POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Un sistema di raccomandazione musicale ibrido

Continuazione automatica di playlist per la "RecSys challenge 2018"

Relatore

Prof. Paolo Garza

Correlatore:

Dott. Enrico Palumbo

Laureando

Enrico Panetta

matricola: 239640

Anno accademico 2017-2018

Sommario

I sistemi di raccomandazione sono diventati un aiuto fondamentale all'utente per la gestione del sovraccarico di informazioni a cui viene sottoposto ed un valido strumento per piattaforme web per mantenere attivo l'interesse dell'utenza, grazie alla capacità di fornire supporto alla decisione personalizzato per ogni utente.

Questa tesi fornisce una panoramica generale sui sistemi di raccomandazione, descrivendo quelle che sono le ragioni storiche che hanno portato al loro sviluppo, oltre alle principali tecniche utilizzate. Inoltre illustra il lavoro nell'ambito della competizione "ACM RecSys Challenge 2018", una competizione basata sullo sviluppo di sistemi di raccomandazione in ambito musicale, con focus sulla continuazione automatica di playlist, durante la quale è stato sviluppato un sistema di raccomandazione ibrido sfruttando un approccio basato sul contenuto, per tentare di risolvere il problema della partenza a freddo, ed un approccio collaborativo. Infine il lavoro da svolto viene confrontato con il lavoro svolto dalle squadre che hanno conquistato la vittoria, analizzando quelli che sono i punti di forza dei loro approcci rispetto alla soluzione presentata in questa tesi.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Con la diffusione di piattaforme di streaming multimediale, l'utilizzo dei sistemi di raccomandazione è diventato il pilastro fondamentale di quella che è un'economia basata sulla fruizione personalizzata di contenuto. L'innovazione di questi sistemi diventa quindi fondamentale per fornire un ausilio di sempre maggiore qualità agli utenti, in un mondo in cui ore di contenuti vengono aggiunte ogni minuto. Dal 2010, la Association for Computer Machinery(ACM) organizza, in collaborazione con varie aziende, una competizione basata sullo sviluppo di sistemi di raccomandazione: la RecSys challenge.

La RecSys challenge 2018, organizzata da Spotify, un popolare servizio di streaming musicale on demand, l'università del Massachusetts, situata ad Amherst, e l'università Johannes Kepler di Linz, si basa su raccomandazioni musicali, nello specifico sulla continuazione automatica di playlist. Per la competizione è stato rilasciato pubblicamente da Spotify un dataset contenente un grande numero di playlist e tracce associate.

Questa tesi presenta una panoramica sui sistemi di raccomandazione ed alcune delle tecniche utilizzate e si concentra sul lavoro effettuato nell'ambito della mia partecipazione alla RecSys challenge 2018, durante la quale ho tentato di sviluppare un sistema che potesse espandere il contenuto di ogni playlist richiesta con un insieme di tracce, basandosi sul contenuto delle playlist e sui metadati delle tracce e delle playlist stesse.

Il capitolo 2 introduce una definizione formale dei sistemi di raccomandazione, i principali approcci, algoritmi e tecniche di valutazione utilizzati. Il capitolo 3 introduce le

specifiche della RecSys challenge 2018, la struttura dei dati forniti e fornisce una panoramica delle tecnologie considerate per l'utilizzo durante la competizione. Il capitolo 4 illustra il processo di implementazione delle componenti del sistema utilizzato, comprese della fase di pre e post processing dei dati. Nel capitolo 5 illustro ed analizzo i risultati e nel capitolo 6 analizzo gli approcci utilizzati dai vincitori della competizione, confrontandoli con quelli da me utilizzati.

Capitolo 2

Stato dell'arte

Lo studio dei sistemi di raccomandazione si è sviluppato come branca indipendente per fornire al consumatore uno strumento di ausilio decisionale, quando posto di fronte al sovraccarico di informazioni contenute in una data piattaforma.

Oggigiorno lo scopo dei sistemi di raccomandazione è fornire raccomandazioni accurate per un grande quantitativo di utenti in poco tempo, di fatto creando una versione personalizzata della piattaforma per ogni utente, avvicinando lo sviluppo dei sistemi di raccomandazione sempre più allo studio dei Big Data[20].

In questo capito introduco la ragione storica dello sviluppo dei sistemi di raccomandazione ed una loro definizione formale, così come i problemi che tentano di risolvere e le metodologie più utilizzate.

2.1 Cenni storici

Lo sviluppo incessante della rete Internet e la sua diffusione su larga scala nell'intorno dei primi anni '90[16], hanno portato allo sviluppo di innumerevoli pagine web sui più disparati argomenti. Prima di questo sviluppo su larga scala, le reti erano limitate ad un contesto locale, accademico o aziendale, dove l'utilizzo primario delle reti era la condivisione di documenti all'interno della località tramite un server centrale interno od il rilascio al pubblico, cioè le poche entità che avevano accesso ad una rete pubblica, di studi di settore.

L'introduzione del protocollo HTTP (HyperText Transfer Protocol), per la trasmissione di informazioni su di una rete, del linguaggio di markup HTML (HyperText Markup Language), per la formattazione del testo e l'impaginazione all'interno della finestra di visualizzazione, e degli URL(Uniform Resource Locator) che, grazie all'introduzione dei DNS(Domain Name System), consentono di individuare risorse all'interno della rete senza dover ricordare indirizzi IP, ma solamente dei nomi, hanno portato allo sviluppo di un gran numero di siti web, ognuno composto da multiple pagine, accessibili non solo a realtà accademiche ed aziendali, ma anche ad un qualunque utente disponesse di una rete a pacchetti.

Nasceva quindi il bisogno, per l'utente medio, di filtrare quella grande quantità di dati per riuscire a trovare le informazioni desiderate. Nascono qui i primi motori di ricerca, che utilizzano tecniche nate durante gli studi di information retrieval(IR), per estrarre fra le innumerevoli pagine web, quelle più rilevanti alla ricerca effettuata, che vengono ordinate secondo alcuni fattori che misurano qualità e rilevanza della pagina web, come può essere il numero di volte che una pagina viene referenziata da un'altra[25].

Con la nascita del commercio online, servizi di streaming di contenuto, e giornali e blog online, i motori di ricerca e le tecniche di information retrieval venivano integrate all'interno di pressocchè ogni sito web e nelle aziende che fornivano contenuti nasceva un nuovo problema: il tempo di permanenza degli utenti. Con l'integrazione dei motori di ricerca, l'attività di un utente medio risultava in:

1. Utente accede al sito.

2. Utente ricerca una risorsa.

3. Utente accede alla risorsa, se trovata.

4. Utente lascia il sito.

La necessità di aumentare il tempo di permanenza degli utenti ha portato allo sviluppo di sistemi in grado, data l'attività dell'utente e l'oggetto a cui l'utente ha fatto attualmente accesso, di fornire una lista di oggetti correlati all'oggetto e/o all'utente. Si è assistito quindi alla comparsa all'interno delle pagine di sezioni del tipo "Video simili",

"Altre notizie sportive", "Musica consigliata". Questi sistemi prendono il nome di "sistemi di raccomandazione".

2.2 Sistemi di raccomandazione

Un sistema di raccomandazione è un componente software utilizzato all'interno di sistemi dinamici distribuiti soggetti ad interazioni utente, solitamente siti di e-commerce o servizi di streaming, per fornire supporto decisionale personalizzato all'utente.

Un sistema di raccomandazione quindi ha lo scopo di "sostituire" l'amico che consiglia un trapano (*rating prediction*) o delle canzoni (*item recommendation*). L'utilità di un sistema di raccomandazione risiede nella capacità di analizzare lo storico delle interazioni dei vari utenti su di una piattaforma e, in base alle caratteristiche degli oggetti con i quali l'utente ha interagito in passato, consigliare altri oggetti che potrebbero soddisfare gli interessi o i bisogni dell'utente, considerando ad esempio:

Storico utente Osservando lo storico utente, oggetti simili a quelli trovati interessanti in passato.

Utenti simili Considerando utenti con gusti simili, oggetti ritenuti interessanti da questi utenti.

Informazioni di contesto Sono informazioni di contorno al contesto di utilizzo del sistema, come informazioni demografiche sugli utenti o anche informazioni temporali. Ad esempio, un sistema di raccomandazione per ristoranti potrebbe consigliare in maniera prioritaria locali con pista da ballo nei weekend o locali che servono aperitivi nel tardo pomeriggio, il cosiddetto "happy hour".

Informazioni di dominio Informazioni specifiche del dominio di applicazione, che permettono un'analisi più approfondita degli oggetti. Ad esempio, un sistema di raccomandazione da utilizzare su di un servizio di streaming video può utilizzare la percentuale di video guardato dall'utente per ricavarne una misura indiretta dell'interesse dell'utente. Richiedono un esperto del dominio di applicazione durante la fase di sviluppo per la definizione delle metriche.

Ciò è necessario, dal lato utente, perchè oggigiorno la quantità di contenuto è tale che ogni utente non può fisicamente processare tutto prima di effettuare la sua scelta, mentre dal lato aziendale, aumentare il tempo di permanenza degli utenti sulla piattaforma comporta un aumento degli introiti.

2.2.1 Il problema delle raccomandazioni

Il problema dello sviluppo di un sistema di raccomandazione si può quindi formalizzare dicendo che un sistema di raccomandazione è un estimatore con lo scopo di stimare una funzione di utilità che[8]:

- Dato lo spazio di tutti gli oggetti O
- Dato lo spazio di tutti gli utenti P
- Misura l'utilità $u : PxO \Rightarrow R$ di un oggetto $o \in O$ per un utente $p \in P$, dove R è un insieme contenente le valutazioni(ratings) date da un utente ad un oggetto.

Utilizzando la funzione di utilità stimata trova per ogni utente l'oggetto(nuovo) $o^{'} \in O$ potenzialmente più utile:

$$\forall p \in P, \qquad o_p^{'} = \underset{o \in O}{\operatorname{argmax}}\, u(o, p). \tag{2.1}$$

Dalla eq. (2.1) è quindi evidente il vantaggio di utilizzare un sistema di raccomandazione al posto o in congiunzione ad un motore di ricerca, in quanto non richiede interazione diretta dell'utente e sopratutto non richiede la conoscenza da parte dell'utente dell'esistenza degli oggetti raccomandati. Un sistema di raccomandazione può quindi consigliare ad un utente oggetti che potrebbero interessargli e di cui non conosceva neanche l'esistenza.

2.3 Tipi di sistemi di raccomandazione

I sistemi di raccomandazione vengono solitamente classificati in base all'approccio utilizzato per la gestione delle relazioni fra utenti ed oggetti[35]:

Basato sul contenuto Il sistema consiglia nuovi oggetti in base a quanto questi sono simili a ciò che l'utente ha ritenuto interessante in passato, andando quindi ad osservare il contenuto degli oggetti stessi.

Collaborativo Il sistema consiglia nuovi oggetti in base a ciò che è stato ritenuto interessante da altri utenti con gusti simili, cioè che hanno ritenuto interessanti gli stessi oggetti o un loro sottoinsieme, osservando quindi solamente le interazioni fra utenti ed oggetti.

Ibrido Una combinazione dei due approcci descritti, nel tentativo di risolvere alcuni dei problemi che affliggono i due approcci qui descritti. Utilizzando un approccio ibrido vengono considerate sia le similarità fra i gusti degli utenti, sia le similarità fra gli oggetti.

Questi sistemi vengono poi ulteriormente classificati in base a come l'interesse per un oggetto viene fornito da un utente:

Feedback esplicito Un sistema che utilizza feedback esplicito richiede un'interazione diretta dell'utente, che deve quindi fornire la sua valutazione riguardo l'oggetto in questione. I casi più diffusi di sistemi con feedback esplicito si possono trovare applicati a piattaforme che consentono agli utenti di scrivere recensioni riguardo ristoranti, film, oggetti acquistati, ecc.

Feedback implicito Un sistema che utilizza feedback implicito non richiede una valutazione dell'oggetto da parte dell'utente, ma estrae i gusti dell'utente dalle azioni compiute sulla piattaforma. Ad esempio, una piattaforma di streaming video può utilizzare come metrica la percentuale di visione di un video come valutazione indiretta di interesse.

I sistemi di raccomandazione portati in produzione ed utilizzati nelle grandi piattaforme, solitamente utilizzano una combinazione dei due tipi di feedback, questo perchè per quanto il feedback esplicito sia più rappresentativo dei gusti effettivi di un utente, la quantità di dati estraibili da feedback implicito è di molto superiore al feedback esplicito, in quanto molti utenti non tendono a lasciare recensioni. Risulterebbe quindi controproducente da parte di un'azienda ignorarlo.

2.4 Approcci basati sul contenuto

Un sistema di raccomandazione basato sul contenuto sfrutta il contenuto degli oggetti con cui l'utente ha precedentemente interagito, ad esempio il contenuto di un documento(articolo, pagina web) o la sua descrizione, per costruire un profilo dei gusti dell'utente. Questo profilo viene quindi utilizzato nel processo di raccomandazione confrontandolo con il contenuto degli oggetti ancora non considerati, ottenendo un metrica di similarità fra l'utente e gli oggetti [24], il cui contenuto indica quanto un oggetto si avvicini ai gusti dell'utente.

L'implementazione di un sistema basato sul contenuto richiede tre componenti fondamentali

Estrazione degli attributi Parte fondamentale di un sistema basato sul contenuto; è il processo di produrre una rappresentazione numerica, quindi processabile matematicamente, degli oggetti a partire dalle loro caratteristiche. Trattandosi di informazioni in formato testuale, nel processo vengono utilizzate tecniche provenienti dall'information retrieval.

Generazione profili Utilizzando gli attributi estratti ed il feedback utente viene costruito un profilo dei gusti utente.

Generazione raccomandazioni Utilizzando i profili utente generati e gli attributi degli oggetti, viene calcolata la similarità fra utente ed oggetti. Gli oggetti con similarità maggiore vengono poi visualizzati all'utente sotto forma di raccomandazione.

Il modello del profilo utente generato viene poi aggiornato con le valutazioni dell'utente gli oggetti raccomandati con cui ha interagito.

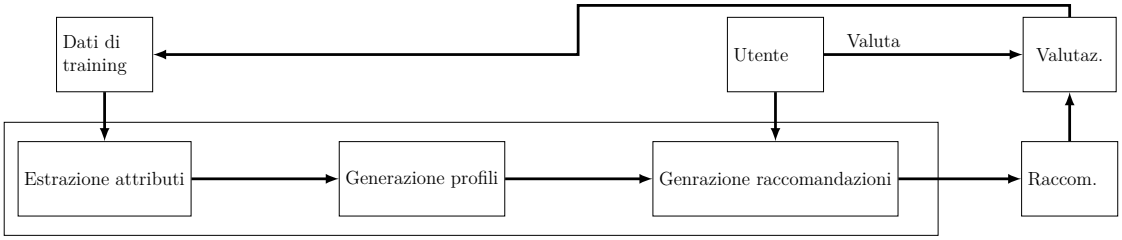

Figura 2.1: Processo di un recommender basato sul contenuto

2.4.1 Estrazione attributi

Nella maggior parte dei casi che richiedono l'utilizzo di sistemi basati sul contenuto gli oggetti sono rappresentati da, appunto, un contenuto o una descrizione ed una serie di parole chiave. Bisogna quindi discernere, fra i termini che rappresentano l'oggetto, quali siano significativi alla creazione di un profilo. Dato quindi l'insieme dei documenti $D = \{d_1, d_2, ..., d_K\}$, ognuno rappresentato un vettore di termini $T = \{t_1, t_2, ..., t_N\}$, l'operazione di estrazione ha lo scopo di fornire per ogni documento d_i un vettore di dimensione N $d_i = \{w_{1,i}, w_{2,i}, ..., w_{N,i}\}$ contenente i pesi $w_{n,i}$ dei termini n all'interno del documento i. Questa rappresentazione dei documenti può quindi essere utilizzata per il calcolo della distanza tra due documenti.

