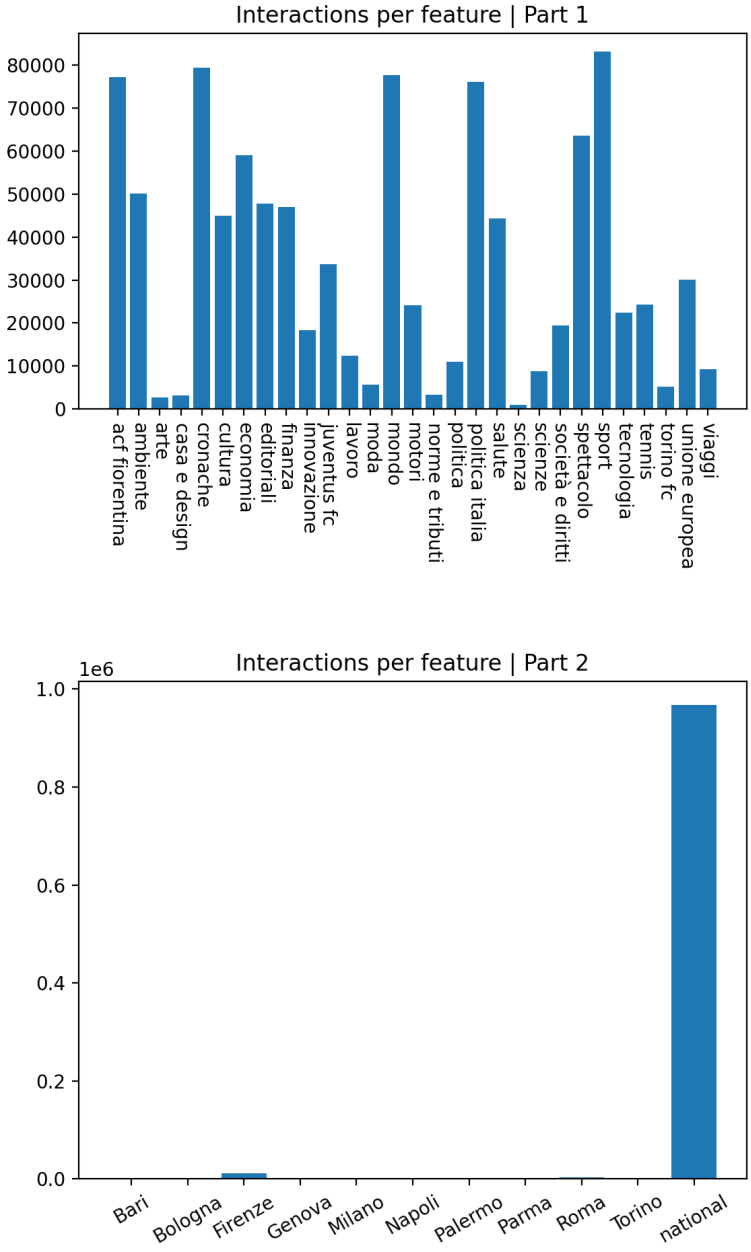

Figura 3.5: Infografica del Dataset 1 su distribuzione interactions per feature.

Si prenda ora in considerazione un campione di 3 utenti del dataset, per convenzione chiamati *Utente 1, Utente 2 ed Utente 3.* L'Utente 1 presenta le features *editoriali, salute, società e diritti* per la Categoria 1 e

national, Torino, Bari per la Categoria 2. L'Utente 2 presenta le features *juventus fc, viaggi, politica* per la Categoria 1 e *national, Torino, Bari* per la Categoria 2. L'Utente 3 presenta le features *ambiente, società e diritti, salute* per la Categoria 1 e *national, Palermo, Napoli* per la Categoria 2.

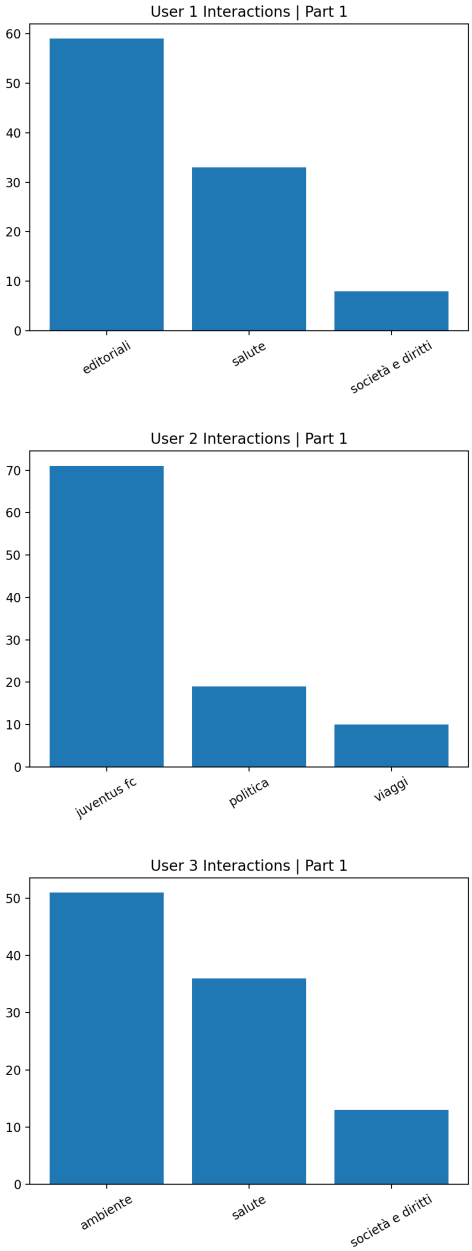

Figura 3.6: Infografiche degli Utenti 1, 2 e 3 del Dataset 1 su distribuzione interactions per feature. Categoria 1.