2.4.1.1 Calcolo dei pesi dei termini

L'approccio più comune utilizzato per il calcolo dei pesi consiste nell'utilizzare una misura inizialmente pensata per l'utilizzo nel campo dell'information retrieval, detta TF-IDF(Term Frequency-Inverse Document Frequency). Si tratta di una misura pensata tentando di definire più importanti quei termini che possono essere trovati di frequente in un dato documento ma raramente nel complesso dei documenti. La TF-TDF di un termine i all'interno di un documento j si calcola tramite il prodotto di due misure[28]:

$$\text{TF-IDF}_{t_i,d_j} = \overbrace{\text{TF}_{t_i,d_j}}^{TF} \cdot \overbrace{\log \frac{N}{n_i}}^{\text{IDF}} \tag{2.2}$$

La frequenza del termine (TF, Term Frequency) misura la frequenza del termine i all'interno del documento j e la frequenza inversa del documento misura l'importanza del termine all'interno della totalità dei documenti:

$$\text{TF}_{t_i,d_j} = \frac{f_{i,j}}{\max_k f_{k,j}} \tag{2.3}$$

$$\text{IDF}_{t_i} = \log \frac{N}{n_i} \tag{2.4}$$

Dove $\max_k f_{k,j}$ viene calcolato considerando tutte le frequenze dei termini del documento j, N è il numero totale di documenti ed n_i è il numero di documenti che contengono il termine i. Le TF-IDF così calcolate per tutti i termini del documento vengono poi normalizzate,

per poter essere utilizzate come peso. Considerando TF-IDF$_{d_j}$ il vettore contenente le TF-IDF dei termini associati al documento d_j

$$w_{i,j} = \frac{\text{TF-IDF}_{t_i,d_j}}{\|\text{TF-IDF}_{d_j}\|} \tag{2.5}$$

2.4.2 Calcolo raccomandazione

Il profilo di un utente nei sistemi basati sul contenuto viene espresso anch'esso come un vettore di pesi associati a termini rilevanti. Solitamente il vettore utente viene calcolato tramite una somma pesata dei vettori con i pesi dei documenti ritenuti interessanti, utilizzando come pesi le votazioni, normalizzate, lasciate dall'utente per quel documento. Il vettore risultante viene a sua volta normalizzato.

Essendo quindi i vettori degli utenti e quelli dei documenti comparabili, è possibile calcolare la similarità, cioè il livello di potenziale interesse, di un utente i per un documento j. La misura di similarità più utilizzata è la similarità del coseno:

$$sim(u_i, d_k) = \frac{\sum_k w_{k,i} \cdot w_{k,j}}{\sqrt{\sum_k w_{k,i}^2} \cdot \sqrt{\sum_k w_{k,j}^2}} \tag{2.6}$$

2.4.3 Problemi degli approcci basati sul contenuto

I sistemi basati sul contenuto sono limitati da problemi derivanti dalla difficoltà di analisi del linguaggio scritto e del contenuto stesso degli oggetti. Ogni oggetto può essere descritto soltanto da un numero limitato di termini. L'estrazione dei termini è un punto cruciale in un sistema di raccomandazione di questo tipo; estrazione che si rivela semplice se gli oggetti analizzati sono di tipo testuale, come libri o pagine web, mentre si rivela ardua nel caso di documenti di tipo multimediale, come immagini, video o audio. Nel secondo caso le parole chiave possono essere inserite manualmente oppure ottenute tramite analisi dei metadati o analisi più sofisticate dell'oggetto stesso.

Un'altra limitazione di questi sistemi è quella della collisione tra oggetti; se due documenti sono descritti dallo stesso insieme di termini vengono considerati identici dal sistema, indipendentemente dalla qualità.

Inoltre, come in tutte le branche del machine learning, i sistemi si imbattono nel problema dell'overfitting, infatti non sempre risulta ottimale consigliare ad un utente oggetti

troppo simili a quelli interessanti in passato, anche se l'introduzione di un filtro sulle raccomandazioni in uscita deve essere considerato in base al contesto di utilizzo del sistema. Ad esempio, se un utente di una piattaforma di e-commerce ha effettuato l'acquisto di un trapano risulta inutile da parte del sistema consigliare l'acquisto di ulteriori trapani, mentre può essere utile da parte di una libreria consigliare libri dello stesso genere letterario o autore.

I sistemi di raccomandazione soffrono, inoltre, del problema dell'introduzione di un nuovo utente, per il quale risulta impossibile effettuare delle raccomandazioni finchè non interagisce per un tempo abbastanza lungo con il sistema, in modo da riuscire a raccomandare oggetti con un grado abbastanza alto di confidenza. Molte piattaforme introducono un sondaggio all'atto della registrazione di un nuovo utente per ovviare a questo problema.

2.5 Approcci collaborativi

I sistemi che utilizzano approcci collaborativi, a differenza dei sistemi basati sul contenuto, non tentano di predire la rilevanza di un oggetto per un utente dalle caratteristiche dell'oggetto, ma dalle votazioni che altri utenti con gusti simili hanno dato allo specifico oggetto; cioè l'utilità $u(o, p)$ dell'oggetto o per l'utente p viene calcolata in base alle utilità $u(o, p_i)$ dell'oggetto o assegnate dagli utenti p_i con gusti simili a c[3]. Il problema della raccomandazione diventa quindi il problema della definizione della similarità fra i gusti degli utenti.

I sistemi che utilizzano approccio collaborativo si dividono in due macro-categorie:

Neighbor based I sistemi basati sui vicini conservano l'interezza delle interazioni fra utente ed oggetti e le utilizzano direttamente per effettuare le raccomandazioni. Queste vengono effettuate generalmente con metodi basati sugli utenti o sugli oggetti. I metodi basati sugli utenti predicono la valutazione di un oggetto basandosi sulle valutazioni degli utenti. I metodi basati sugli oggetti invertono invece il problema, cercando di calcolare la valutazione di un utente per un dato oggetto. In entrambi questi approcci due oggetti risultano simili se più utenti hanno dato una valutazione simile.

Model based I metodi basati su modelli, invece, allenano un modello predittivo. Tentando di estrarre proprietà latenti di utenti ed oggetti dalle interazioni fra questi. Il modello, allenato utilizzando i dati disponibili, è poi utilizzato per le raccomandazioni.

2.5.1 Metodi "Neighbor based"

I metodi basati sui vicini hanno lo scopo di rendere automatico il processo sociale della richiesta di consigli a persone con gli stessi gusti. Un sistema di questo tipo, non utilizzando come fonte primaria di informazione parametri latenti, è capace di raccomandare oggetti che non ricadono nella sfera di interesse solita dell'utente. Inoltre l'utilizzo di sistemi basati sui vicini offre una serie di vantaggi[15]:

Semplicità I sistemi basati sui vicini sono semplici da implementare, in quanto il ragionamento utilizzato è abbastanza intuitivo.

Chiarezza Questi sistemi consentono inoltre una chiarezza sulla natura delle raccomandazioni che altri metodi solitamente non hanno. Psicologicamente un utente è portato a fidarsi maggiormente di una raccomandazione se questa è accompagnata da una motivazione per la sua presenza. Mostrare una sezione del tipo "Film piaciuti ad altri utenti a cui è piaciuto 'Altrimenti ci arrabbiamo'" aumenta la comprensione dell'utente sulle raccomandazioni e la fiducia verso il sistema.

Prestazioni I sistemi basati sui vicini non richiedono il training di un modello, operazione molto dispendiosa su una grande quantità di dati e che occorre rifare periodicamente per aggiornare il modello con nuovi utenti ed oggetti. Le similarità tra utenti/oggetti e la lista dei vicini può essere calcolata prima della messa in campo. Inoltre l'aggiunta di una nuova entità richiede soltanto il calcolo di similarità e vicini per quella entità.

2.5.1.1 K-NearestNeighbors per utenti

Un sistema modellato per utilizzare K-NearestNeighbors(K-NN) [14] per utenti rappresenta ogni utente come un vettore $u = r_1, r_2, ..., r_n$ contenente i punteggi(rating) assegnati dall'utente ad ogni oggetto $i \in I$, assegnando automaticamente un valore considerato non valido se il punteggio non è stato fornito. Le similarità fra due utenti u e u' vengono poi

calcolate tramite la similarità del coseno (eq. (2.6)) o la similarità Pearson:

$$sim(u, u') = \frac{\sum_{i \in I_u \cap I_{u'}} (r_{i,u} - \overline{r_u}) \cdot (r_{i,u'} - \overline{r_{u'}})}{\sqrt{\sum_{i \in I_u \cap I_{u'}} (r_{i,u} - \overline{r_u})^2} \cdot \sqrt{\sum_{i \in I_u \cap I_{u'}} (r_{i,u'} - \overline{r_{u'}})^2}}. \tag{2.7}$$

Una volta calcolate le similarità l'algoritmo seleziona per ogni utente u i K utenti u' che hanno similarità maggiore, formando per ogni utente l'insieme S_u contenente gli utenti simili ad u, utilizzando S_u per calcolare la valutazione potenziale che l'utente u darebbe all'oggetto o:

$$r_{u,i} = \overline{r_u} + \frac{\sum_{u' \in S} sim(u, u') \cdot (r_{i,u'} - \overline{r_{u'}})}{\sum_{u' \in S} |sim(u, u')|}. \tag{2.8}$$

2.5.1.2 K-NearestNeighbors per oggetti

L'utilizzo di KNN basando il focus dello studio sugli oggetti è una casistica analoga a quella dell'utilizzo degli utenti. Si tratta del caso in cui la matrice delle interazioni utente-oggetto, costruita utilizzando i vettori utente (sezione 2.5.1.1) come righe, viene trasposta. In questo caso ogni oggetto i viene rappresentato come un vettore $i = r_1, r_2, ..., r_n$ contenete le valutazioni degli utenti e ciò che viene calcolato non è la valutazione potenziale di un oggetto, la valutazione potenziale di un utente rispetto ad oggetti simili. Il calcolo delle similarità (vedi eq. (2.6) e eq. (2.7)), così come il calcolo della valutazione potenziale dell'utente (vedi eq. (2.8)) sono analoghi al caso di KNN per utente (vedi sezione 2.5.1.1).

2.5.1.3 K-Nearest Neighbors per utenti vs K-Nearest Neighbors per oggetti

Scegliere tra l'utilizzo di KNN per utenti e KNN per oggetti richiede la considerazione di alcuni punti fondamentali[33], tra cui:

Confidenza La confidenza nelle raccomandazioni fornite è una metrica basata sulla valutazione della quantità di valutazioni utilizzate durante il calcolo delle similarità. Più grande è la quantità di valutazioni utilizzate, maggiore è la confidenza. Tra utenti ed oggetti, si vuole quindi scegliere il caso che porta ad avere un numero medio di valutazioni per similarità più alto.

Efficienza Durante la scelta del metodo da utilizzare è importante considerare la quantità di memoria utilizzata per la conservazione delle similarità e dei vicini, insieme al

tempo utilizzato per il calcolo delle similarità stesse. In applicazioni reali, l'approccio utilizzato è quello per oggetti, in quanto il loro numero è generalmente di molto inferiore quello degli utenti. Inoltre ogni utente può realisticamente fornire valutazioni solamente su un piccolo sottoinsieme di oggetti, per cui è possibile ridurre la quantità di valutazioni salvate soltanto a quelle effettivamente fornite. La scelta dell'approccio si riduce quindi alla scelta del caso che porta un numero minore di celle all'interno della matrice delle interazioni.

Mutabilità Un fattore da considerare durante la scelta riguarda la frequenza con cui avvengono cambi agli insiemi degli utenti e degli oggetti. Dato che gli utenti in un sistema aumentano tipicamente ad un ritmo di molto superiore agli oggetti, utilizzare un approccio basato su questi ultimi può essere vantaggioso, in quanto le nuove similarità possono essere calcolate in un secondo tempo senza inficiare le raccomandazioni ai nuovi utenti. Lo stesso vale per un sistema in cui gli oggetti aumentano ad un ritmo maggiore rispetto agli utenti.

Eterogeneità Utilizzando KNN per oggetti, il sistema tende a raccomandare oggetti che sono simili fra di loro, quindi le raccomandazioni tendono a concentrarsi intorno ad una data categoria. Nell'utilizzo di KNN per utenti, invece, utilizza similarità fra utenti, che vengono quindi calcolate in base all'interezza dello spazio degli oggetti. Per questo motivo le raccomandazioni fornite da KNN per utenti tendono ad essere più eterogenee, il che può essere un vantaggio in quei casi in cui si vuole che il sistema possa fornire raccomandazioni che possano discostarsi dai gusti tipici dell'utente.

2.5.2 Metodi basati su fattorizzazione di matrici

I metodi basati su fattorizzazione di matrici (MF, Matrix Factorization) sono diventati sempre più diffusi grazie alla capacità di rivelare informazioni latenti, partendo dalla matrice delle interazioni utente-oggetto, ed alla loro scalabilità ed accuratezza delle predizioni.

Le matrici delle interazioni sono generalmente una rappresentazione esageratamente specifica dei gusti degli utenti, il che porta ad un tempo di computazione delle similarità, nel caso di metodi basati sui vicini, non indifferenti. Inoltre in sistemi complessi si trova

il problema della sinonimità degli oggetti. Ad esempio, un utente che ha fornito una valutazione positiva per "La Divina Commedia" e per "L'Inferno" ha lasciato sostanzialmente una valutazione positiva per lo stesso oggetto.

Lo scopo dei metodi basati su MF è fornire una rappresentazione più compatta dei dati, dividendo la matrice delle interazioni in matrici di dimensione minore, che quando moltiplicate restituiscono la matrice stessa, per poi ridurne la dimensionalità, ricavando una versione approssimata, ma comunque accurata, della matrice di partenza. Utilizzando la rappresentazione dei fattori sia gli utenti che gli oggetti vengono raggruppati in categorie e interazioni fra utenti ed oggetti diventano concetti più generali.

2.5.2.1 Singular Value Decomposition

La decomposizione a valori singolari (SVD, Singular Value Decomposition) è una tecnica utilizzata in algebra lineare per la decomposizione di matrici, cercando di trasformare lo spazio delle interazioni in uno spazio di valori latenti che possono misurare ad esempio, in un sistema per raccomandazioni musicali, quanto un brano si avvicina ad un dato genere o quanto è apprezzato da una data fascia demografica. Lo spazio dei valori latenti, però, raramente divide gli oggetti in maniera così chiara e per riuscire ad avere più chiarezza nelle raccomandazioni, queste richiedono ulteriori analisi a posteriori.

L'obiettivo della SVD è, data la matrice delle interazioni $A \in R^{m \times n}$, scomporla in modo tale da poter essere scritta come [21]:

$$A = U \Sigma V^T \tag{2.9}$$

Dove:

- $U \in R^{m \times m}$ è la matrice contenente le rappresentazioni degli utenti nello spazio degli attributi latenti. Le colonne di questa matrice possono essere considerate come categorie di utenti. I valori delle righe rappresentano quindi quanto dei gusti dell'utente ricadono nella data categoria.
- $V \in R^{n \times n}$ è la matrice contenente le rappresentazioni degli oggetti nello spazio degli attributi latenti. Le righe di questa matrice possono essere considerate come categorie di oggetti. I valori delle colonne rappresentano la parte delle qualità dell'oggetto che rientra in una data categoria.

- $\Sigma \in R^{m \times n}$ è una matrice diagonale contenente gli autovalori della matrice A, arrangiati in modo tale da essere tutti maggiori di 0 ed in ordine decrescente. I valori di questa matrice possono essere considerati come la rappresentazione di quanto una data categoria di utenti apprezza una data categoria di oggetti.

Una volta calcolata la fattorizzazione, è possibile ridurre il rango delle matrici ignorando le righe e colonne della matrice Σ, ignorando anche le colonne corrispondenti di U e le righe corrispondenti di V, corrispondenti agli autovalori con valore minore, in modo tale da ridurre la matrice ad un dato rango k oppure considerando soltanto gli autovalori maggiori di un dato valore x. Le matrici ottenute alla fine di questo processo sono una rappresentazione approssimata delle interazioni originali; rappresentazione che però non porta una grande perdita di informazioni dato che il rango viene ridotto ignorando le righe corrispondenti agli autovalori con più basso valore, cioè ignorando quelle informazioni che possono essere considerate superflue in quanto la loro conoscenza non porta un grande valore aggiunto alle raccomandazioni.

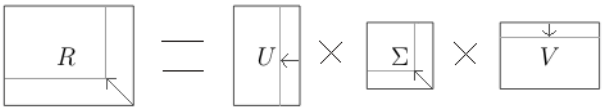

Figura 2.2: Visualizzazione SVD

2.5.2.1.1 Problemi con Singular Value Decomposition L'utilizzo della SVD per la produzione di raccomandazioni si è complicata dal fatto che la matrice delle interazioni è generalmente una matrice sparsa. Si presenta quindi il problema del riempimento dei valori mancanti. Un altro problema nell'utilizzo dell SVD si trova della ricerca del rango ottimale k a cui troncare le matrici. La fattorizzazione di una matrice è un processo computazionalmente pesante $O(righe^2 \times colonne + colonne^3)$. Occorre quindi un metodo per riuscire a ridurre il tempo di creazione del modello.

2.5.2.2 Gradient Descent

L'algoritmo Gradient Descent è probabilmente l'algoritmo di ottimizzazione più utilizzato nel campo del machine learning. L'algoritmo Gradient Descent, così come i suoi derivati, ha

come scopo la minimizzazione di una funzione obiettivo $J(\theta)$ caratterizzata da un numero p di parametri tale che $\theta \in R^p$, aggiornando i parametri θ nella direzione opposta a quella del gradiente della funzione obiettivo $\nabla_\theta J(\theta)$[31].

L'utilizzo di questo algoritmo di ottimizzazione in congiunzione alla SVD consente di ridurre il carico computazionale fermando l'algoritmo di fattorizzazione quando vengono trovate le k dimensioni ottimali, quindi quando si è calcolata la k-esima riga della matrice Σ. Il criterio utilizzato per trovare l'approssimazione migliore, cioè la funzione $J(\theta)$ utilizzata, è l'errore quadratico medio (eq. (2.10)) globale. Quindi, calcolando una dimensione della SVD alla volta e fermando il calcolo della dimensioni quando viene raggiunto un minimo locale di questa funzione $J(\theta)$, si aumenta notevolmente la velocità dell'algoritmo, che non deve calcolare completamente i fattori della matrice delle interazioni.

Lo svantaggio di utilizzare questa tecnica è il fatto che le matrici U,Σ e V non sono più una vera SVD, in quanto le dimensioni trovate non sono più ortogonali.

2.5.2.3 Fattorizzazione di matrici non-negativa

La fattorizzazione di matrici non-negativa(NNMF, Non Negative Matrix Factorization) è una famiglia di algoritmi per la fattorizzazione di matrici utilizzata quando si ha a disposizione una matrice contenente solamente valori positivi e la si vuole fattorizzare in maniera tale da ottenere solamente matrici positive. La NNMF, infatti, data una matrice $X = [x_1, x_2, ..., x_n] \in R_{>0}^{m \times n}$, dove x_i è un vettore n-dimensionale, tenta di scomporre X in una matrice non negativa $U = [u_1, u_2, ..., u_l] \in R_{>0}^{m \times l}$ ed una matrice non negativa $V = [v_1, v_2, ..., v_n] \in R_{>0}^{l \times m}$, con $L \ll \min(M, N)$, in modo tale che $X \approx UV$[23].

Da questa rappresentazioni si può intuire come la NNMF generi una proiezione dello spazio delle interazioni utente-oggetto in uno spazio di dimensionalità molto minore, generando quindi un'approssimazione dello spazio originale. Questa approssimazione è efficiente solamente se l'algoritmo riesce a rappresentare nella matrice U gli attributi latenti. mentre la matrice V rappresenta i pesi attribuiti agli attributi contenuti nella matrice U.

2.5.3 Problemi degli approcci collaborativi

L'utilizzo di approcci collaborativi per la progettazione di un sistema di raccomandazione offre alcuni vantaggi rispetto ad un approccio basato sul contenuto. Il beneficio principale rispetto ad un approccio basato sul contenuto è che, utilizzando solamente le informazioni sulle interazioni fra utenti ed oggetti, possiedono un'indipendenza dalla natura effettiva degli oggetti, il che consente ad un sistema collaborativo di fornire raccomandazioni su oggetti anche molto eterogenei fra di loro, risultando ideale, ad esempio, nel caso di una piattaforma di e-commerce, che deve consigliare al cliente oggetti molto diversi fra di loro.

L'utilizzo di un approccio collaborativo ha, però, anch'esso alcune limitazioni sulle raccomandazioni che è possibile effettuare[3].

L'aggiunta di nuovi utenti al sistema causa, così come nei sistemi basati sul contenuto, l'impossibilità ad effettuare raccomandazioni finchè non sono note le sue preferenze, cioè finchè l'utente non interagisce con il sistema per un periodo abbastanza lungo. Una soluzione a questo problema può essere l'aggiunta di un sondaggio al momento della creazione di un nuovo utente, in modo tale da conoscere fin da subito una parte dei suoi gusti. Un'altra soluzione può essere cambiare completamente approccio, utilizzando un approccio ibrido per la creazione del sistema di raccomandazione, oppure semplicemente consigliare all'utente gli oggetti più popolari, almeno inizialmente.

L'aggiunta di un nuovo oggetto è anch'essa problematica per un sistema collaborativo, in quanto questi considerano solamente le interazioni utente come criterio per effettuare le raccomandazioni, il che impedisce di consigliare l'oggetto finchè questo non è stato valutato da un numero sufficiente di persone. Una soluzione al problema è data dall'utilizzo di soluzioni ibride.

Il problema maggiore che si incontra durante l'utilizzo di un approccio collaborativo è la sparsità della matrice delle interazioni, cioè l'avere un numero di valutazioni disponibili di molto inferiore a quello delle valutazioni da predire. Un sistema collaborativo per funzionare in maniera efficiente richiede, quindi, un numero significativo di utenti. Un oggetto con una valutazione media molto alta data da pochi utenti, infatti, verrà consigliato più raramente di un oggetto con valutazione media discreta, ma data da molti utenti. Una soluzione a questo problema può essere quella di utilizzare nel calcolo delle similarità fra

utenti anche informazioni degli utenti stessi come, ad esempio, informazioni demografiche, andando quindi a ricadere ancora una volta nell'utilizzo di un approccio ibrido.

2.6 Approcci ibridi

Alcune delle limitazioni degli approcci basati sul contenuto e collaborativi (sezione 2.4.3 e sezione 2.5.3) possono essere risolte dall'utilizzo di un approccio ibrido. Solitamente l'utilizzo di un approccio ibrido consiste nel combinare tecniche basate sul contenuto e collaborative al fine di tentare di ovviare alle lacune rispettive di ogni approccio.

Esistono diversi metodi per creare un sistema di raccomandazione ibrido, che si possono sostanzialmente dividere in alcune macro-categorie[7]:

Misto/Switched L'utilizzo di questo tipo di approccio è concettualmente il più semplice da implementare, in quanto consiste nell'utilizzo in maniera indipendente di sistemi basati sul contenuto e collaborativi. I due sistemi così generati operano in maniera indipendente e forniscono raccomandazioni separate. Il problema quindi si sposta sulla combinazione dei risultati forniti. Un approccio può essere quello di calcolare la predizione finale come combinazione delle due predizioni oppure utilizzare il risultato del sistema con più confidenza o ancora impostare un approccio preferenziale ed utilizzare l'altro quando il primo è impossibilitato a fornire la raccomandazione richiesta.

Aumento delle feature Consiste nell'aggiunta di predizioni o classificazioni definite da una tecnica nella fase di training della seconda. Un metodo può essere quello di conservare insieme alle valutazioni fornite da un utente per un oggetto, anche i vettori basati sul contenuto degli oggetti. Questi vettori possono essere utilizzati, ad esempio, per agire come utenti artificiali, quelli che vengono definiti *filterbot*, all'interno di sistemi collaborativi.

Incremento delle feature Consiste nell'utilizzo di tecniche tipiche dei sistemi collaborativi all'interno di sistemi basati sul contenuto. Tipicamente si tende ad utilizzare tecniche di fattorizzazione di matrici ai fini di ridurre la dimensionalità su gruppi

di utenti, in modo da utilizzare le informazioni raccolte sullo spazio degli attributi latenti come feature aggiuntive da utilizzare in un sistema basato sul contenuto.

Creazione di un modello ex novo Creare un modello ex novo da completa libertà nello sviluppo del sistema di raccomandazione. È possibile quindi utilizzare un sistema unico per predire le valutazioni potenziali degli oggetti sconosciuti utilizzando informazioni sulla similarità fra utenti, fra oggetti ed informazioni collaborative. Inoltre creando un sistema ibrido ad hoc, è possibile utilizzare informazioni di dominio e tecniche per sistemi di raccomandazione basati sulla conoscenza del dominio, che però richiedono conoscenza umana del dominio di applicazione del sistema, quindi esperti del dominio di applicazione come consulenti durante la fase di sviluppo[6].

2.7 Valutazione dei sistemi di raccomandazione

Lo sviluppo di un sistema di raccomandazione richiede l'integrazione di un una fase di valutazione. La valutazione, idealmente, è parte integrante dell'intero processo di sviluppo, in quanto aiuta a discernere l'approccio, o gli approcci, da utilizzare, aiuta l'impostazione dei parametri degli algoritmi e la scelta degli stessi.

Valutare un sistema di raccomandazione significa considerare non solo fattori numerici, quali possono essere misure di precisione o di confidenza o la scalabilità, ma considerare anche fattori di natura più empirica, come la fiducia degli utenti nelle raccomandazioni fornite e la preferenza degli utenti quando posti di fronte a diversi tipi di raccomandazioni.

Le principali proprietà da considerare quando si valuta un sistema di raccomandazione sono[34]:

Preferenza utente/Fiducia Durante la scelta fra diversi sistemi di raccomandazione bisogna considerare anche la soddisfazione degli utenti nelle raccomandazioni fornite. La scelta del campione di utenti da utilizzare è anch'esso importante e legato al contesto di utilizzo del sistema e l'interpretazione delle preferenze utente. Solitamente, quindi si tende a dividere le preferenze utente in più punti. La fiducia, invece è il livello di credibilità del il sistema di raccomandazione agli occhi degli utenti. Non è possibile definire una misura definitiva per la fiducia, ma è possibile fare delle assunzioni sul

comportamento degli utenti. Ad esempio si può assumere un alto livello di fiducia in un utente che utilizza spesso il sistema.

Accuratezza Probabilmente la classe di metriche più importante e più utilizzata, in quanto utilizzata durante la fase di impostazione dei parametri degli algoritmi e di scelta fra questi. La scelta del tipo di misura da utilizzare ricade solitamente sul tipo di dati disponibili, cioè sul tipo di feedback utente:

Feedback esplicito Nel caso di feedback esplicito le misure utilizzate solitamente sono quelle che forniscono metriche sull'accuratezza delle precisioni, che quindi forniscono informazioni su quanto il sistema di raccomandazione riesce a prevedere bene le valutazioni utente.

Feedback implicito Nel caso di feedback implicito sono preferibili metriche di supporto alle decisioni, che quindi forniscono informazioni su quanto valide sono i consigli che il sistema presenta all'utente.

Copertura In sistemi con un grande quantitativo di oggetti, è importante considerare anche quanto dello spazio degli oggetti il sistema effettivamente viene considerato. È importante non fornisca solamente predizioni accurate su un sottoinsieme degli oggetti, ma che riesca a raccomandare anche oggetti poco comuni ed oggetti nuovi per l'utente.

Confidenza La confidenza è una misura della sicurezza che il sistema ha nel fornire le raccomandazioni. È una misura utile tanto nella fase di sviluppo, quanto nella fase di messa in campo, in quanto si possono escludere dalla lista delle raccomandazioni quelle con un livello di confidenza inferiore ad una certa soglia.

Capacità di adattamento Se un sistema di raccomandazione è impiegato in un contesto in cui lo spazio degli oggetti varia di frequente, è importante che il sistema si adatti a questi cambiamenti. Un esempio può essere quello di una piattaforma di abbigliamento, ambito in cui le mode cambiano di frequente e nuovi capi vengono spesso introdotti, o quello di una piattaforma che consiglia notizie durante un evento imprevisto, quale può essere un disastro naturale o un attentato terroristico, durante il quale l'interesse degli utenti si sposta drasticamente.

Scalabilità I sistemi di raccomandazione sono progettati per fornire aiuto all'utente quando posti di fronte a grandi quantità di oggetti. Alcuni sistemi, per mantenere bassi i tempi di produzione delle raccomandazioni, tendono a trascurare altre proprietà come la confidenza, la precisione o la copertura. La scalabilità viene solitamente misurata sperimentando con vari dataset di dimensione crescente.

2.7.1 Metriche con feedback esplicito

In sistemi in cui la preferenza utente è misurata su una scala, vengono impiegati sistemi di raccomandazione che tentano di predire la valutazione di un utente per un oggetto sconosciuto. Si vuole quindi misurare l'errore della valutazione predetta rispetto a quella reale.

2.7.1.0.1 Root Mean Square Error (RMSE) Si tratta della misura più popolare di errore nella predizione di feedback esplicito. Dato l'insieme delle predizioni P, dove ogni elemento $p = (u, i)$ è una coppia utente-oggetto, per ogni coppia si confrontano la valutazione reale $\hat{r}_{u,i}$ e quella predetta $r_{u,i}$.

$$\text{RMSE} = \sqrt{\frac{1}{|P|} \sum_{(u,i) \in P} (\hat{r}_{u,i} - r_{u,i})^2}. \tag{2.10}$$

L'elevamento al quadrato dell'errore consente di dare più peso ad errori grandi e meno peso ad errori piccoli.

2.7.2 Metriche con feedback implicito

In sistemi in cui la preferenza utente è misurata tramite feedback implicito oppure con metodi unari, quali può essere l'aggiunta di un oggetto ad una lista dei preferiti, vengono impiegati tipicamente strumenti tipici del supporto alle decisioni.

2.7.2.0.1 Precision Si tratta della percentuale di elementi selezionati che sono anche rilevanti. Tipicamente ottimizzata se lo scopo del sistema di raccomandazione è consigliare gli oggetti più utili.

$$\text{Precision} = \frac{N_{r,s}}{N_s} \tag{2.11}$$

2.7.2.0.2 Recall Si tratta della percentuale di oggetti rilevanti che sono stati selezionati. Tipicamente ottimizzata se lo scopo del sistema è evitare le sviste ed evitare di non consigliare all'utente oggetti utili.

$$\text{Recall} = \frac{N_{r,s}}{N_r} \tag{2.12}$$

2.7.2.0.3 Curva ROC Si tratta di un grafico in cui si rappresentano la percentuale di falsi positivi e veri positivi al variare dei parametri dell'algoritmo. Si tratta quindi del grafico del compromesso che si ha in termini di errore delle raccomandazioni al variare dei parametri dell'algoritmo. L'area sotto la curva rappresenta la sensibilità dell'algoritmo al variare dei parametri. La curva ROC ideale è un gradino.

Capitolo 3

Progettazione

La RecSys challenge 2018, organizzata da Spotify, l'Università del Massachusetts, Amherst, e l'Università Johannes Kepler di Linz, è una competizione internazionale che ha come focus i sistemi di raccomandazione musicali. La challenge si concentra sulla continuazione automatica delle playlist, con obiettivo lo sviluppo un sistema che, dato un insieme di proprietà di una playlist, generi un insieme di nuove tracce correlate, di fatto continuandola[9].

In questo capitolo introduco la challenge ed illustro le tecnologie considerate per l'utilizzo.

3.1 Specifiche della challenge

Per la RecSys challenge 2018, Spotify ha costruito il Million Playlist Dataset (MPD), un dataset composto da 1.000.000 di playlist create da utenti del servizio. Il dataset è reso disponibile nella sua interezza ai ricercatori accademici, mentre i ricercatori privati hanno avuto la possiblità di utilizzare un suo sottoinsieme[9].

La competizione è stata divisa in due competizioni parallele:

Main track I partecipanti alla main track della challenge potevano considerare soltanto le informazioni fornite nel dataset per effettuare il training degli algoritmi e le raccomandazioni.

Creative track I partecipanti alla creative track della challenge erano autorizzati ad includere dati addizionali per lo sviluppo dei sistemi di raccomandazione, purchè questi dati siano pubblicamente disponibili e ne fosse specificata la fonte e il metodo di utilizzo

La parte di competizione a cui ho preso parte è la Main track.

3.1.1 Million Playlist Dataset

Il Million Playlist Dataset è stato generato estraendo 1.000.000 playlist dai miliardi di playlist create dagli utenti di Spotify, in maniera casuale, fra tutte le playlist che soddisfacessero una serie di criteri[26]:

- La playlist deve contenere almeno cinque tracce ed un massimo di 250 tracce, create da almeno tre artisti diversi ed appartenenti ad almeno due album diversi. Inoltre non vengono considerate playlist contenenti tracce non appartenenti alla libreria di Spotify, ma inserite perchè presenti nel dispositivo dell'utente e playlist seguite soltanto dal creatore.
- La playlist deve essere stata creata da un cittadino almeno tredicenne degli Stati Uniti d'America e deve aver avuto visibilità pubblica al momento della creazione del Million Playlist Dataset.
- La playlist non deve avere titolo o descrizione offensive o potenzialmente offensive. La playlist è esclusa anche se il titolo presenta temi per adulti ed è stata creata da un minore di 18 anni.