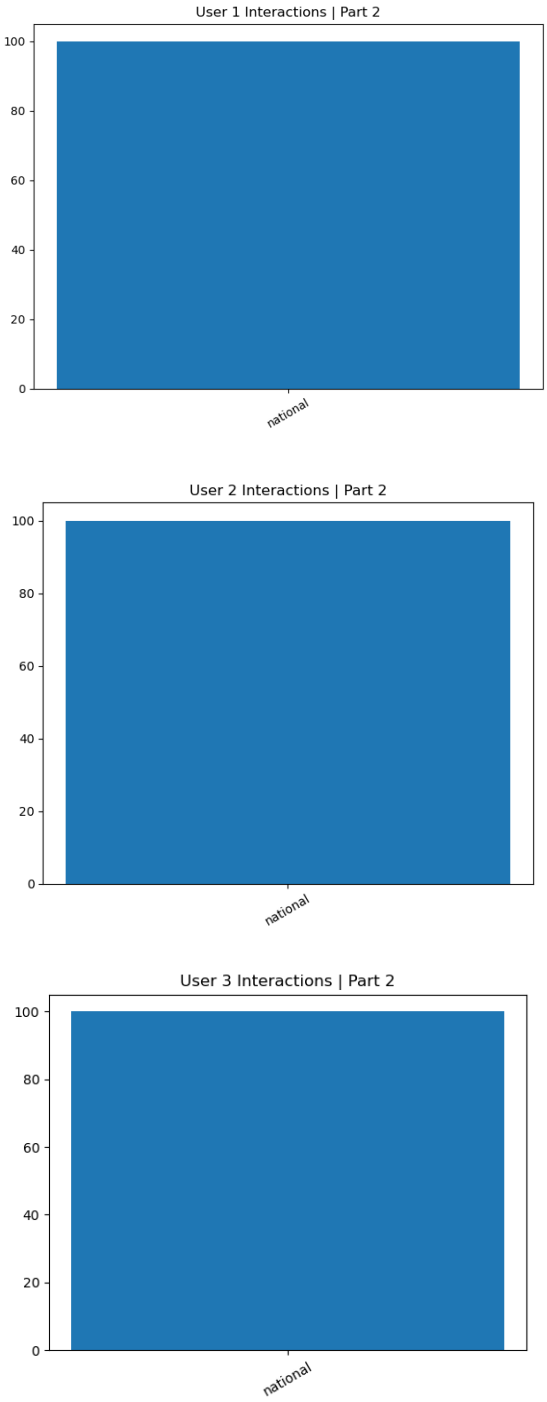

Figura 3.7: Infografiche degli Utenti 1, 2 e 3 del Dataset 1 su distribuzione interactions per feature. Categoria 2.

Dalle infograficahe 3.6 e 3.7 è possibile osservare che, in coerenza con quanto precedentemente affermato, le interazioni generate per ognuno degli utenti sono composte da un quantitativo maggiore di item che sono caratterizzati da determinate features, ovvero quelle più *frequenti* tra gli item del dataset.

La conseguenza diretta di questa osservazione è che i suggerimenti finali, generati dal raccomandatore, saranno fortemente influenzati da tutto ciò: osservando le raccomandazioni generate (ordinate secondo un punteggio decrescente) nella figura 3.8, si deduce che verranno suggeriti maggiormente item correlati con le features più frequenti, riducendo la diversità delle raccomandazioni in alcuni casi, come mostrato ad esempio per l'Utente 1.

score	theme	place
-103598.671875	editoriali	national
-103598.796875	editoriali	national
-103599.117188	editoriali	national
-103599.125000	editoriali	national
-103599.203125	editoriali	national
-103599.218750	editoriali	national
-103599.382812	editoriali	national
-103599.406250	editoriali	national
-103599.421875	editoriali	national
-103599.445312	editoriali	national

score	theme	place
-100766.968750	juventus fc	national
-100767.164062	politica	national
-100767.218750	juventus fc	national
-100767.242188	juventus fc	national
-100767.296875	politica	national
-100767.320312	juventus fc	national
-100767.343750	politica	national
-100767.359375	juventus fc	national
-100767.406250	juventus fc	national
-100767.453125	juventus fc	national

score	theme	place
-103032.750000	ambiente	national
-103032.812500	salute	national
-103032.906250	società e diritti	national
-103032.929688	salute	national
-103032.960938	società e diritti	national
-103032.960938	società e diritti	national
-103032.984375	ambiente	national
-103032.992188	società e diritti	national
-103033.031250	salute	national
-103033.031250	ambiente	national

Figura 3.8: Utenti 1, 2 e 3 del Dataset 1. Top 10 dei risultati.

Per un nuovo utente che non abbia mai interagito con gli item, considerandolo caratterizzato dalle features *"unione europea", "finanza", "casa e design", "torino fc"*, si ottengono i risultati della figura 3.9.

score	theme	place
-171168.750000	finanza	national
-171169.234375	finanza	national
-171169.921875	finanza	national
-171170.453125	finanza	national
-171171.125000	casa e design	national
-171171.125000	finanza	national
-171171.640625	finanza	national
-171171.890625	finanza	national
-171172.718750	finanza	national
-171172.796875	finanza	national

Figura 3.9: Nuovo utente del Dataset 1. Top 10 dei risultati.

Per quanto riguarda il **Dataset 2**, si presentano a scopo illustrativo, parallelamente a quanto fatto con il Dataset 1, le stesse tipologie di infografiche. Inoltre anche qui vogliamo fornire dei dati per un campione di 3 utenti del dataset e le rispettive raccomandazioni.

L'Utente 1 presenta le features *tecnologia, acf fiorentina, politica italia* per la Categoria 1 e *national, Roma, Bari* per la Categoria 2. L'Utente 2 presenta le features *acf fiorentina, arte, finanza* per la Categoria 1 e *national, Genova, Parma* per la Categoria 2. L'Utente 3 presenta le features *finanza, salute, politica italia* per la Categoria 1 e *national, Bari, Napoli* per la Categoria 2. Infine si forniscono i risultati per un nuovo utente.

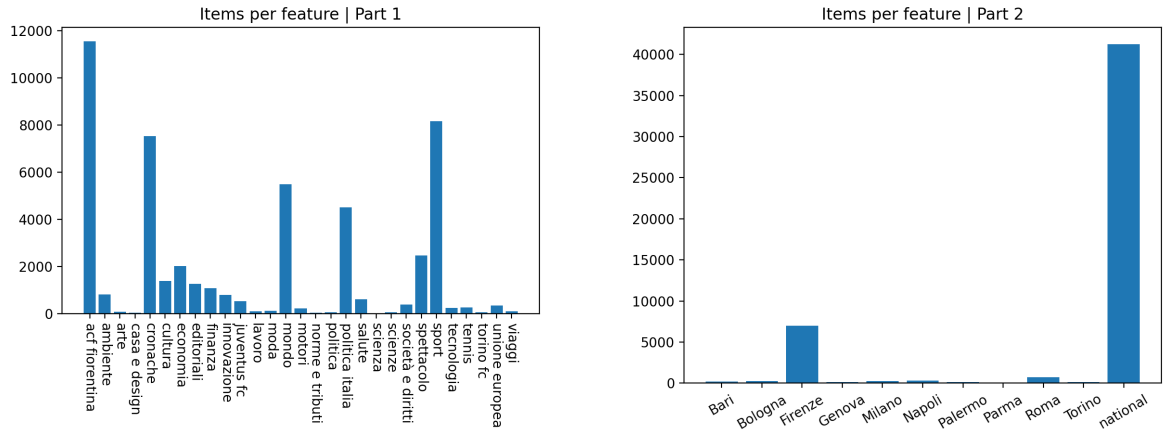

Figura 3.10: Infografica del Dataset 2 su distribuzione item per feature.

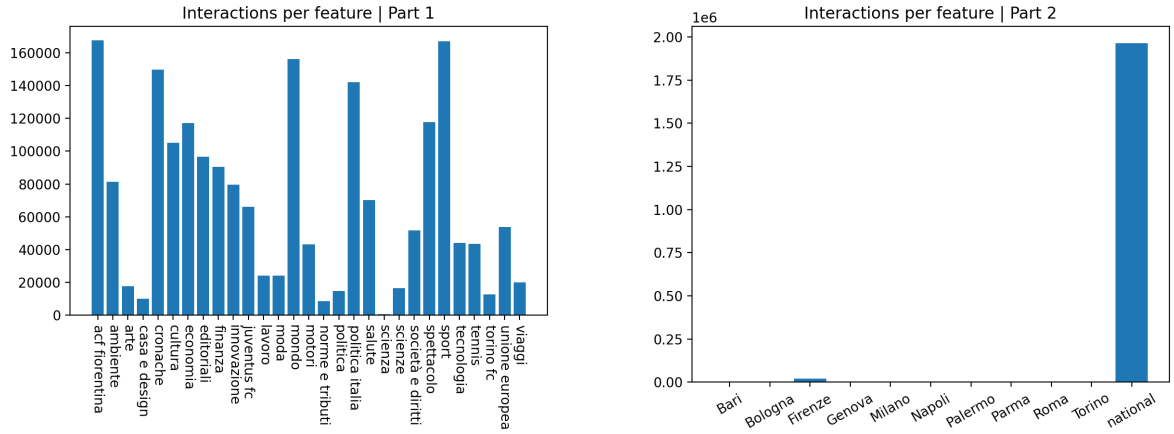

Figura 3.11: Infografica del Dataset 2 su distribuzione interactions per feature.

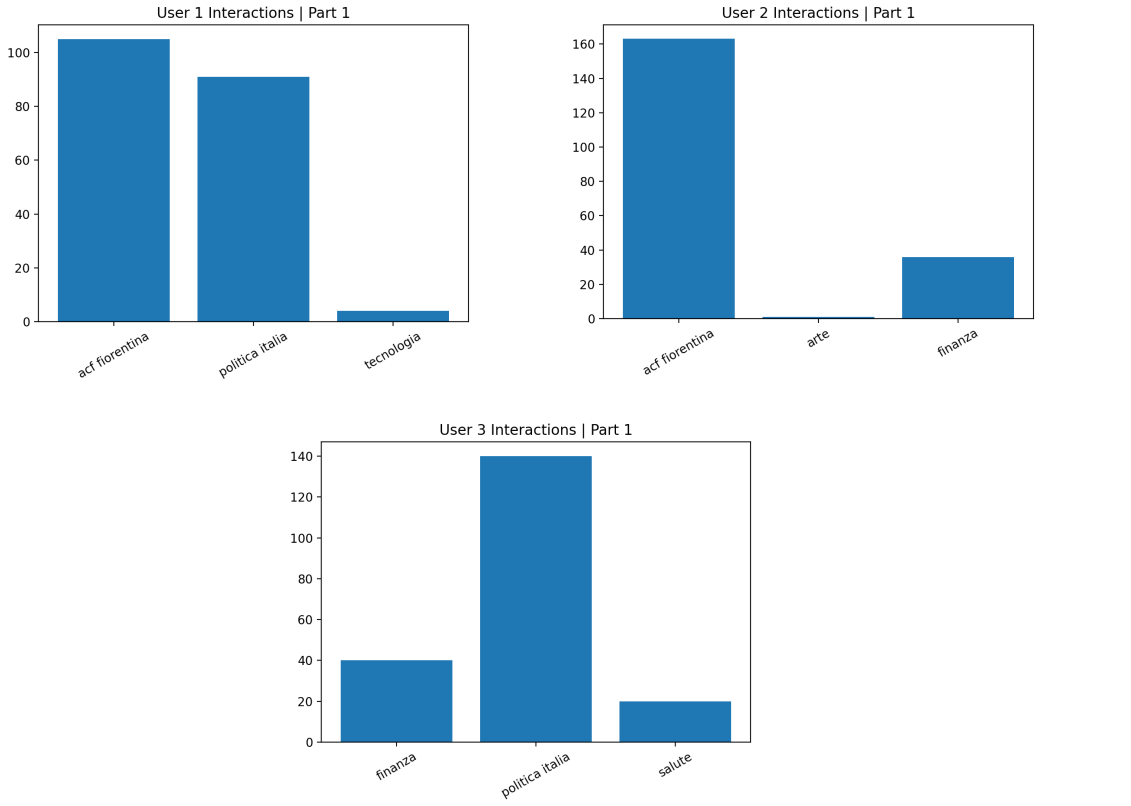

Figura 3.12: Infografiche degli Utenti 1, 2 e 3 del Dataset 2 su distribuzione interactions per feature. Categoria 1.