Alcune playlist fittizie sono state inoltre aggiunte, per identificare utilizzi impropri del dataset.

3.1.1.1 Struttra del MPD

Il Million Playlist Dataset è stato distribuito come archivio `.tar.gz` con una dimensione di circa 5GB contenente alcuni script di servizio scritti in linguaggio `python` ed i dati, divisi in 1000 file json, ognuno contenente 1000 playlist, per una dimensione totale dopo l'estrazione

di oltre 32GB. Ogni porzione del dataset, detta `slice`, utilizza la seguente convenzione per la nomenclatura del file:

```
mpd.slice.ID_PRIMA_PLAYLIST-ID_ULTIMA_PLAYLIST.json
```

Il dataset contiene in totale un milione di playlist ed oltre due milioni di tracce uniche, oltre 700.000 album unici e quasi 300.000 artisti unici.

Ogni file contiene un dizionario json contenente informazioni sullo slice, trascurabili dal punto di vista della computazione, una lista di playlist contenente vari metadati sulla playlist stessa e, per ogni playlist, una lista delle tracce associate con i relativi metadati.

I metadati delle playlist comprendono informazioni come il nome della playlist, l'identificativo all'interno del dataset, statistiche sulle tracce contenute, come il numero di artisti ed album, il numero di playlist contenute ed altre informazioni (appendice A).

I metadati delle tracce comprendono informazioni generali sulla traccia, come gli Uniform Resource Identifier(URI [5]) di traccia, album ed artista, i nomi di traccia, album ed artista e la durata della traccia, e la posizione della traccia all'interno della playlist.

3.1.2 Million Playlist Challenge Set

Il challenge set consiste in un singolo file json al cui interno sono contenute 10.000 playlist. Lo scopo della challenge è fornire 500 raccomandazioni per ogni playlist contenuta in questo file. Il challenge set è stato generato partendo da playlist che soddisfano gli stessi criteri di quelle contenute nel Million Playlist Dataset (vedi sezione 3.1.1) ed estraendole in maniera casuale, con delle restrizioni aggiuntive: tutte le tracce presenti nel challenge set, comprese quelle omesse, devono comparire anche nel MPD.

3.1.2.1 Categorie di challenge

Le 10.000 playlist che compongono il challenge set possono essere divise in 10 categorie differenti, ognuna composta da 1000 playlist, per i quali sono identificabili le seguenti tipologie:

- Predizione di tracce per playlist di cui solo il titolo è noto oppure è noto il titolo e la prima traccia.

- Predizione di tracce per playlist di cui sono noti il nome e le prime cinque tracce oppure soltanto le prime cinque tracce.
- Predizione di tracce per playlist di cui sono noti il nome e le prime dieci tracce oppure le prime cinque tracce.
- Predizione di tracce per playlist di cui sono noti il nome e le prime 25 tracce oppure il nome e 25 tracce estratte casualmente.
- Predizione di tracce per playlist di cui sono noti il nome e le prime 100 tracce oppure il nome e 100 tracce estratte casualmente.

Queste categorie possono essere di fatto ridotte in soltanto due tipologie di sfida, in base al numero di tracce note:

0-1 tracce In questa categoria di sfide, gli elementi utili per identificare delle playlist simili sono principalmente gli attributi della playlist, per pochi che siano nel challenge set. Le tracce contenute nelle playlist hanno un ruolo quasi marginale in questo caso.

5-10-25-100 tracce In questa categoria di sfide gli elementi utili sono le tracce contenute nelle playlist. Gli attributi delle playlist ricoprono un ruolo marginale.

3.1.2.2 Struttura del Challenge Set

Il challenge set è composto di un unico file json(appendice B), al cui interno sono contenute informazioni sul file stesso, come la versione e la data di creazione e tutte le 10.000 playlist della sfida. Ogni playlist contiene metadati riguardanti la sfida, come il numero di tracce omesse, il numero di tracce totali ed il numero di tracce seme, il nome della playlist e l'insieme delle tracce seme con i relativi metadati, gli stessi che possono essere trovati nel dataset originale(vedi sezione 3.1.1.1).

3.1.3 Valutazione delle sottomissioni

Le soluzioni alla challenge vengono sottomesse al sistema di valutazione di Spotify tramite la sezione contenente il profilo utente del sito internet della RecSys challenge 2018. Il sistema consente la sottomissione di una sola soluzione al giorno per gruppo.

Un file, per avere validità ed essere sottoposto a valutazione deve rispettare i seguenti criteri:

- Le righe valide devono avere il formato descritto in sezione 4.3.2.1
- Le raccomandazioni di ogni playlist non devono contenere ne le tracce seme fornite per la playlist stessa, ne tracce duplicate.
- Le tracce raccomandate per ogni playlist devono ammontare ad esattamente 500.
- È possibile per il file contenere righe vuote o righe commentate. Una righe commentata deve iniziare con il carattere '#'. Queste righe vengono ignorate in fase di valutazione.

Ogni file che non rispetti uno qualsiasi di questi criteri viene ignorato e non valutato.

Se un file sottomesso al sistema risulta valido, viene valutato considerando solamente il 50% del challenge set fino alla chiusura del sistema, dopo cui il punteggio dell'ultima sottomissione viene ricalcolato utilizzando l'interezza del challenge set. Per ogni playlist contenuta nel file, considerando G l'insieme delle tracce omesse(ground truth) e definendo come $|\cdot|$ l'operatore che ritorna il numero di elementi contenuti in un insieme, vengono calcolate le seguenti metriche[26]:

R-precision La metrica R-precision consiste nel rapporto tra il numero di tracce raccomandate che sono anche rilevanti ed il numero di tracce rilevanti, cioè il numero di tracce omesse.

$$\text{R-precision} = \frac{\left|G \cap R_{1:|G|}\right|}{|G|}. \tag{3.1}$$

Si tratta di una metrica che tiene conto del numero totale di tracce raccomandate e rilevanti, a prescindere dall'ordine. Questa metrica viene mediata su tutte le playlist del file.

NDCG La metrica NDCG, Normalized Discounted Cumulative Gain, o Normalized DCG è basata sul calcolo del Discounted Cumulative Gain o DCG, che misura la qualità dell'ordinamento delle tracce raccomandate e del DCG ideale, cioè il DCG che si otterrebbe nel caso in cui le tracce fossero perfettamente ordinate. Il DCG aumenta

quando le tracce rilevanti sono piazzate prima nella lista di raccomandazioni:

$$DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2(i+1)}. \tag{3.2}$$

Il DCG ideale, o IDCG viene calcolato come:

$$IDCG = 1 + \sum_{i=2}^{|G|} \frac{1}{\log_2(i+1)}. \tag{3.3}$$

Infine il NDCG viene calcolato rapportando le due misure calcolate precedentemente:

$$NDCG = \frac{DCG}{IDCG}. \tag{3.4}$$

Recommended Songs clicks La metrica Recommended Songs clicks, o semplicemente click, è basata su un servizio fornito da Spotify (Recommended Songs) che, dato un insieme di tracce in una playlist, fornisce un insieme di dieci playlist che potrebbero essere aggiunte alla playlist. Nuove dieci playlist possono essere richieste al sistema. I click consistono nel numero di richieste effettuate al sistema prima che una traccia rilevante venga raccomandata e vengono calcolati come segue:

$$\text{clicks} = \left\lfloor \frac{\arg\min_i\{R_i : R_i \in G|\} - 1}{10} \right\rfloor \tag{3.5}$$

Se non vengono raccomandate tracce rilevanti, alla metrica viene assegnato il valore 51, pari al massimo numero di click possibili incrementato di uno. Per l'utilizzo nei grafici, insieme a NDCG e R-precision, ho utilizzato una versione "normalizzata" della metrica:

$$\text{N-Clicks} = \frac{\text{clicks}}{51}. \tag{3.6}$$

3.1.3.1 Aggregazione delle metriche

Le metriche vengono calcolate dal sistema di sottomissione di Spotify una volta al giorno e vengono tabulate in modo tale che ogni metrica abbia una sua classifica. Le tre classifiche vengono poi aggregate utilizzando la tecnica nota come `Borda count` per la generazione della classifica totale. Per ogni metrica viene assegnato un punteggio, corrispondente al numero di partecipanti che si trovano in posizione più bassa in classifica rispetto all'utente considerato più uno.

Il punteggio utilizzato per la classifica finale viene calcolato sommando i punteggi così derivati. In caso di parità viene considerato vincente il partecipante che ha ottenuto un numero di posizioni più alte in classifica maggiore.

3.2 Tecnologie

Le due tipologie principali di sfida proposte richiedono l'utilizzo di sistemi di raccomandazione specializzati principalmente sul contenuto (CB, dall'inglese Content Based) e sul filtraggio collaborativo (CF, dall'inglese Collaborative Filtering). Sono state quindi considerate diverse librerie per sistemi di raccomandazione e linguaggi di programmazione da utilizzare per una o entrambe le tipologie di sfida.

3.2.0.0.1 Specifiche tecniche hardware Le prove basate sull'utilizzo di MyMediaLite sono state effettuate su un server con Ubuntu Linux versione 16.04, processore Intel(R) Xeon(R) X5650 e 32GB di memoria RAM. Le prove basate sull'utilizzo di Apache Spark sono state effettuate sul cluster condiviso del laboratorio interdisciplinare BigData@Polito.

3.2.1 Libreria Surprise

Surprise [19] è una libreria sviluppata come modulo aggiuntivo per SciPy, un ecosistema matematico per il linguaggio Python, avente come focus la costruzione ed analisi di sistemi di raccomandazione basati su CF. Mette a disposizione vari algoritmi per la messa a punto di sistemi di raccomandazione, la maggior parte dei quali utilizzano feedback esplicito, tra cui:

NormalPredictor Questo algoritmo assegna come predizione una valutazione casuale basandosi sulla distribuzione delle valutazioni del training set, assumendo che le valutazioni abbiano una distribuzione normale.

BaselineOnly Questo algoritmo assegna come predizione la media delle predizione a cui vengono sommati i bias di utente ed oggetto.

KNNBasic Implementazione di base dell'algoritmo K-NN. Può essere impostato per predire la valutazione di un oggetto per un utente utilizzando diverse misure di similarità, che possono essere calcolate fra utenti o fra oggetti. Sono inoltre presenti alcune estensioni: `KNNWithMeans`, che prende in considerazione nel calcolo della valutazione anche la valutazione media data da un utente agli oggetti, `KNNWithBaseline`, che prende in considerazione la baseline per ogni utente e `KNNWithZScore`, un'estensione di `KNNWithMeans` che prende in considerazione nel calcolo della valutazione anche la standardizzazione delle valutazioni di ogni utente.

SVD Implementazione dell'algoritmo SVD(Singular Value Decomposition, vedi sezione 2.5.2.1) per la fattorizzazione di matrici, popolarizzata durante la competizione "Netflix Prize" del 2006, ed una sua estensione: `SVD++`, che consente di utilizzare anche feedback implicito.

NMF Algoritmo per CF basato sulla fattorizzazione non negativa per la decomposizione di matrici.

SlopeOne Implementazione dell'algoritmo SlopeOne.

CoClustering Implementazione di un algoritmo per CF basato su CoClustering.

Surprise, quindi, fornisce un solo algoritmo ottimizzato per feedback implicito, l'algoritmo `SVD++`, mentre gli altri algoritmi richiedono che l'utente abbia assegnato all'oggetto una valutazione. È possibile adattare il training set in maniera tale da poter utilizzare gli altri algoritmi, assegnando una valutazione di 1 se la traccia è stata aggiunta alla playlist e 0 altrimenti.

La libreria Surprise non è stata utilizzata a causa del focus su algoritmi di puro CF, quindi l'assenza dei un supporto nativo per l'aggiunta di attributi di oggetti ed utenti, e per i tempi di utilizzo estremamente lunghi.

3.2.2 Libreria LightFM

LightFM [22] è una libreria per il linguaggio Python che implementa un sistema di raccomandazione ibrido basato su fattorizzazione di matrici, pensato sia per situazioni che richiedono l'utilizzo di feedback implicito che esplicito.

Include implementazioni efficienti delle funzioni di costo(ranking losses BPR(Bayesian Personalized Ranking) e WARP(Weighted Approximate-Rank Pairwise). Si tratta di una libreria semplice nell'utilizzo, dato che si basa sulle strutture Dataframe messe a disposizione di Pandas, una libreria open source per l'analisi di dati in Python.

Inoltre, trattandosi di un sistema di raccomandazione ibrido, è possibile integrare attributi di utenti ed oggetti all'interno dell'algoritmo di fattorizzazione classico, rappresentando ogni utente ed oggetto tramite la somma delle rappresentazioni latenti dei loro attributi.

Il modello genera delle rappresentazioni di utenti ed oggetti in un spazio multidimensionale, tramite metodi basati su SGC(Stochastic Gradient Descent), in modo tale da codificare i gusti e le preferenze di un utente sugli oggetti. Queste rappresentazioni, quando moltiplicate, forniscono un punteggio per ogni oggetto per un dato utente, cioè la potenzialità per un oggetto di piacere all'utente. Più alto è questo valore, più l'oggetto è potenzialmente interessante per l'utente.

LightFM supporta diverse funzioni di costo:

logistic Utile in situazioni in cui l'utente può dare valutazioni sia positive che negative ad un oggetto. Si tratta di una funzione di costo suscettibile alla presenza di valori anomali(outlier) nei dati.

BPR: Bayesian Personalized Ranking pairwise loss Funzione di costo che tende a massimizzare la differenza fra la predizione di un esempio di interazione positiva fra utente ed oggetto ed un esempio casuale di interazione negativa, considerando negativa la non interazione fra utente ed oggetto[29]. Utilizzata quando sono disponibili solamente interazioni positive e si vuole massimizzare l'area sotto la curva della curva ROC.

WARP: Weighted Approximate-Rank Pairwise loss Funzione di costo che massimizza il numero di elementi correttamente predetti estraendo in maniera casuale delle predizioni finchè non trova una coppia di elementi che non sono stati correttamente predetti, applicando in tal caso una correzione al modello. Si tratta di una funzione di costo utile quando si trova in una situazione con disponibili solamente interazioni positive e si vuole ottimizzare la prima parte delle raccomandazioni (precision@k)[36].

k-OS WARP: k-th order statistic loss Un'estensione dell'algoritmo WARP, che utilizza la k-esima interazione positiva di un utente con un oggetto come base per aggiornare le coppie estratte.

LightFM mette a disposizione inoltre due tipi di stabilizzatori per il learning rate dell'algoritmo, che consentono di correggerlo durante la fase di training:

adagrad È un algoritmo per l'ottimizzazione del learning rate basato sui gradienti, che lo adatta a seconda degli elementi considerati. Imposta un learning rate piccolo, che quindi causa poche modifiche al modello, nel caso di elementi con attributi frequenti, mentre imposta un learning rate di valore maggiore per gli elementi con feature poco ricorrenti[32]. Ad ogni iterazione corregge la variazione da effettuare sul learning rate considerando i gradienti di tutti gli istanti di tempo precedenti per ogni parametro di ogni oggetto. Utilizzato su distribuzioni di dati sparse.

adadelta È un'estensione dell'algoritmo adagrad che tenta di limitare il ritmo di decremento del learning rate causato dall'utilizzo di adagrad, limitando il numero gradienti accumulati riducendo la finestra temporale di accumulo ad un intervallo temporale definito[32].

LightFM è una libreria che possiede tutte le caratteristiche per essere utilizzata per entrambe le categorie di sfida ed è stata la libreria che avevo scelto per l'implementazione dei sistemi di raccomandazione da utilizzare. Non è stato però possibile utilizzarla a causa del quantitativo di memoria che lo script e l'algoritmo tentavano di allocare, maggiore di quella presente nel sistema, che causava errore di memoria e la conseguente conclusione del processo con eccezione.

3.2.3 Libreria e interfaccia da linea di comando MyMediaLite

MyMediaLite è una libreria per sistemi di raccomandazione sviluppata per il Common Language Runtime (CLR), il componente della piattaforma `.NET` che si occupa di gestire l'esecuzione dei programmi, utilizzando il linguaggio di programmazione C#[18].

Si specializza in sistemi di CF e nello specifico nelle due casistiche più comuni che si possono incontrare:

- Predizione del punteggio potenzialmente assegnato da un utente(*rating prediction*), cioè le casistiche in cui si hanno ha a disposizione feedback esplicito da parte degli utenti.