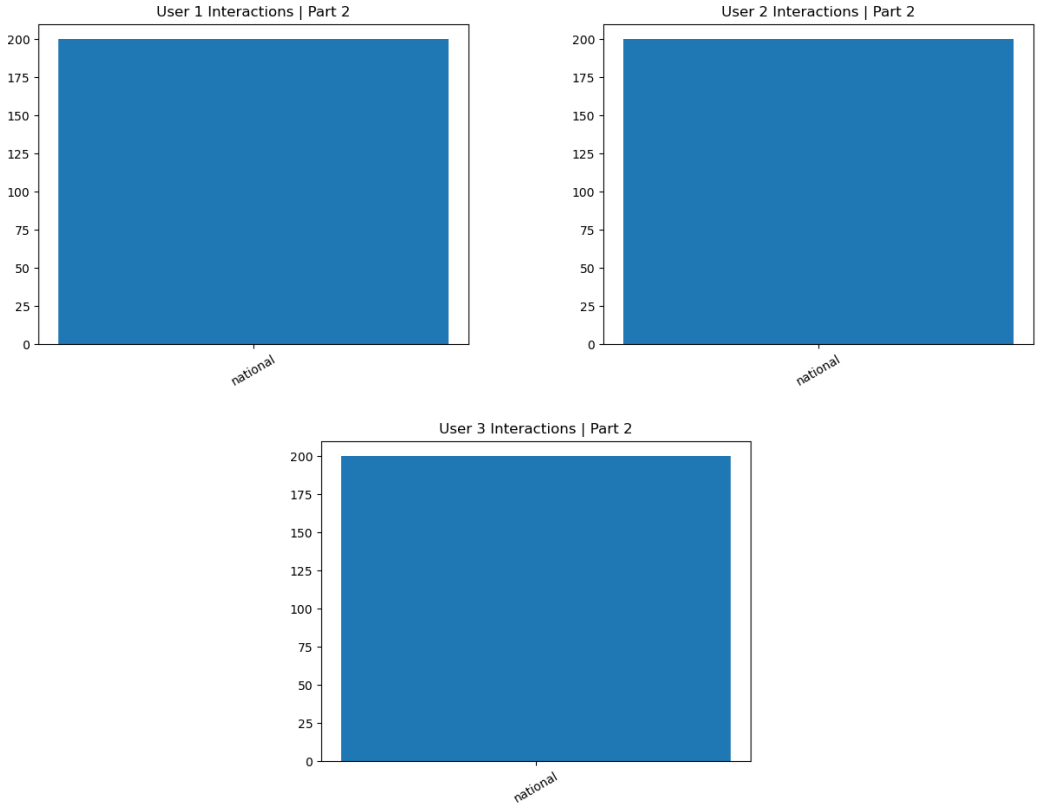

Figura 3.13: Infografiche degli Utenti 1, 2 e 3 del Dataset 2 su distribuzione interactions per feature. Categoria 2.

score	theme	place
-197179.093750	acf fiorentina	national
-197179.125000	acf fiorentina	national
-197179.171875	politica italia	national
-197179.234375	acf fiorentina	national
-197179.453125	politica italia	national
-197179.468750	politica italia	national
-197179.515625	politica italia	national
-197179.515625	acf fiorentina	national
-197179.531250	acf fiorentina	national
-197179.625000	politica italia	national

score	theme	place
-193033.718750	acf fiorentina	national
-193033.875000	acf fiorentina	national
-193034.015625	acf fiorentina	national
-193034.109375	acf fiorentina	national
-193034.218750	acf fiorentina	national
-193034.390625	acf fiorentina	national
-193034.437500	acf fiorentina	national
-193034.468750	acf fiorentina	national
-193034.500000	acf fiorentina	national
-193034.500000	acf fiorentina	national

score	theme	place
-193953.156250	finanza	national
-193953.203125	finanza	national
-193953.515625	finanza	national
-193953.562500	finanza	national
-193953.718750	salute	national
-193953.765625	finanza	national
-193953.796875	finanza	national
-193953.828125	politica italia	national
-193953.828125	politica italia	national
-193953.859375	finanza	national

Figura 3.14: Utenti 1, 2 e 3 del Dataset 2. Top 10 dei risultati.

Per un nuovo utente, caratterizzato dalle features *"società e diritti", "salute", "lavoro", "tennis"*, si ottengono i risultati in figura 3.15.

```
        score  theme    place
-440661.59375 salute national
-440663.78125 salute national
-440664.06250 salute national
-440664.78125 salute national
-440665.21875 salute national
-440665.34375 salute national
-440665.90625 salute national
-440666.12500 salute national
-440666.18750 salute national
-440666.28125 salute national
```

Figura 3.15: Nuovo utente del Dataset 2. Top 10 dei risultati.

A conclusione di questa sottosezione si forniscono i punteggi, indicati nella Tabella 3.2, inerenti alla metrica *"Area Sottostante la Curva ROC"*, per entrambi i dataset, relativamente ai dati utilizzati per l'addestramento (*Train*) del raccomandatore e ai dati di (*Test*).

Tabella 3.2: Tabella della metrica "Area Sottostante la Curva ROC" per il raccamondatore prototipale basato su LightFM.

	Train Set	**Test Set**
Dataset 1	0.9850049614906311	0.9424001574516296
Dataset 2	0.9847089052200317	0.9416191577911377

3.3 Architettura del Raccomandatore

Nella sezione corrente si vuole presentare l'**architettura** software ideata, in fase di progettazione, per lo sviluppo del sistema di raccomandazione. A tal fine, viene presentato lo *schematico* dell'architettura che è stato prodotto ed una descrizione accurata di quest'ultimo in termini di tecnologie utilizzate, componenti di progetto e loro finalità.

Con lo scopo di fornire una visione generale sull'architettura software del progetto, lo schematico in Figura 3.16 ci permette di osservare tre macro-aree di progetto ben distinte:

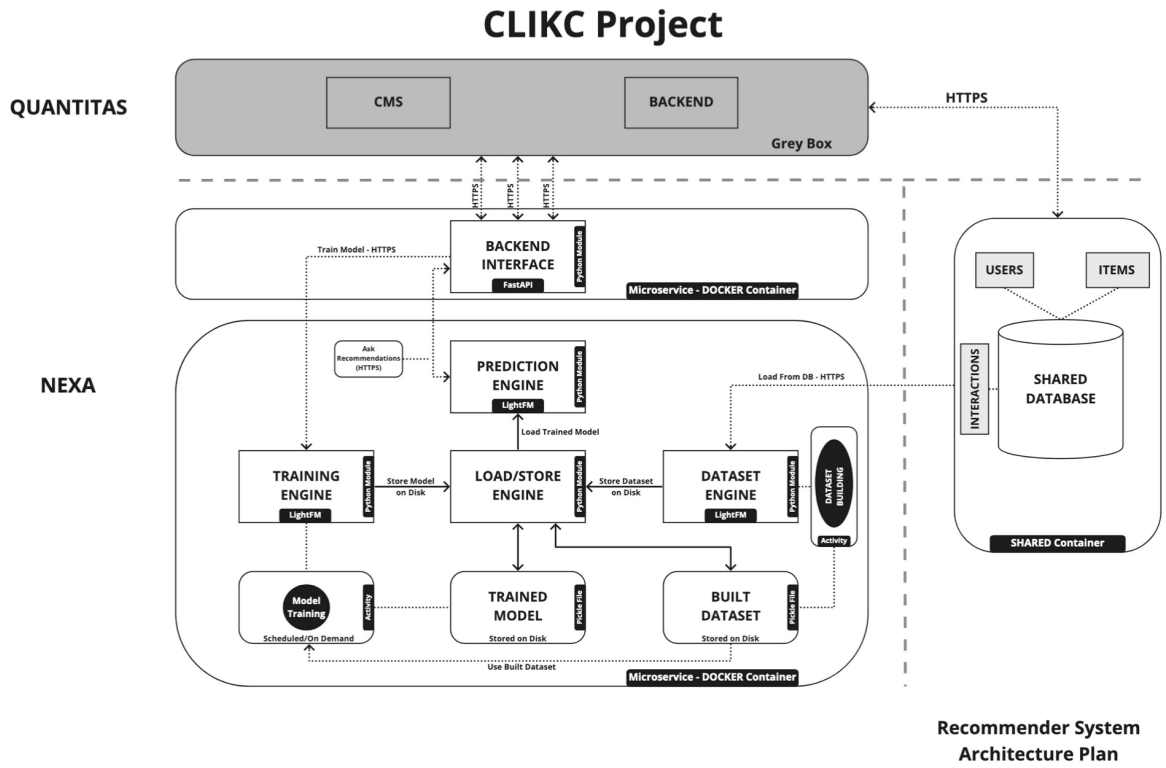

Figura 3.16: Schematico dell'Architettura del Raccomandatore per il progetto CLIKC.

1. L'area relativa al **Backend** della piattaforma del progetto CLIKC, sviluppato dal partner di progetto Quantitas Srl: esso ha lo scopo di interfacciarsi con il **Sistema di Raccomanazione**, al fine di ottenere delle raccomandazioni da presentare poi all'utente finale.

2. Un'area dedicata al **Database Condiviso**, contente tutti i *dati* e i *metadati* sugli ***item*** relativi ai contenuti sviluppati dai partner didattici di progetto. Tale Database contiene inoltre le informazioni sugli ***utenti*** registrati alla piattaforma, essenziali al fine di ottenere raccomandazioni personalizzate. Infine esso contiene i dati essenziali al funzionamento del *Filtraggio Collaborativo*: le ***interazioni***. Gli item vengono inseriti nel Database tramite l'ausilio di un **Content Management System** sviluppato da Quantitas Srl.

3. Infine abbiamo l'area relativa al **Sistema di Raccomandazione**, di competenza del Centro Nexa su Internet & Società - Politecnico di

Torino. Esso è stato sviluppato con lo scopo di fornire raccomandazioni a partire da un *Dataset* costruito sulla base dei dati estratti dal Database Condiviso.

Il **Sistema di Raccomandazione** è stato strutturato nella forma di due Microservizi inglobati in opportuni **Container Docker**. Tali Microservizi possono essere contattati tramite un ulteriore servizio, offerto anch'esso tramite Container Docker, che agisce da *Reverse Proxy* sfruttando la tecnlogia offerta da **NGINX**[3].

L'intero sistema applicativo multi-container viene definito ed eseguito tramite il tool **Docker Compose**[4]. La configurazione, inerente all'avvio dei microservizi sopracitati, avviene tramite la definizione di un *file YAML*. Con tale file è possibile esplicitare i servizi da eseguire, le dipendenze di avvio di questi, l'*esposizione* e il *mapping* (tra host e container) delle porte in ambito networking e molto altro ancora. Configurando un *Dockerfile* all'interno della directory principale di ogni applicativo (che nel nostro caso sono offerti nella forma di microservizio), e racchiudendo tali directory in una directory principale che a sua volte contiene il file YAML, è possibile eseguire l'intero sistema di microservizi con un singolo comando impartito da riga di comando:

- ***docker-compose up -d***.

Tale comando andrà a generare i file immagine Docker, per gli applicativi desiderati, ottenendo dal repository ufficiale[5] ulteriori immagini Docker da cui queste dipendono, secondo un sistema di dipendenze ben definito, e successivamente verranno generati ed infine eseguiti i Container Docker.

I due microservizi sono stati sviluppati tramite il framework web **FastAPI**[6], basato su Python. Tale framework permette di esporre le funzionalità offerte da tali microservizi secondo il paradigma delle API REST.

FastAPI, secondo la documentazione al riferimento [18], offre tutta una serie di funzionalità e vantaggi tra cui: velocità di sviluppo, alte prestazioni, documentazione interattiva generata automaticamente e riduzione

[3] https://nginx.org/en/

[4] https://docs.docker.com/compose/

[5] https://hub.docker.com

[6] https://fastapi.tiangolo.com/

del 40% della probabilità di introdurre bug nel codice.