- Raccomandazione di oggetti da feedback implicito(*item recommendation*).

MyMediaLite è una libreria di utilizzo molto semplice, in quanto fornisce un'interfaccia da linea di comando(CLI) ben documentata per entrambe le casistiche e richiede come input file testuali di formato molto semplice. Fornisce vari metodi per il calcolo di misure per la valutazione delle predizioni di punteggio e delle raccomandazioni di oggetti: MAE, RMSE(eq. (2.10)), CBD, AUC, prec@N, MAP, and NDCG(eq. (3.4)).

La libreria, così come la CLI, sfrutta la serializzazione per il salvataggio su disco dei modelli allenati ed il loro caricamento in memoria, per l'utilizzo futuro senza bisogno di dover ripetere il training, oltre al supporto per molti dei recommender implementati di fornire aggiornamenti in tempo reale sullo stato del training .

Caratteristica interessante di MyMediaLite è il fatto di poter aggiungere attributi ad oggetti ed utenti, così da poter specializzare il modello, tramite semplici file testuali, nonostante l'impossibilità per i recommender di studiare gli attributi in base al contenuto, infatti gli attributi che MyMediaLite accetta corrispondono a semplici identificativi numerici.

La porzione di CLI di MyMediaLite specializzata in feedback implicito offre una varietà di algoritmi:

KNN Implementazione dell'algoritmo k-nearest neighbor basato sugli oggetti `ItemKNN` (sezione 2.5.1.2) e sugli utenti `UserKNN` (sezione 2.5.1.1), con supporto per training incrementale del modello se si utilizzano le misure di similarità `BinaryCosine` e `Cooccurrence`. Inoltre fornisce delle estensioni dei due algoritmo, rispettivamente `ItemAttributeKNN` e `UserAttributeKNN` che, aggiungono il supporto alla correlazione fra gli attributi di, rispettivamente, oggetti ed utenti, perdendo il supporto al training incrementale.

MostPopular Semplice algoritmo, che raccomanda semplicemente le tracce più popolari. La popolarità di un oggetto è data dal numero di volte che l'oggetto compare

nei dati di training. Supporta il training incrementale. Inoltre fornisce l'estensione `MostPopularByAttributes`, che aggiunge il supporto agli attributi degli oggetti, perdendo il supporto al training incrementale.

BPRSLIM Implementazione dei metodi SLIM(Sparse Linear Methods) per la raccomandazione di oggetti, ottimizzato per l'utilizzo del Bayesian Personalized Ranking(BPR) tramite Stochastic Gradient Ascent(SGA). L'algoritmo supporta il training incrementale.

LeastSquareSLIM Implementazione dei metodi SLIM che utilizza per il training del modello un algoritmo del tipo coordinate descent che supporta il training incrementale.

SoftMarginRankingMF Implementazione di un algoritmo per la fattorizzazione di matrici che utilizza una funzione di costo con margine soft tramite Stochastic Gradient Descent. L'algoritmo supporta il training incrementale.

BPRMF Algoritmo per la fattorizzazione di matrici ottimizzato per il BPR ed utilizza la classificazione a coppie degli oggetti per il loro ordinamento. L'implementazione differenzia i parametri di regolarizzazione per utenti ed oggetti, il che consente di avere più controllo sull'algoritmo, che supporta il training incrementale. Inoltre fornisce le estensioni `WeightedBPRMF`, che introduce un meccanismo di sampling degli oggetti che favorisce gli elementi con basso punteggio, e `MultiCoreBPRMF`, che effettua il calcolo della fattorizzazione su più thread. `MultiCoreBPRMF` supporta il training incrementale, che però viene calcolato nella versione utilizzata su singolo core.

WRMF Implementazione di un algoritmo di fattorizzazione di matrice basato su ALS (Alternating Least Squares). Pesi vengono assegnati agli elementi osservati. L'algoritmo supporta training incrementale.

Caratteristica interessante di MyMediaLite è la possibilità di sviluppare algoritmi per le raccomandazioni aggiuntivi a quelli forniti, utilizzando il linguaggio C#, sfruttando l'infrastruttura di classi preesistente ed integrarli nella CLI e nella libreria. Questa caratteristica di MyMediaLite, insieme alla efficiente gestione della memoria riscontrata durante l'utilizzo della CLI, sono stati i motivi che hanno mi hanno portato a sceglierla per l'utilizzo durante la sfida.

3.2.4 Spark MLlib

MLlib [27] è una libreria per machine learning, scritta nel linguaggio Scala e basata sull'infrastruttura distribuita Apache Spark, che fornisce interfacce per l'utilizzo del framework in diversi linguaggi di programmazione: Java, Python, R ed, ovviamente, Scala.

MLlib riesce a sfruttare pienamente l'architettura altamente distribuita ed ottimizzata per algoritmi iterativi di Spark, implementando algoritmi che sfruttano efficientemente l'iterazione. La libreria fornisce un vasto assortimento di algoritmi per il machine learning, compresi algoritmi per la trasformazione, estrazione e selezione di feature, tra cui:

K-means L'algoritmo di clustering probabilmente più comune e più utilizzato nel campo dell'unsupervised learning. L'implementazione di MLlib consiste in una versione parallelizzata dell'algoritmo k-means++ [4] chiamata kmeans||.

Tokenizer Classe per la trasformazione di feature che si occupa di dividere del testo in parole. È possibile utilizzare un'espressione regolare come criterio per la divisione del testo tramite la classe `RegexTokenizer`.

StopWordsRemover Classe per la trasformazione di feature che si occupa di rimuovere da un testo quelle parole di uso estremamente comune e che, di conseguenza, non aggiungono molte informazioni al contesto.

n-gram La classe `NGram` si occupa di creare combinazioni di parole di dimensione N a partire da una lista di parole. Viene tipicamente utilizzato l'output di un tokenizer come input per la creazione degli n-gram.

PCA La classe `PCA` di MLlib si occupa di effettuare l'analisi delle componenti principali, una procedura utilizzata per ridurre la dimensionalità di uno spazio delle variabili applicando delle trasformazioni in modo tale da convertire un insieme di variabili correlate in un insieme di variabili scorrelate, in modo che ogni variabile del nuovo spazio spieghi la maggiore percentuale di varianza possibile. Ognuna di queste nuove variabili porta quindi delle informazioni sulla varianza totale, per cui è possibile ridurre la dimensionalità ignorando alcune di queste variabili, a scapito di parte delle informazioni.

Calcolo TF-IDF MLlib fornisce la classe IDF per il calcolo della TF-IDF. La classe IDF prende un insieme delle frequenze dei termini, che può essere creato sia dalla classe `HashingTF` che dalla classe `CountVectorizer`, che calcolano la frequenza dei termini all'interno del testo, e riduce il peso dei termini più frequenti, scalando tutti gli attributi.

Word2Vec La classe Word2Vec prende una serie di parole rappresentative di un documento, solitamente l'output di uno `StopWordsRemover`, ed allena un modello che associa ad ogni parola un vettore di una data dimensione. Ad ogni documento viene poi associato un vettore corrispondente alla media dei vettori associati alle singole parole.

Ho scelto di utilizzare MLlib nella parte della sfida che richiede principalmente l'utilizzo di tecniche di clustering per effettuare le raccomandazioni, nello specifico le playlist del test set che presentano una o nessuna traccia seme, per la possibilità di sviluppare il codice utilizzando il linguaggio python e di utilizzare il cluster universitario per i test e la messa a punto.

3.2.5 Linguaggio C#

C# [2] è un linguaggio di programmazione sviluppato da Microsoft per lo sviluppo di applicativi da utilizzare sulla piattaforma `.NET`. Si tratta di un linguaggio di programmazione evolutosi dai linguaggi C e C++ ed ispiratosi parzialmente al linguaggio Java, infatti i programmi scritti in C# sono destinati ad essere eseguiti all'interno del framework `.NET`. Nonostante esistano altri linguaggi di programmazione utilizzabili con il framework `.NET`, C# è il linguaggio che più aiuta a seguire le linee guida di buona programmazione fornite da Microsoft, dato che è stato creato specificatamente per aiutare il programmatore a trarre vantaggio dalle caratteristiche del framework.

Il framework `.NET` è costituito da due componenti principali:

librerie Sono librerie, file contenti codice intermedio(CIL) precompilato, contenenti componenti generici a tutte le applicazioni o specifici per alcune aree di interesse dello sviluppo di applicativi come, ad esempio, lo sviluppo web tramite la libreria `ASP.NET`.

CLR Il common language runtime(CLR) è l'ambiente controllato in cui vengono eseguite le applicazioni per il framework. Si occupa di leggere il codice intermedio, precompilato dal sorgente C#, convertirlo in linguaggio macchina e gestire l'esecuzione dei programmi[12].

C# è quindi un linguaggio gestito ed in quanto tale beneficia dei classici vantaggi di un linguaggio del genere come, ad esempio, robustezza del codice grazie ai controlli su overflow quando devono essere effettuate operazioni aritmetiche, alla limitazione delle conversione di dati implicite solamente a quelle conversioni che non portano a perdita di informazione (come la discesa dell'albero di ereditarietà o presenza di operatore di conversione esplicito), limitazione all'utilizzo dei puntatori solamente in particolari blocchi di codice, che devono essere impostati esplicitamente come unsafe ed abilitati in fase di compilazione del sorgente in CIL. Inoltre il CLR dispone di un *garbage collector*[11], che si occupa della deallocazione degli oggetti creati dinamicamente quando non vengono più utilizzati, evitando i fenomeni dei *dangling pointer* e *memory leak*. Nonostante si tratti di un linguaggio gestito, è possibile utilizzare codice non gestito richiamando all'occorrenza funzioni da dll esterne alle librerie del framework.

C# è un ottimo linguaggio, che ho utilizzato per lo sviluppo di numerosi progetti, sia in ambiente universitario che esterno, e che ho considerato fortemente per l'utilizzo a causa delle sue caratteristiche, della possibilità di sviluppare, all'occorrenza, un recommender da integrare con MyMediaLite e della presenza del componente LINQ del framework `.NET`, che avrebbe potuto rivelarsi utile durante lo sviluppo del suddetto recommender.

La ragione del mancato utilizzo di C# ricade principalmente nell'assenza di un supporto ufficiale da parte di Apache per l'utilizzo di Spark.

3.2.5.1 Language Integrated Query

Language Integrated Query(LINQ)[13] è un componente del framework `.NET` che aggiunge supporto nativo alle query, cioè espressioni che specificano come recuperare dati da un'origine, per dati di origine arbitraria. LINQ estende il linguaggio C# aggiungendo la possibilità di definire query tramite parole chiave del linguaggio, che possono essere utilizzate per estrarre dati da varie sorgenti, come liste, file xml e database o qualunque tipo di

dato per il quale venga reso disponibile un'integrazione con LINQ.

Trattandosi di un linguaggio unificato per la creazione di query, LINQ elimina la necessita per il programmatore di imparare diversi linguaggi e diverse sintassi per effettuare quelle che sono fondamentalmente le stesse operazioni.

La sintassi di LINQ permette la scrittura di operazioni anche complesse su una sorgente di dati in maniera molto compatta tramite parole chiave di alto livello, come accade per il linguaggio SQL, e funzioni lambda da utilizzare su oggetti di tipi anonimi.

LINQ estende il linguaggio non soltanto tramite parole chiave, ma anche aggiungendo una serie di metodi, che hanno pressocchè le stesse funzioni delle parole chiave, agli oggetti che implementano la classe `IEnumerable`, in maniera simile agli oggetti *Stream* del linguaggio Java.

3.2.6 Linguaggio Python

Python[17] è un linguaggio di programmazione estremamente ad alto livello che si presta facilmente ai più disparati paradigmi di programmazione, da quella orientata agli oggetti a quella funzionale. Si tratta di un linguaggio semplice da utilizzare e da capire, in quanto i blocchi di codice sono delimitati dall'indentazione, cioè l'aumento o riduzione dell'indentazione portano rispettivamente ad entrare od uscire da un dato blocco di codice. Inoltre, trattandosi di un linguaggio di programmazione ad alto livello, la scrittura del codice si avvicina molto a quella del linguaggio parlato.

Python, in un maniera simile a C#, è un linguaggio gestito, il che significa che anch'esso beneficia di una robustezza intrinseca del codice maggiore rispetto ad altri linguaggi di programmazione e di una gestione automatica della memoria. Python è inoltre un linguaggio con tipizzazione dinamica delle variabili, ciò significa che le variabili in un script python possono essere considerate semplicemente dei nomi assegnati a degli oggetti che risiedono in memoria. Lo stesso oggetto può essere associato a più nomi ed un nome può essere associato ad oggetti diversi, cambiando tipo a seconda dell'oggetto associato.

Nonostante Python sia un linguaggio interpretato, quindi leggermente più lento di linguaggi compilati, come C o C++, o semicompilati, come C# e Java, è possibile estenderlo

con chiamate a codice compilato laddove ci sia bisogno di una maggiore efficienza, il che rende Python un linguaggio estremamente versatile. Inoltre Python dispone di una comunità molto attiva ed un grande numero di moduli aggiuntivi al linguaggio.

Python è un linguaggio estremamente potente, che ho scelto di utilizzare durante la sfida per la facilità con cui si possono gestire file json tramite un modulo che li legge in memoria come dizionari con chiavi testuali, per la possibilità di scrivere codice compatto e facilmente leggibile tramite comprensione di liste, set e dizionari, per la semplice gestione delle risorse tramite gestori di contesto (Context Manager) e per il supporto ufficiale a Spark ed MLlib.

Capitolo 4

Implementazione

Alla base di un buon sistema di raccomandazione c'è un modello dei dati raccolti sugli utenti. È quindi fondamentale, durante le fasi di sviluppo del sistema, evitare di incorrere nei fenomeni chiamati underfitting ed overfitting, che causano, rispettivamente lo sviluppo di modelli inefficienti e, cosa addirittura peggiore, modelli che sembrano avere grande efficienza durante le fasi di sviluppo, per poi rivelarsi pressocchè inutili sul campo. A tal fine il dataset totale viene solitamente diviso in:

- Training set: la parte di dataset utilizzata per allenare effettivamente il modello. Solitamente composto dalla percentuale più grande di dati.

- Validation set: la parte di dataset utilizzata durante la fase di scelta dell'algoritmo e la regolazione dei suoi parametri per la valutazione del modello.

- Test set: la parte di dataset utilizzata per controllare la correttezza della scelta effettuata. Consente di evitare la scelta di un modello che si è rivelato efficiente sul validation set per pura casualità.

Il test set (challenge set) è stato reso disponibile poco dopo l'inizio della challenge, per cui non è stato necessario costruirne uno. In questo capitolo illustro il lavoro effettuato durante le fasi di costruzione del validation set, pre e post processing dei dati per l'utilizzo con la libreria MyMediaLite e sperimentazione con MyMediaLite e Spark MLlib.

4.1 Costruzione set di validazione

4.1.1 Studio del challenge set

Le proprietà interessanti delle playlist contenute nel challenge set ai fini della costruzione di un set di validazione che ne condivida le proprietà sono la quantità di tracce omesse e la quantità di tracce rimaste all'interno della playlist. Tali proprietà del challenge set sono le uniche analizzabili, dato che queste sono in numero molto inferiore rispetto a quelle del MPD (appendice A e appendice B).

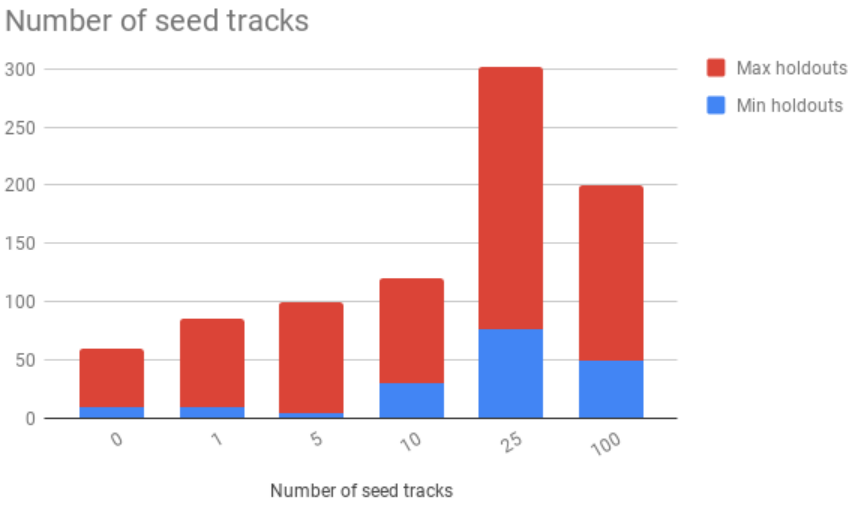

Figura 4.1: Numero tracce seme vs numero tracce omesse per il test set

Anche se, stando alle specifiche della competizione (vedi sezione 3.1.2), le playlist del challenge set sono state estratte in maniera casuale, come si può vedere dal grafico in fig. 4.1 e dalla tabella 4.1 sembra che una volta estratte queste siano state raggruppate in modo tale che il numero di tracce totali per playlist rientrasse in un certo range.