Il sistema prodotto dal Centro Nexa su Internet & Società - Politecnico di Torino permette di esporre, in cascata, il servizio chiamato **Backend Interface** e quello relativo al **Modello**.

Il microservizio di Backend Interface è raggiungibile dall'esterno, tramite degli *endpoint (url)* ben definiti, ed ha il compito di raccogliere le richieste provenienti dall'esterno, nel nostro caso il Backend prodotto da Quantitas Srl. Ciascun endpoint definisce una richiesta per una specifica tipologia di raccomandazione o per un'altra funzionalità. Tali richieste vengono inoltrate, tramite API REST, agli endpoint opportuni del microservizio del Modello.

La comunicazione tra i due microservizi è stata implementata con l'ausilio della libreria **AIOHTTP**[7].

Il microservizio del Modello, che può essere contattato esclusivamente dal Backend Interface, costituisce il componente fondamentale dell'intero Sistema di Raccomandazione; esso è composto da 4 moduli principali:

- **Dataset Engine**: Contatta il *Database Condiviso* per ottenere i dati[8] relativi ad utenti, item ed interazioni. Successivamente, a valle di alcune elaborazioni sui dati in ingresso, viene eseguita l'attività di *costruzione* del **Dataset** che sarà dato in input al Modello da addestrare. Tra le elaborazioni sugli item in input avviene anche un'attività di estrazione di *parole chiave* di rilevanza, eseguita tramite *Named Entity Recognition*. Tali parole chiave vengono utilizzate per meglio caratterizzare (o descrivere) gli item a disposizione. In tale contesto l'estrazione viene eseguita con l'ausilio della libreria **spaCy**[9], utilizzata in ambito *Natural Language Processing*.
- **Load/Store Engine**: È deputato al salvataggio, su disco in locale ed in formato *pickle*, del Dataset costruito e del Modello addestrato. Questo modulo Python si occupa anche di caricare tali file in memoria quando è necessario generare le raccomandazioni. Il Load/Store Engine infatti interagisce direttamente con i moduli **Prediction Engine**, **Dataset Engine** e **Training Engine**.

[7] https://docs.aiohttp.org/en/stable/

[8] Si noti che, in attesa dell'ottenimento dei dati di progetto definitivi, viene utilizzato il dataset provvisorio utilizzato nelle fasi di analisi.

[9] https://spacy.io

- **Training Engine**: Costituisce il componente fondamentale del microservizio del Modello; esso si occupa, caricato in mermoria il Dataset precedentemente costruito, di addestrare il Modello, secondo degli *iper-parametri* ben definiti, tramite l'attività di **Training** (addestramento). In tale contesto, il framework utilizzato per il raccomandatore è **LightFM**. Al termine del training il modello viene salvato come file *pickle* con l'ausilio del modulo **Load/Store Engine**. È bene notare che l'addestramento del Modello è stato programmato per essere eseguito manualmente, tramite una specifica API, oppure per essere ripetuto in modo automatico dopo un certo intervallo di tempo ben definito. A supporto di tale *scheduling* automatizzato è stata utilizzata la libreria **APScheduler**[10].
- **Prediction Engine**: Quest'ultimo modulo Python è predisposto alla generazione delle raccomandazioni. Una volta contattata l'API opportuna viene validata la richiesta e successivamente vengono caricati in memoria il Dataset costruito e il Modello precedentemente addestrato. A questo punto vengono generate le raccomandazioni i cui risultati verranno poi restituiti al microservizio di Backend Interface e, rispettivamente in successione, al Backend sviluppato da Quantitas Srl.

3.4 I Dati a Disposizione

Nella presente sezione si vogliono discutere la *struttura* e il *significato* dei dati del progetto CLIKC. Nel dettaglio vogliamo discutere i dati inerenti agli item che, quando saranno disponibili, verranno forniti come input al Sistema di Raccomandazione.

Come precedentemente anticipato nella sezione 2.3, l'obiettivo dell'Intellectual Output 1 è stato quello di produrre un ***catalogo*** di contenuti didattici che avesse come scopo la *formazione* finalizzata al potenziamento delle *skills trasversali*. I *partner didattici* di progetto, i quali si sono dedicati a tale compito, sono: ETI Malta (Executive Training Institute Ltd.), Tecum Srl, Acción Laboral, bit Schulungscenter GmbH, Association Europeenne pour la Formation professionnelle.

[10] https://apscheduler.readthedocs.io/en/3.x/index.html

Al fine di fornire dei contenuti conformi al metodo didattico *Content Language Integrated Learning and Micro-Learning*, sono state selezionate **4 skills** o abilità. Ogni skill va intesa come una sorta di area tematica o aggregato a cui afferiscono le competenze scelte nel dettaglio. Queste ultime sono organizzate in moduli che sono tecnicamente definiti come ***Cluster***. A ciascun Cluster, appartenente ad una specifica area tematica di skills, viene dunque associata una competenza ben definita.

All'area della **skill 1** sono associate competenze, raccolte in Cluster, inerenti alla *gestione della complessità*, *gestione dello stress* e *gestione ottimizzata del tempo*. Per ciò che concerne l'area della **skill 2** troviamo tematiche quali il *pensiero critico*, la *presa di decisioni* e il *problem solving*. L'area inerente alla **skill 3** include competenze relazionali quali la *comunicazione costruttiva*, la gestione della *fiducia e della tolleranza* e l'*intelligenza emotiva*. Infine nell'area tematica affrontata dalla **skill 4** confluiscono competenze quali la *leadership*, il *lavoro di squadra* e il *lavoro in autonomia*.

Ogni Cluster è strutturato in modo tale da inglobare un certo quantitativo di contenuti didattici, definiti ***Learning Units*** (Unità di Apprendimento). Infine ogni Cluster viene proposto per 3 diversi livelli di ***EQF***[11] (European Qualifications Framework), i quali fanno riferimento alla capacità di comprensione ed elaborazione dei contenuti da parte dell'utente. In particolare nel progetto sono offerti contenuti disponibili per i livelli **EQF2**, **EQF3** ed **EQF4**. Tutti i contenuti sono offerti in lingua Inglese; il livello EQF4 prevede anche dei contenuti in Italiano, Spagnolo e Tedesco. Al completamento di ciascun Cluster è infine presente un test da proporre all'utente finale al fine di fornire una valutazione sugli obiettivi di apprendimento raggiunti.