4.1.2 Generazione set di validazione

Partendo dalle informazioni in tabella 4.1, raccolte durante l'analisi (vedi sezione 4.1.1), ho creato un set di validazione le cui playlist seguissero la stessa distribuzione e fossero allo stesso tempo estratte in maniera casuale.

		Number of holdouts		Total tracks	
Number of seed tracks	Amount of playlists	Min	Max	Min	Max
0	1000	10	50	10	50
1	1000	9	77	10	78
5	2000	5	95	10	100
10	2000	30	90	40	100
25	2000	76	225	101	250
100	2000	50	150	150	250

Tabella 4.1: Numero minimo e massimo tracce omesse e totali per tracce seme

Sapendo già che le playlist del training set, il MPD, sono state estratte in maniera casuale (vedi sezione 3.1.1) ho assunto che la distribuzione delle playlist all'interno dei singoli file fosse anch'essa randomica. Per ottenere una distribuzione il più eterogenea possibile tra gli elementi del set di validazione, ho optato per estrarre porzioni (slice) del MPD casualmente e da queste selezionare le playlist che soddisfano i criteri sulle tracce totali.

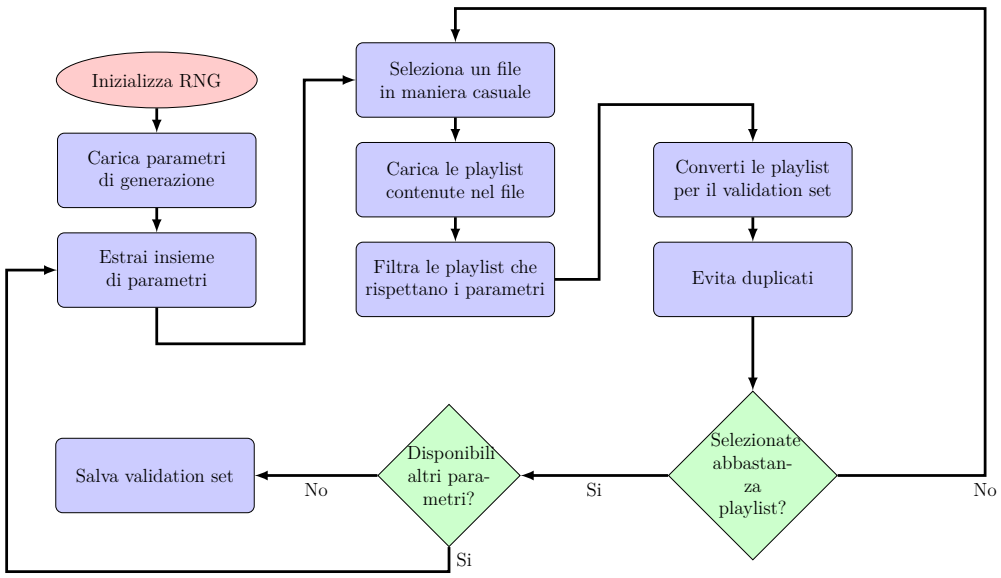

Figura 4.2: Diagramma di flusso creazione validation set

In fig. 4.2 viene illustrato, tramite un diagramma di flusso, il processo di creazione del set di validazione, che si basa su un generatore di numeri randomici o RNG (Random

Number Generator), che viene istanziato all'inizio del processo. Per effettuare scelte randomiche, utilizzo la classe `SystemRandom` del pacchetto `random` in Python, che si basa sulla funzione `os.urandom` per ottenere numeri casuali utilizzando chiamate di sistema, quindi utilizzando sorgenti casuali proprie del Sistema Operativo, se queste sono disponibili. Vengono poi caricati i parametri necessari per la generazione del validation set. Questi parametri corrispondono alle prime due ed alle ultime due colonne della tabella 4.1, ogni insieme di parametri corrisponde ad una riga della sopracitata tabella. Per ogni insieme di parametri, viene estratto e caricato in memoria uno dei file contenenti una sezione del MPD. Delle playlist contenute all'interno della struttura dati, ne vengono filtrate quelle che soddisfano i parametri in esame. Ogni playlist del file, per essere selezionata deve quindi:

- Avere un numero di tracce (parametro `num_tracks`) superiore al minimo ed inferiore al massimo per le tracce seme (seed tracks) considerate.
- Non essere già stata selezionata in precedenza.

Se il numero di playlist selezionate dopo il processo di filtraggio non ammonta a quello desiderato, viene estratto e caricato in memoria un nuovo file.

Il processo di estrazione e selezione delle playlist viene ripetuto per ogni insieme di parametri. Il set di validazione così ottenuto viene salvato su file utilizzando lo stesso formato del test set (vedi sezione 3.1.2).

4.2 Conversione formato file

L'interfaccia da linea di comando di Mymedialite [18] richiede quattro file per effettuare delle raccomandazioni significative:

- Training file: il file contenente le informazioni sulle relazioni utente-oggetto. Per il file ho scelto il formato con colonne separate da tab, tsv. Trattandosi di uno use case in cui le relazioni hanno soltanto feedback positivo, ogni linea dei file sarà composta da

  ```
  id_utente\tid_oggetto
  ```

- Item attributes file e user attributes: i file contenente le informazioni, rispettivamente, sugli oggetti e sugli utenti. Anche per questo file è stato scelto un formato separato da tab. Ogni riga di questi due file è composta da

```
Item attributes file:  id_oggetto\tid_attributo
User attribute file:   id_utente\tid_attributo
```

  Per motivi di performance hardware del server utilizzato e, quindi, tempi di esecuzione del singolo comando del recommender, diverse versioni di questi file sono state utilizzate (vedi tabella 4.2).

- Test user file: il file contente la lista degli utenti per i quali effettaure le raccomandazioni. Ogni riga di questo file è composta da:

```
id_utente
```

Version	User attributes	Item attributes
V1	Empty	Empty
V2	Empty	Artist uri - Album uri
V3	- Collaborative - Modified at (last modification time) - Name - Num albums - Num artists - User attributes - Num edits - Num followers - Num tracks	- Artist uri - Album uri - Artist name - Album name - Track name

Tabella 4.2: Versioni mappa

4.2.1 Mappa attributi-identificativo

La costruzione di file utilizzando il formato di cui alla sezione 4.2 richiede la creazione di una mappa per la conversione degli attributi da un identificativo testuale ad un identificativo numerico, utilizzabile dall'interfaccia da linea di comando.

Alcuni degli attributi sono stati esclusi durante la creazione della versione finale della mappa, in quanto non rilevanti ai fini del recommender. Nello specifico:

- L'identificativo della playlist, in quanto esso è già numerico ed univoco.
- La descrizione della playlist, in quanto essa è un attributo opzionale e raro all'interno del dataset, essendo la possibilità di aggiungere una descrizione un'aggiunta relativamente nuova al servizio fornito da Spotify.
- La durata in millisecondi della playlist e della singola traccia dato che, non essendo MyMediaLite capace di analizzare gli attributi per ciò che realmente rappresentano, ma confrontandone soltanto gli identificativi, non è possibile tenere in considerazione di durate simili di tracce o playlist, in quanto due tracce dovrebbero avere durata identica al millisecondo per essere considerate simili dal sistema.
- Posizione della traccia all'interno della playlist, in quanto irrilevante ai fini dei recommender in MyMediaLite ed utilizzata soltanto in fase di calcolo delle metriche.

Il processo di costruzione della mappa (vedi fig. 4.3) risulta poco complicato. Data la lista di file che contengono le porzioni del dataset, ogni slice viene caricata in memoria e per ogni playlist e traccia vengono inseriti in un set i rispettivi attributi. L'utilizzo di un set consente di evitare di avere entrate duplicate all'interno della mappa finale, in quanto l'implementazione di un set nel linguaggio python non consente di conservare elementi duplicati.

Lo stesso procedimento viene applicato alle playlist del test set, aggiungendone i pochi attributi, quando presenti, al set. Durante la lettura del test set può essere evitata la lettura delle tracce seme delle playlist e l'aggiunta dei loro attributi al set, dato che nelle specifiche è stato precisato che non esistono tracce appartenenti al test set che non compaiano anche all'interno del training set.

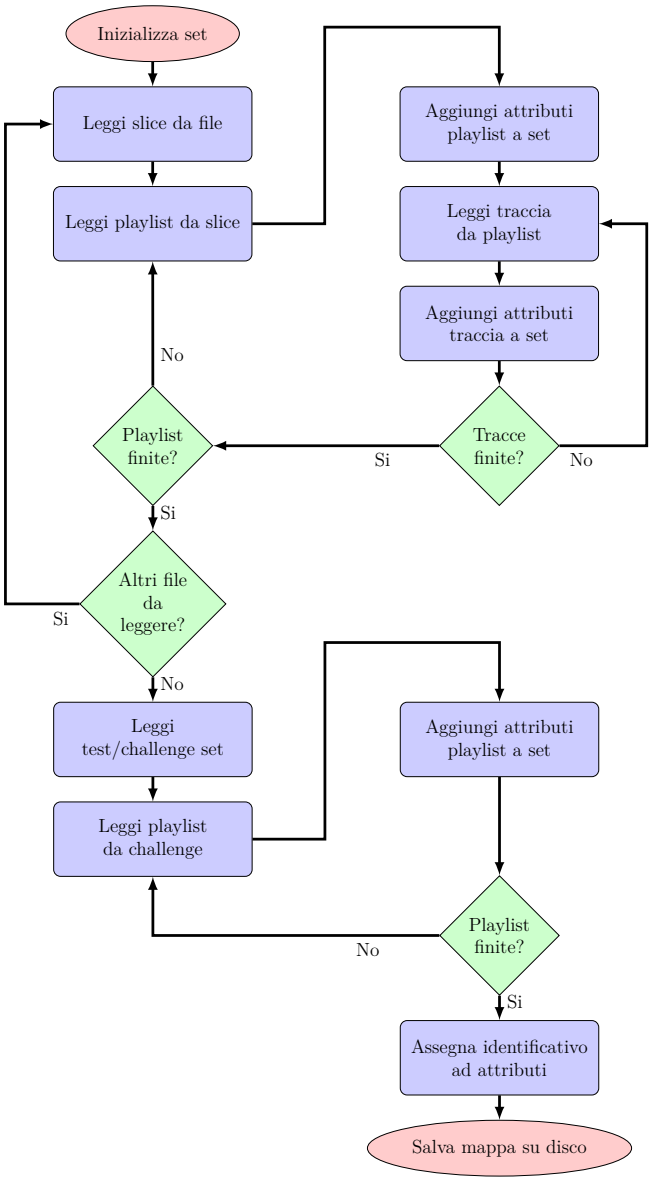

Figura 4.3: Diagramma di flusso creazione mappa

4.2.2 Generazione dei file

Il diagramma di flusso di figura fig. 4.4 illustra il processo di creazione dei file necessari (vedi tabella 4.3). Lo script inizializza le liste contenenti gli attributi da inserire nei file e la lista contenente i nomi dei file dell'MPD. Vengono poi caricate in memoria le mappe attributo-identificativo ed il test/challenge set. Da quest ultimo vengono estratti gli identificativi delle playlist per cui effettuare le raccomandazioni e salvati su disco all'interno del file `user_to_predict.txt`.

Ogni porzione dell'MPD viene letta in memoria e, delle playlist contenute, vengono considerate soltanto quelle non appartenenti anche al test set, in quanto queste ultime, secondo le specifiche (vedi sezione 3.1.1.1 sezione 3.1.2.2), hanno meno attributi rispetto alle loro controparti del dataset. Per ogni playlist vengono quindi salvate le corrispondenze tra il pid e l'id dell'attributo. Per ogni traccia appartenente alla playlist in esame viene salvata al relazione tra pid e l'identificativo corrispondente all'URI della traccia e le corrispondenze tra URI ed attributi della traccia. La scelta di utilizzare dei set per conservare le relazioni prima del salvataggio su file viene giustificata da queste ultime corrispondenze, dato che è pressocchè garantito che una traccia sia contenuta in più playlist.

Tale processo viene ripetuto per le playlist contenute nel test/challenge set ed i set risultanti risultanti dal procedimento vengono salvati nei file `training_file.tsv`, `user_attributes.tsv` e `item_attributes.tsv`.

4.3 Automatizzazione degli esperimenti con MyMediaLite

L'automatizzazione degli esperimenti è stato un passo fondamentale verso l'ottimizzazione delle tempistiche di esecuzione e verso la ricerca dell'algoritmo e dei parametri adatti al raggiungimento di un risultato ritenuto ottimale. La scelta dei parametri è stata, ovviamente, subordinata alla configurazione hardware del sistema in utilizzo, considerando il fatto che questo fosse un sistema condiviso con altri utenti ricercatori e che le tempistiche medie per il completamento di un solo comando fossero nell'ordine delle ore o dei giorni, a seconda dell'algoritmo utilizzato.

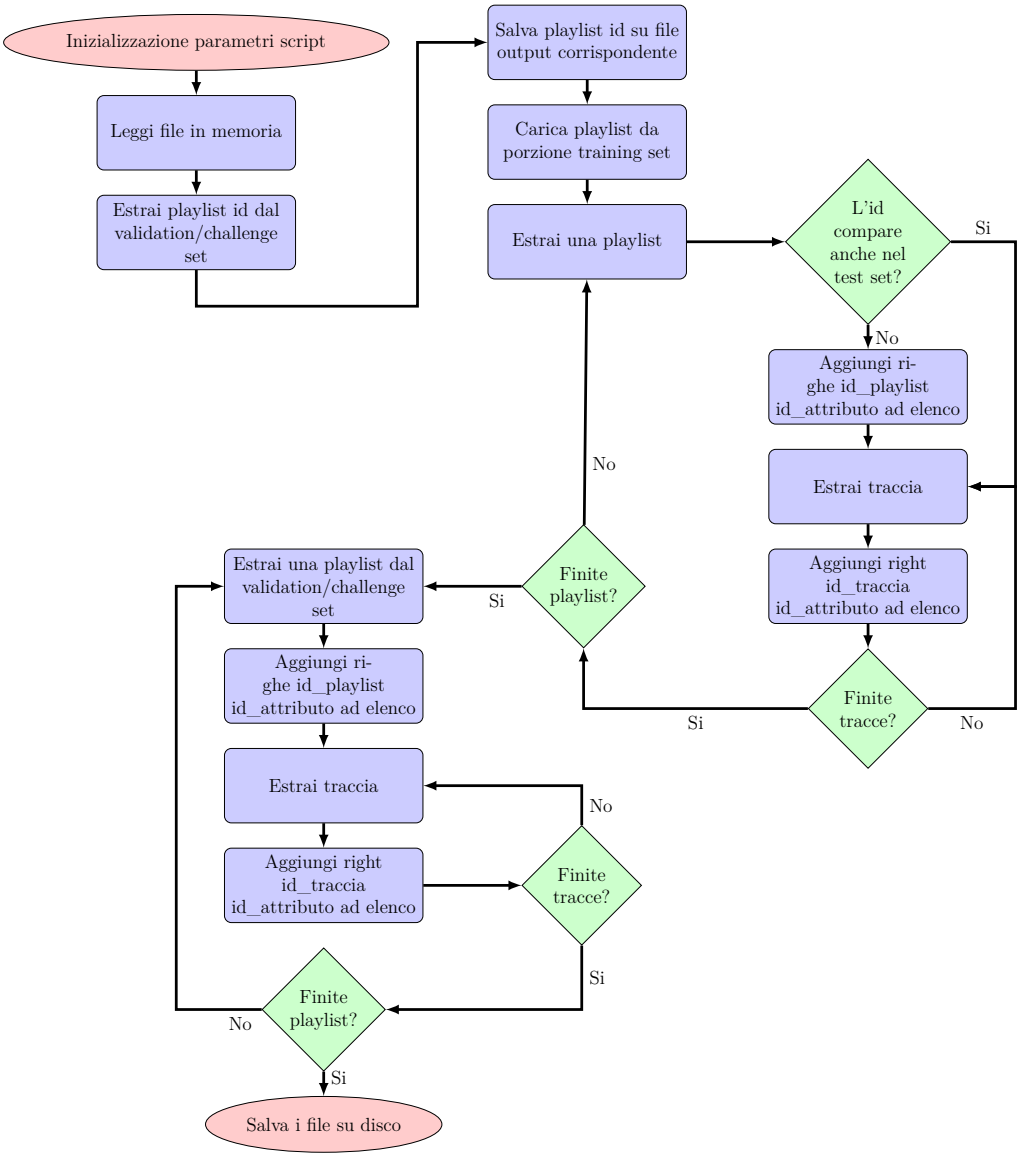

Figura 4.4: Diagramma di flusso della creazione grafico

4.3.1 Interfaccia da linea di comando MyMediaLite

Un singolo esperimento può essere effettuato utilizzando l'interfaccia da linea di comando di MyMediaLite[18], nello specifico il comando `item_recommendation`, i cui parametri principali sono strutturati come segue:

```
item_recommendation --training-file=FILE --recommender=METHOD [OPTIONS]
```

Parametro	Descrizione parametro
- -recommender-options=OPTIONS	Elenco delle opzioni specifiche del recommender scelto
- -user-attributes=FILE	File contenente le informazioni sugli attributi utente, una tupla per riga.
- -item-attributes=FILE	File contenente le informazioni sugli attributi degli oggetti, una tupla per riga.
- -all-items	Indica al sistema di utilizzare come candidati per le raccomandazioni tutti gli oggetti conosciuti
- -predict-items-number=N	Imposta il numero di oggetti da raccomandare ad ogni utente
- -test-users=FILE	File contenente gli identificativi degli utenti per cui effettuare le raccomandazioni
- -prediction-file=FILE	Indica il file il cui salvare le raccomandazioni

Tabella 4.3: Parametri opzionali rilevanti della CLI di MyMediaLite

I parametri opzionali di cui alla tabella tabella 4.3 sono generali dell'eseguibile e validi per tutti i recommender messi a disposizione. Di questi, per il parametro "- -predict-items-number" ho scelto un valore fisso di 800. Questa scelta è subordinata alla necessità di utilizzare il parametro "- -all-items" per indicare all'applicativo di utilizzare ogni oggetto, quindi ogni traccia, conosciuto come candidato per le raccomandazioni. Ciò è dovuto all'impossibilità, a meno di utilizzare un ulteriore file in input, di impostare come candidati, per ogni utente, tutti gli oggetti presenti all'interno del training set, ma che non sono presenti anche all'interno del test set. Per ovviare a questa limitazioni ho scelto un numero

di raccomandazioni per utente maggiore delle 500 richieste dalle specifiche challenge, per poi rimuovere quegli oggetti che sono presenti tra le tracce seme, per utente. La struttura dei file richiesti dai parametri che li prevedono è discussa nella sezione sezione 4.2. I

Recommender	Parameter	Default value
WRMF	num_factors	10
	regularization	0.015
	alpha	1
	num_iter	15
BPRMF family: - BPRMF - MultiCoreBPRMF - WeightedBPRMF	num_factors	10
	bias_g	0
	reg_u	0.0025
	reg_i	0.0025
	reg_j	0.00025
	num_iter	30
	learn_rate	0.05
-BPRMF -MultiCoreBPRMF	uniform_user_sampling	False
	with_placement	False
	update_j	True
-MultiCoreBPRMF	num_threads	100

Tabella 4.4: Parametri opzionali dei principali recommender utilizzati

parametri dei recommender di cui alla tabella tabella 4.4 richiedono un formato del tipo `parametro=valore`.

4.3.2 Script di automatizzazione

Lo script sfrutta un file di testo, contenente i parametri dei comandi da lanciare. Ogni riga di questo file viene quindi strutturata come segue:

```
recommender \t training file \t test users file \t user attribute file \t
↪ item attribute file \t output file \t model file \t model mode
↪ (save/load) \t Calculate ratings (yes/no)\t Json format test set \t
↪ recomm_option1=Value recomm_option2=Value... recomm_optionN=Value
```

Lo script lancia i comandi, uno per riga, utilizzando i parametri forniti nel file, converte il file fornito in output dal comando nel formato richiesto da Spotify e, se impostato, calcola i punteggi della raccomandazione sul test set.

Trattandosi di uno script i cui tempi di esecuzione possono essere dell'ordine degli svariati giorni ho pensato di implementare un logger, che sfrutta la mia chat con il mio bot di Telegram, per mandare aggiornamenti sullo stato di avanzamento dell'esecuzione. La classe `TelegramLogger`, per quanto semplice ed elementare, adempie perfettamente allo scopo. Un'istanza di `TelegramLogger` viene sostituita allo standard output dello script, in modo tale da poter utilizzare la funzione `print` senza parametri aggiuntivi, come se si stessero stampando messaggi sul terminale[1].

Il diagramma di flusso in figura fig. 4.5 illustra il processo di esecuzione della prima sezione dello script utilizzato per automatizzare l'esecuzione degli esperimenti. Il logger viene istanziato, inizializzandolo con i parametri relativi al mio bot di telegram quali il token del bot e l'identificativo della chat a cui inviare gli aggiornamenti e l'istanza viene sostituita allo standard output del processo. Il file contenente i parametri viene caricato in memoria, ignorando la prima riga, che contiene il formato delle successive. Data una riga valida, questa viene analizzata e ne vengono estratti i parametri. Viene quindi costruito il comando da eseguire e, una volta aggiunto il percorso del file di output del comando alla lista di raccomandazioni di cui calcolare le metriche, se specificato dall'utente, l'eseguibile lanciato tramite la funzione `Popen`. La funzione `Popen` esegue un comando in modalità non bloccante, il che significa che viene data la scelta all'utente se e quando aspettare la fine del comando eseguito. Se l'opzione `--parallel` è stata inserita fra i parametri da linea di comando, la struttura dati contenente le informazioni sul processo appena iniziato viene inserita in una lista, altrimenti lo script aspetta la conclusione del comando prima di eseguire il successivo. Alla conclusione dell'analisi del file, viene controllata la lista contenente le informazioni sui processi e ne viene attesa la conclusione. Se l'opzione `--parallel` non è stata inserita, la lista sarà ovviamente vuota e lo script procede alla

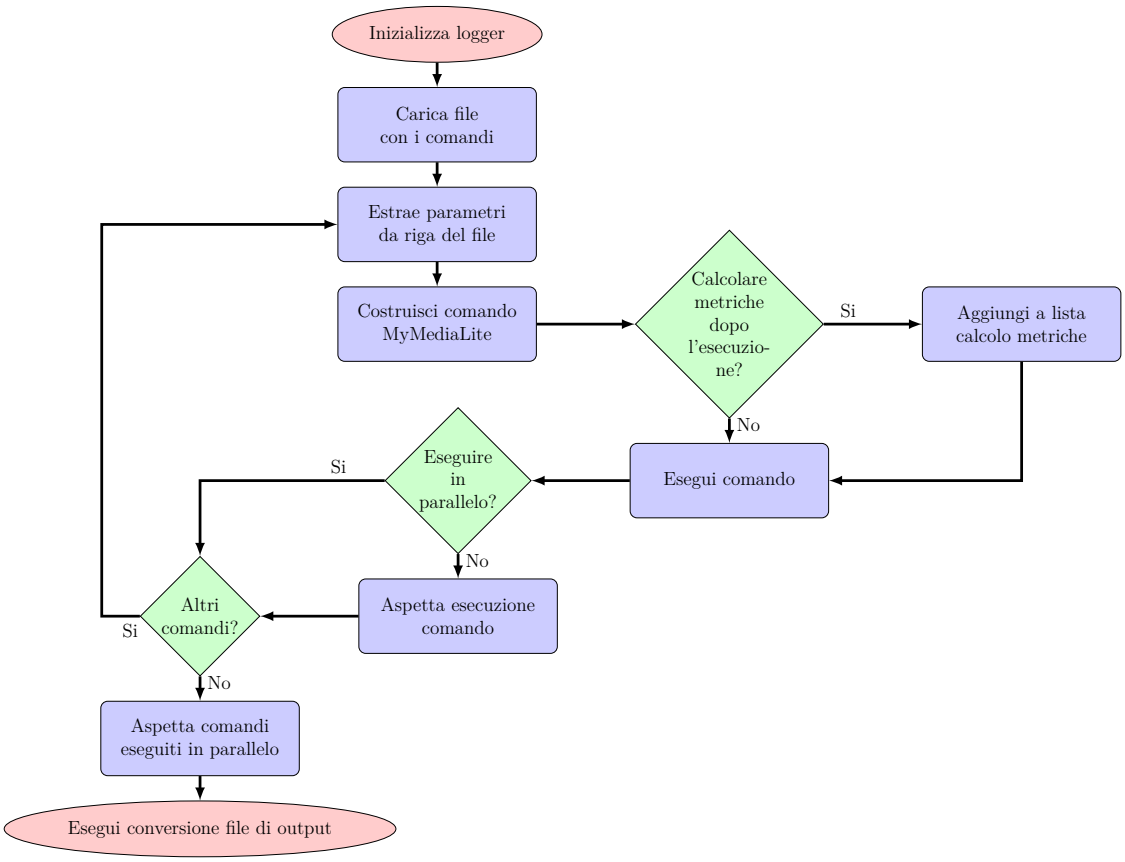

Figura 4.5: Diagramma di flusso script di esecuzione MyMediaLite

conversione dei file di output dal formato di uscita di MyMediaLite al formato richiesto dalle specifiche.

4.3.2.1 Conversione file di Output

Il comando `item_recommendation` di MyMediaLite produce un file di output contenente, per ogni riga, l'identificativo dell'utente, che corrisponde all'identificativo della playlist per cui effettuare le raccomandazioni, e la lista degli identificativi degli oggetti raccomandati con il relativo punteggio sulla confidenza della raccomandazione. Ogni riga di tale file è quindi strutturata come segue:

```
id_utente \t [id_oggetto1:score1, id_oggetto2:score2,... ,
↪  id_oggettoN:scoreN]
```

Le specifiche della challenge invece richiedono, per la sottomissione della soluzione al sistema remoto, un file del seguente formato seguente formato, cioè l'output dello script di conversione:

- La prima riga valida, cioè non bianca o commentata, deve necessariamente iniziare con `team_info`, seguito dal nome del team, sezione della challenge, ed una email di contatto. La sezione della challenge può avere valore `main` o `creative`

  ```
  team_info, team_name, main, team_email@example.com
  ```

- Ogni riga successiva sarà composta dall'identificativo della playlist, seguito da esattamente 500 raccomandazioni:

  ```
  pid, trackuri_1, trackuri_2, trackuri_3, ..., trackuri_499,
  ↪ trackuri_500
  ```

La conversione di formato (vedi figura fig. 4.6), viene effettuata dallo script una volta terminati tutti i comandi lanciati, siano essi stati eseguiti in parallelo o meno.

La mappa creata in sezione 4.2.1 ed utilizzata in sezione 4.2.2 per assegnare un identificativo numerico agli URI delle tracce viene caricata in memoria e l'associazione viene invertita, consentendo una facile conversione dell'identificativo presente nel file ogni file di uscita di MyMediaLite nell'URI originale. Scorrendo ogni riga dei file, si estrae l'identificativo della playlist, cioè il pid. Dalla rimanenza della riga vengono estratti gli identificativi delle tracce e riconvertiti in URI. La riga viene salvata in attesa della scrittura su disco, che avviene alla conclusione delle operazioni su tutte le righe del file di output di MyMediaLite, il quale viene sostituito dal file nel formato desiderato.

4.3.2.2 Calcolo della valutazione delle raccomandazioni

Per quelle righe del file con impostata la generazione delle metriche(sezione 3.1.3), vengono estratti i file da valutare ed i file contenenti i test set, quindi le varie tracce omesse, da una coda precedentemente riempita(sezione 4.3.2).

Per ogni playlist viene calcolata una versione semplificata delle tre metriche calcolate dal sistema di sottomissione, in quanto vengono calcolate soltanto a livello di traccia esatta e non anche a livello di artista ed album. Il calcolo della metrica R-precision (vedi

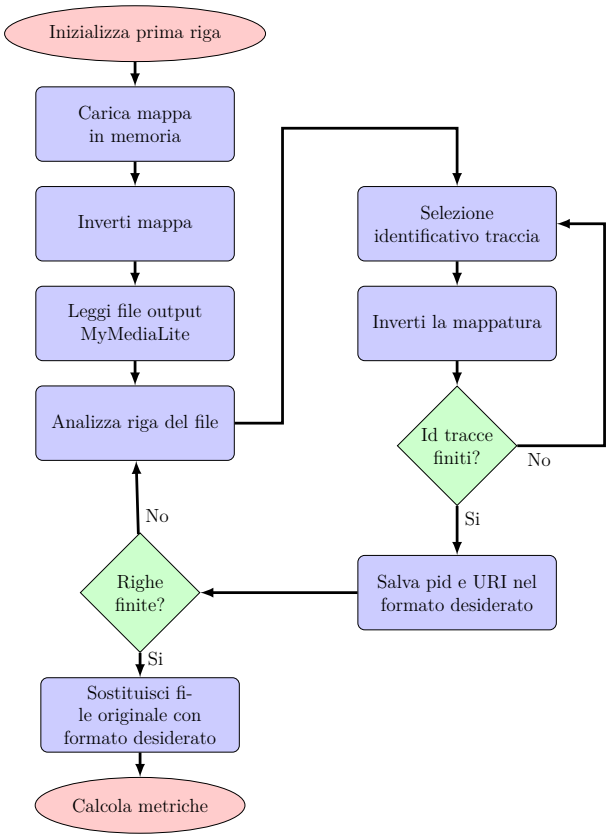

Figura 4.6: Diagramma di flusso sezione conversione output dello script

eq. (3.1)) risulta il più banale, in quanto si tratta del rapporto tra il numero di tracce rilevanti consigliate dal recommender ed il numero totale di tracce rilevanti. È una metrica che valuta esclusivamente la presenta di tracce rilevanti nelle raccomandazioni, indipendentemente dall'ordine. Il calcolo della metrica NDCG, Normalized Discount Cumulative Gain, richiede il calcolo di due metriche separate, la metrica DCG (vedi eq. (3.2)), Discount Cumulative Gain, e la metrica IDCG, Ideal Discount Cumulative Gain che ha un valore costante per ogni playlist, in quanto si tratta del DCG ideale, cioè il DCG che che si avrebbe se tutte le tracce raccomandate fossero rilevanti e nell'ordine corretto. Il DCG, invece, si calcola tenendo conto della rilevanza della traccia in esame, a cui viene assegnato il valore 1 se è presente nella lista di tracce omesse, in caso contrario viene assegnato il valore 0. Si tratta di una metrica che tiene in alta considerazione la posizione della traccia nelle raccomandazioni, in quanto avrà valore maggiore se le tracce raccomandate rilevanti

si trovano nelle prime posizioni. Il valore del NDCG (vedi eq. (3.4)) viene infine calcolato facendo il rapporto tra DCG e IDCG. Il calcolo della metrica Recommended Songs Clicks (vedi eq. (3.5)) risulta anch'esso semplice, in quanto si tratta di una metrica che tiene conto puramente della posizione della traccia all'interno delle raccomandazioni. Si tratta semplicemente del calcolo della parte intera del rapporto fra l'indice della traccia all'interno della lista ed il valore 10. Il valore di default di 51 viene ritornato se nessuna traccia raccomandata risulta rilevante. Le metriche finali del file in esame sono, infine, la media delle metriche per ogni playlist. Il processo di calcolo (vedi fig. 4.7) viene ripetuto per ogni file inserito nella coda.