Una singola Unità di Apprendimento si presenta come un contenuto, consumabile in breve tempo, che può essere composto da testi, immagini, video e audio inerenti all'argomento trattato. Infine ciascun contenuto didattico è contrassegnato da *parole chiave*, appartenenti ad un vocabolario generato appositamente, aventi lo scopo di migliorare i risultati del Sistema di Raccomandazione.

[11] https://europa.eu/europass/en/european-qualifications-framework-eqf

Capitolo 4

Considerazioni Finali e Direzioni Future

Con il presente capitolo si vuole fornire una sintesi inerente alle tematiche trattate nei precedenti capitoli con l'aggiunta di riflessioni sui maggiori risultati ottenuti. Inoltre si vuole proporre una possibile direzione futura al fine di migliorare il Raccomandatore sviluppato.

La presente esposizione tratta dello sviluppo, ancora in corso e con conclusione fissata a Maggio 2023, di un Sistema di Raccomandazione per il progetto europeo *CLIKC - Content and Language Integrated learning for Key Competences*, promosso dall'Erasmus+ Programme, tramite la mia collaborazione con il Centro Nexa su Internet & Società - Politecnico di Torino.

Il progetto è stato concepito con l'obiettivo finale di fornire una piattaforma web che proponga contenuti basati sulla metodologia didattica *Content Language Integrated Learning and Micro-Learning.* Tramite il potenziamento delle competenze personali, ci si pone come fine ultimo la riduzione della disoccupazione e degli ingenti costi che ne derivano.

Nel contesto di tale progetto, il Raccomandatore, oggetto di questa Tesi, si pone lo scopo di fornire un contributo sostanziale alla formazione degli utenti della piattaforma; il Raccomandatore è infatti in grado di generare dei suggerimenti per il prossimo contenuto da fruire sulla base della storia passata dell'interazione dell'utente con la piattaforma e sulla base delle caratteristiche degli utenti e degli item.

La prima fase del lavoro svolto ha visto una fase di ricerca e documentazione sui principali modelli e approcci utilizzati nell'ambito dei raccomandatori; ne emerge che l'Approccio Ibrido risulta essere particolarmente valido in quanto attenua il fenomeno del *Cold Start* e combina l'efficacia degli approcci e dei modelli inerenti al Filtraggio Collaborativo e di quelli Basati sul Contenuto. Durante tale fase sono stati selezionati due framework per lo sviluppo di raccomandatori, LightFM e Collie, scelti sulla base delle funzionalità offerte.

La seconda fase del lavoro, definita *"esplorativa"*, ha visto lo sviluppo di due applicativi prototipali, strutturati similmente, e basati rispettivamente su Collie e LightFM; tutto ciò ha permesso di evidenziare analogie e differenze tra i due framework in termini di processamento dei dati di input, tipologie di raccomandazione fornite in output e metriche di valutazione incorporate. Nel contesto di tale fase sono stati utilizzati dataset sintetici e dataset pre-esistenti. In conclusione è stato scelto il framework LightFM, per il futuro sviluppo del progetto, per la sua versatilità e le funzionalità fornite, tra cui la facile risoluzione del problema del *Cold Start*.

Successivamente a tale scelta sono stati condotti dei test specifici, utilizzando due dataset pre-esistenti a disposizione, per testare l'efficacia delle raccomandazioni; i metadati sugli item hanno evidenziato, con l'ausilio delle infografiche presentate, che per le feature utilizzate dal raccomandatore ne esistevano alcune a cui erano associati un maggiore quantitativo di item rispetto ad altre. Le interazioni utente-item sono state generate sinteticamente ed in modo casuale, a patto che una determinata feature dell'item fosse annoverata tra quelle caratterizzanti un dato utente. Dunque, ad un utente a cui veniva associata una determinata feature, appartenente all'insieme delle feature più frequenti tra gli item, venivano attribuite con alta probabilità un numero maggiore di interazioni con item associati a quella feature. Il risultato diretto di tutto ciò, in termini di raccomandazioni, è stata la maggiore presenza di suggerimenti di item che erano associati con le feature più utilizzate.

Nella parte finale dell'esposizione, si è poi passati ad una presentazione inerente all'effettivo sviluppo del progetto, discutendo l'architettura software proposta: si tratta di una soluzione composta da due microservizi, disposti in cascata, gestiti tramite Docker Compose. Il microservizio nominato *"Backend Interface"* ha il compito di *dialogare* con il backend della

piattaforma, sviluppato dal partner di progetto Quantitas Srl, e di inoltrare le richieste al microservizio che gestisce le attività del raccomandatore. Infine la trattazione è stata conclusa con una riflessione sui dati degli item che, quando saranno disponibili, verranno forniti in ingresso al raccomandatore: ne è stata spiegata la struttura e il significato delle categorie di cui fanno parte.

A conclusione del capitolo si vogliono suggerire e/o valutare possibili migliorie da applicare al progetto. Al fine di migliorare le raccomandazioni per l'utente finale si potrebbe agire in diversi modi.

Ad esempio si potrebbe intraprendere un'attività dedicata ad una migliore scelta degli iper-parametri utilizzati dal framework LightFM durante il training del modello (si veda ad esempio il parametro "learning rate").

Si può inoltre valutare anche un'ulteriore attività di *Feature Engineering* volta alla rielaborazione di determinate features inerenti ad utenti ed item così da meglio caratterizzarli.

Si potrebbe prendere in considerazione di agire, tramite la funzionalità offerte da LightFM ed in fase di costruzione del relativo Dataset, sui *pesi* numerici che è possibile assegnare alle singole interazioni utente-item e/o all'associazione di un utente con una determinata feature. Tali pesi potrebbero essere scelti arbitrariamente, sulla base di nuove necessità di progetto, per andare a penalizzare o favorire determinate categorie di item o specifici item; inoltre questi potrebbero portare ad un parziale risoluzione dei problemi di sbilancimento del dataset a disposizione.

In termini di funzionalità aggiuntive, pensando anche al di fuori dei vincoli imposti dal progetto, si potrebbe pensare di implementare un sistema di *approfondimenti* da proporre all'utente che non alteri il suo percorso di apprendimento di base; gli approfondimenti non sarebbero altro che porzioni di Learning Unit già esistenti. Tali approfondimenti verrebbero presentati nella stessa pagina della Learning Unit che l'utente sta attualmente consultando. Si potrebbe dunque ipotizzare che le Learning Unit siano scomposte ulteriormente, in modo più granulare, e di eseguire su queste porzioni un'operazione di keyword extraction: le keyword potrebbero essere dunque utilizzate come feature in un *raccomandatore di approfondimenti*, appositamente progettato, che funzionerebbe parallelamente a quello già implementato per le Learning Unit. Ciò consentirebbe all'utente di ampliare il suo spazio di conoscenze e di massimizzare l'utilizzo dei

contenuti prodotti.

Bibliografia

[1] B. Rocca, "Introduction to recommender systems," towardsdatascience.com, Jun. 3rd 2019, [ultimo accesso in data: 19 Aprile 2022]. [Online]. Available: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

[2] R. Sharma and R. Singh, "Evolution of recommender systems from ancient times to modern era: A survey," *Indian Journal of Science and Technology*, vol. 9, 05 2016, [Online]. doi:/10.17485/ijst/2016/v9i20/88005.

[3] C. Borodescu, "The anatomy of high-performance recommender systems – Part 1," algolia.com, Apr. 2nd 2021, [ultimo accesso in data: 19 Aprile 2022]. [Online]. Available: https://www.algolia.com/blog/ai/the-anatomy-of-high-performance-recommender-systems-part-1/

[4] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 263–272, [Online]. doi:10.1109/ICDM.2008.22.

[5] J. Brownlee, "What is Deep Learning?" machinelearningmastery.com, August 16th 2019, [ultimo accesso in data: 19 Aprile 2022]. [Online]. Available: https://machinelearningmastery.com/what-is-deep-learning/

[6] Torch Contributors, "PYTORCH 1.11.0 Documentation," 2019, [ultimo accesso in data: 14 Aprile 2022]. [Online]. Available: https://pytorch.org/docs/stable/index.html

[7] M. Kula, "Metadata embeddings for user and item cold-start recommendations," in *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.*, ser. CEUR Workshop Proceedings,

T. Bogers and M. Koolen, Eds., vol. 1448. CEUR-WS.org, 2015, pp. 14–21. [Online]. Available: http://ceur-ws.org/Vol-1448/paper4.pdf

[8] Lyst (Maciej Kula), "LightFM 1.16 Documentation," 2016, [esempio utilizzato: https://making.lyst.com/lightfm/docs/examples/movielens_implicit.html; ultimo accesso in data: 11 Aprile 2022]. [Online]. Available: https://making.lyst.com/lightfm/docs/index.html

[9] J. Brownlee, "Difference Between a Batch and an Epoch in a Neural Network," machinelearningmastery.com, July 20th 2018, [ultimo accesso in data: 19 Aprile 2022]. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

[10] ——, "Loss and Loss Functions for Training Deep Learning Neural Networks," machinelearningmastery.com, January 28th 2019, [ultimo accesso in data: 19 Aprile 2022]. [Online]. Available: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

[11] ——, "Understand the Impact of Learning Rate on Neural Network Performance," machinelearningmastery.com, January 25th 2019, [ultimo accesso in data: 19 Aprile 2022]. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[12] M. Qutbuddin, "An Exhaustive List of Methods to Evaluate Recommender Systems," towardsdatascience.com, Apr. 20 2020, [ultimo accesso in data: 19 Aprile 2022]. [Online]. Available: https://towardsdatascience.com/an-exhaustive-list-of-methods-to-evaluate-recommender-systems-a70c05e121de

[13] ShopRunner Data Science Team, "Collie Documentation - version 1.2.0," collie.readthedocs.io, documentation, 2021, [ultimo accesso in data: 7 Aprile 2022]. [Online]. Available: https://collie.readthedocs.io/en/latest/

[14] Z. Meng, R. McCreadie, C. Macdonald, I. Ounis, S. Liu, Y. Wu, X. Wang, S. Liang, Y. Liang, G. Zeng, J. Liang, and Q. Zhang, "BETA-Rec: Build, Evaluate and Tune Automated Recommender Systems," in *Fourteenth ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2020, p. 588–590, ISBN 9781450375832. [Online]. Available:

https://doi.org/10.1145/3383313.3411524

[15] Beta-team, "BetaRecSys Documentation - latest version," beta-recsys.readthedocs.io, documentation, 2020, [ultimo accesso in data: 7 Aprile 2022]. [Online]. Available: https://beta-recsys.readthedocs.io/en/latest/

[16] W. X. Zhao, S. Mu, Y. Hou, Z. Lin, K. Li, Y. Chen, Y. Lu, H. Wang, C. Tian, X. Pan, Y. Min, Z. Feng, X. Fan, X. Chen, P. Wang, W. Ji, Y. Li, X. Wang, and J. Wen, "Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms," *CoRR*, vol. abs/2011.01731, 2020, arXiv:2011.01731. [Online]. Available: https://arxiv.org/abs/2011.01731

[17] Erasmus+ - European Commission, "Content and Language Integrated learning for Key Competences - Erasmus+ programme - Project Reference: 2020-1-IT01-KA226-VET-009021," Erasmus+ Project Results Platform - European Commission website, [ultimo accesso in data: 10 Aprile 2022]. [Online]. Available: https://erasmus-plus.ec.europa.eu/projects/search/details/2020-1-IT01-KA226-VET-009021

[18] Sebastián Ramírez (tiangolo), "FastAPI 0.75.1 Documentation," [ultimo accesso in data: 14 Aprile 2022]. [Online]. Available: https://fastapi.tiangolo.com