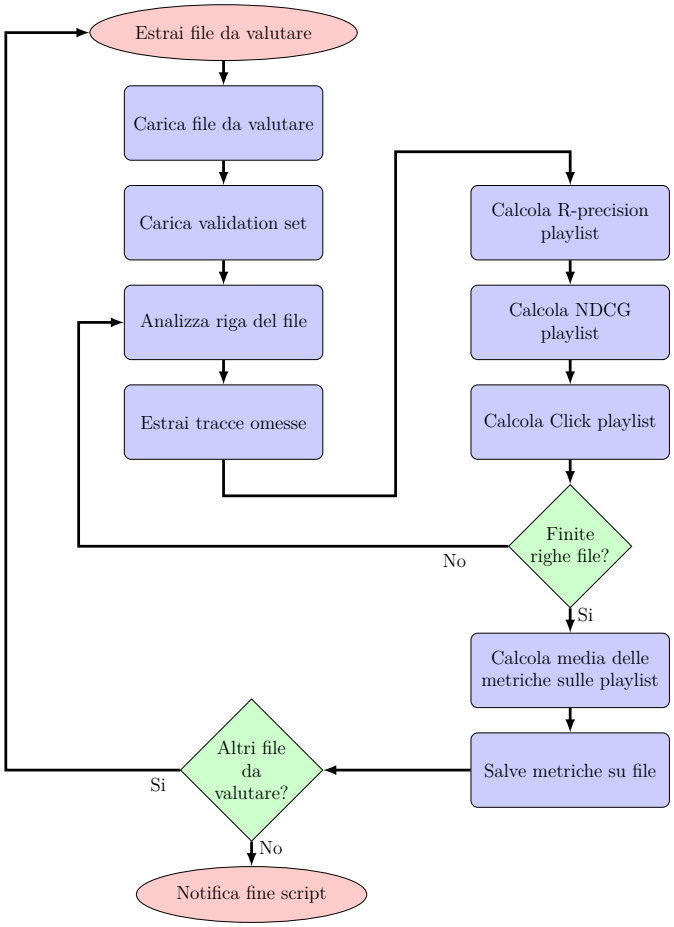

Figura 4.7: Diagramma di flusso calcolo metriche

4.4 Esperimenti con Spark MLlib

Dato che il nome delle playlist è l'unico elemento utile del validation set a disposizione per la sezione della sfida basata sul cold start, per effettuare raccomandazioni basate sul contenuto, la scelta è ricaduta sull'utilizzo di Spark e la sua libreria MLlib, principalmente per la possibilità di utilizzare il linguaggio python per lo sviluppo, tramite le librerie pySpark, ed il fatto che gli oggetti Dataframe di pySpark hanno la possibliità di venire creati a partire da file json, evitando quindi di dover effettuare modifiche del formato dei file.

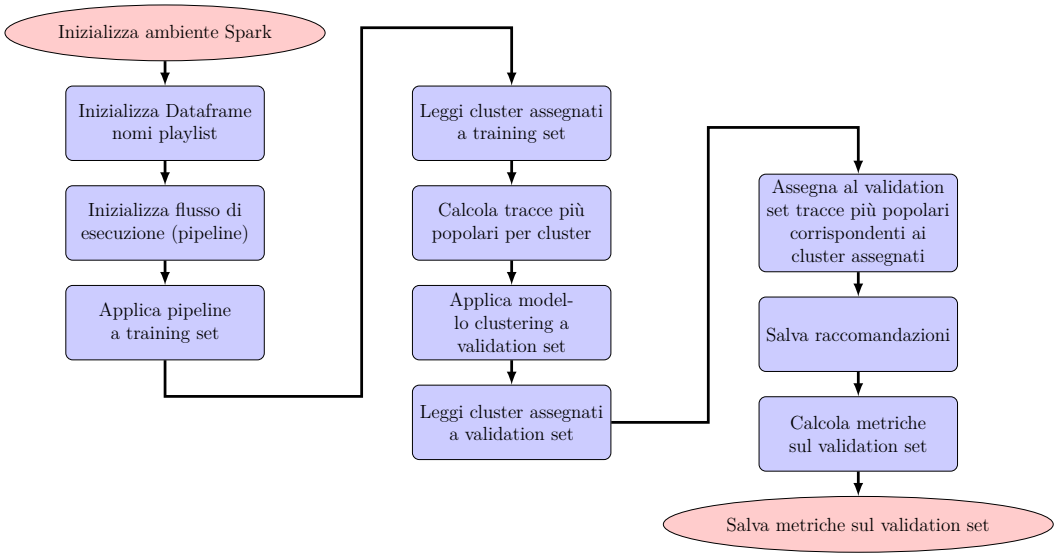

Figura 4.8: Diagramma di flusso script pySpark

L'esecuzione di uno degli script utilizzati per lo studio con pySpark risulta apparentemente semplice, se ci si focalizza solamente sul diagramma di flusso, dato che la complessità viene spostata sull'infrastruttura sottostante e lo script lavora su strutture dati molto di alto livello, quali i dataframe, capaci di definire operazioni su gruppi molto grandi di dati. Gli script effettuano inizialmente il setup dell'ambiente Spark, cioè caricano la configurazione da utilizzare. Nel caso di esecuzione dello script sul cluster, questa configurazione consiste solamente nell'impostare il nome del lavoro all'interno del cluster, dato che gli altri parametri vengono forniti alla CLI `spark2-submit`, altrimenti nel caso di esecuzione su di un server standalone, bisogna impostare il `master` del lavoro come `local` e le impostazioni sulla memoria massima assegnabile all'esecutore ed al driver e sulla dimensione massima

dei risultati che possono essere restituiti al driver, a seconda delle capacità della macchina in utilizzo. Successivamente lo script inizializza i dataframe contenenti le playlist, leggendo le porzioni di dataset dai file json ed estraendone i parametri di interesse, cioè l'identificativo, il nome e la lista delle traccie di ogni playlist. Viene poi inizializzata la pipeline con la lista di trasformazioni da effettuare sui dati. La lista di trasformazioni varia a seconda delle feature da utilizzare come input per l'algoritmo di clustering:

TFIDF Per estrarre la TfIdf dal nome delle playlist per l'utilizzo come input bisogna che i nomi di ogni playlist vengano divisi in singole parole, il che rende necessario l'utilizzo di un `Tokenizer` come primo passo da eseguire, seguito da uno `StopWordsRemover` e dal calcolo della Tf, che calcola la frequenza dei singoli termini all'interno del nome, tramite l'utilizzo della classe `HashingTF`. Viene poi calcolata la TfIdf tramite la class `IDF` di pySpark. Il vettore generato da quest ultimo passo viene infine fornito come input all'algoritmo di clustering.

Ngrams TFIDF Consiste nell'utilizzo di pressocchè la stessa pipeline del punto precedente, con l'aggiunta dell'utilizzo della classe `Ngram` tra la classe `StopWordsRemover` e la classe `HashingTF`, per generare, a partire dalle parole originali contenute nel nome della playlist, degli n-grammi, cioè delle combinazioni di dimensione n.

Word2Vec La classe `Word2Vec` genera un modello a partire dai dati di training, cioè le parole contenute nel nome date dall'utilizzo di `Tokenizer` e `StopWordsRemover`, ed associa ad ogni parola un vettore di dimensione fissa. Il modello associa poi ad ogni nome di playlist un vettore, dato dalla media dei vettori di tutte le parole. Questo vettore viene poi utilizzato come input per l'algoritmo di clustering.

Come algoritmo di clustering ho scelto di utilizzare gli algoritmi K-means e bisecting K-means. Viene quindi effettuata la fase di training sul modello, tramite la classe `KMeans` o la classe `BisectingKMeans`, e vengono assegnati i cluster di appartenenza alle playlist del training set e del validation set. Utilizzando le playlist del training set, queste vengono raggruppate per cluster di appartenenza e, per ogni cluster, vengono calcolate le tracce più popolari. Ad ogni playlist del validation set vengono quindi assegnate le tracce più popolari all'interno del cluster corrispondente, salvandole su disco. Il sistema infine calcola e salva su disco le metriche di valutazione sulle raccomandazioni.

Capitolo 5

Risultati

In questo capitolo introduco ed analizzo i risultati ottenuti utilizzando i sistemi di raccomandazione sviluppati.

5.1 Esperimenti con versione 'V1' file attributi

I primi esperimenti sono stati effettuati utilizzando la versione V1 dei file attributi come da tabella tabella 4.2, cioè con file attributi vuoti, per testare le prestazioni dell'algoritmo sul dataset in condizioni di CF puro e scegliere l'algoritmo meglio performante fra quelli forniti da MyMediaLite, utilizzando le sottomissioni alla challenge per testare, inoltre, se le metriche calcolate dal sistema sul validation set avessero lo stesso andamento di quelle calcolate dal server remoto sul test set, a prova di una corretta implementazione della sezione di codice che le calcola.

Le raccomandazioni fornite da MyMediaLite soffrono del problema del problema rappresentato dalla presenza di nuovi utenti. MyMediaLite non è infatti in grado di fornire raccomandazioni per quegli utenti che sono sconosciuti al sistema. Come conseguenza gli algoritmi utilizzati si rivelano inefficaci per tutte quelle playlist che presentano come tracce seme un insieme vuoto, cioè `num_holdouts=num_tracks`. Per queste playlist, le raccomandazioni vengono effettuate ricorrendo all'algoritmo `MostPopular` che, banalmente, raccomanda a tutte le playlist richieste lo stesso insieme di tracce, cioè le più popolari.

Le metriche relative all'esperimento `1-P` sono state calcolate tenendo conto soltanto della porzione di validation set che non presenta tracce seme, mentre le altre metriche sono relative alla totalità del set di validazione, cioè le raccomandazioni fornite come insieme vuoto vengono sostituite dalla lista delle 500 tracce più popolari.

Experiment	Algorithm	Parameters	R-precision	NDCG	Clicks
1-P	MostPopular	0 seed tracks	0.03607	0.10896	35.6859
1-V	BPRMF	n_factors=100	0.02262	0.0578	10.482
2-V	BPRMF	n_factors=150	0.0201	0.05144	27.512
3-V	BPRMF	n_factors=200	0.01979	0.0547	24.3265
4-V	BPRMF	n_factors=300	0.01996	0.05132	26.669
5-V	SoftMarginRankMF	n_factors=100	0.00623	0.02525	26.8113
6-V	WeightedBPRMF	n_factors=100	0.03154	0.09698	2.71005
7-V	WRMF	n_factors=100	0.0498	0.14889	2.6981

Tabella 5.1: Esperimenti con attributi V1 su validation set

Experiment	Algorithm	Parameters	R-precision	NDCG	Clicks
1-T	BPRMF	n_factors=100	0.045195	0.098431	12.797
2-T	BPRMF	n_factors=200	0.044461	0.098198	12.8114
3-T	WeightedBPRMF	n_factors=100	0.107841	0.183901	6.795

Tabella 5.2: Esperimenti con attributi V1 su test(challenge) set

Come si può evincere dai dati in tabella 5.1, utilizzando l'algoritmo BPRMF ed aumentando il numero di fattori oltre i 100, si incorre in un fenomeno di overfitting dei dati di training, con un forte degradamento delle prestazioni per quanto riguarda la metrica Clicks. Il fatto che il degrado delle altre metriche non sia così marcato suggerisce un ordinamento non ottimale delle tracce in fase di output. Tale fenomeno sembra curiosamente avere un effetto meno marcato sul test set.

È interessante notare come le metriche migliorino al cambio di algoritmo ed utilizzando l'algoritmo WeightedBPRMF, che introduce un meccanismo di sampling che favorisce

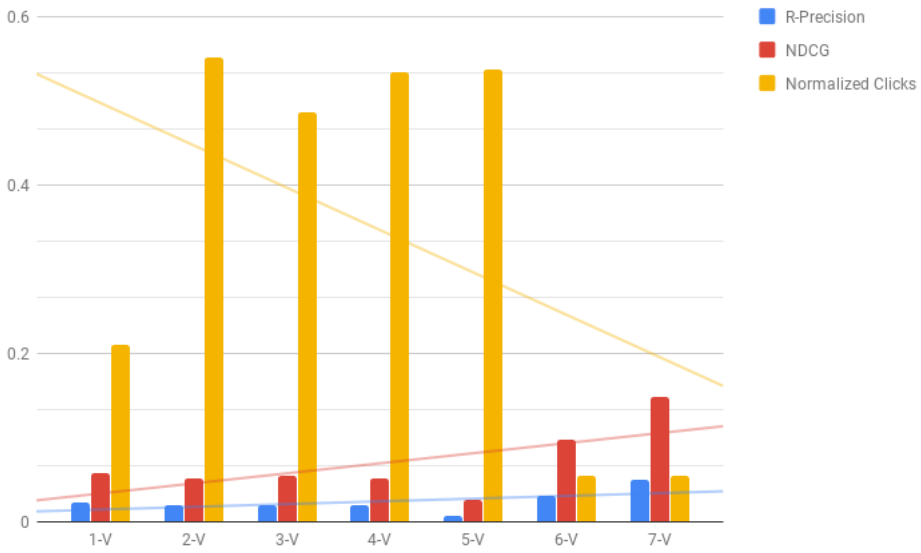

Figura 5.1: Misure per esperimento con attributi V1 - Validation Set

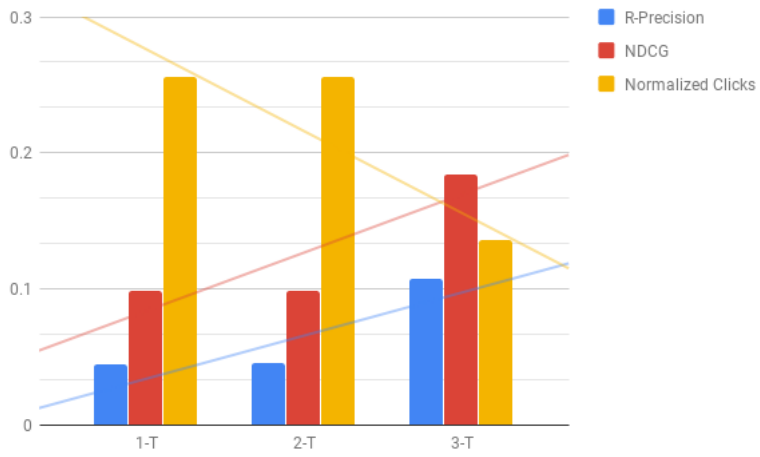

Figura 5.2: Misure per esperimento con attributi V1 - Test (challenge) Set

elementi con basso punteggio. Questo comportamento può essere dovuto al fatto che l'algoritmo BPRMF utilizza come metrica per l'ordinamento degli oggetti il numero di interazioni. Nel campo delle raccomandazioni musicali non risulta sempre ottimale consigliare le tracce o gli album più popolari. Ciò sembra essere vero anche nel caso in esame, in quanto quelli che per il recommender sono utenti, sono le playlist del dataset ed in molti casi

le playlist sono incentrate su un tema. Variando il meccanismo di sampling, l'algoritmo WeightedBPRMF consiglia tracce con meno interazioni, quindi meno conosciute, il che ha come effetto collaterale, apparentemente, di consigliare tracce più in linea con il tema delle playlist.

L'algoritmo WRMF, che utilizza la tecnica degli Alternating Least Squares (ALS) per l'apprendimento, ottiene risultati migliori in tutte e tre le metriche(fig. 5.1), ma non è stato approfondito in quanto necessita di scandire l'intero dataset ad ogni iterazione e si è dimostrato computazionalmente inusabile, con un tempo di esecuzione di quasi una settimana.

5.2 Esperimenti con versione 'V2' file attributi

La seconda sezione di esperimenti è stata effettuata utilizzando la versione V2 dei file attributi, cioè aggiungendo le informazioni sulla relazione fra la traccia, l'artista e l'album. In questa istanza gli esperimenti effettuati miravano al miglioramento delle prestazioni dell'algoritmo WeightedBPRMF. Dal test `4-T` della tabella 5.4 sembra quindi essersi presentato, all'aggiunta degli attributi, un fenomeno di underfitting. Aumentando il numero di fattori, infatti, le performance dell'algoritmo aumentano nuovamente. Aumentando il numero di fattori da 200 a 300 non si riscontra un aumento significativo delle performance sulle metriche.

Ho pensato, quindi di tentare una messa a punto dei vari parametri di regolarizzazione degli utenti e degli oggetti, rispettivamente `reg_u` e `reg_i`, per capire l'effetto della loro variazione sulle metriche e diminuire l'indice di apprendimento, cioè il parametro `LearnRate`. Trattandosi di il WeightedBPRMF di un algoritmo che utilizza come tecnica di apprendimento una SGD, la scelta di diminuire l'indice di apprendimento a 0.03, invece del valore di default di 0.05, è stata effettuata con lo scopo di ottenere un'approssimazione migliore dell'andamento veritiero dei dati. L'esperimento 10-V mostra infatti un miglioramento delle misure, che non si è però riscontrato anche nell'esperimento 7-T. A questo punto ho deciso di modificare il numero di iterazioni effettuate dall'algoritmo, cioè il parametro `num_iter`, dato che la discrepanza di andamento tra i risultati dell'esperimento 10-V e l'esperimento 7-T potrebbe essere stata causata da un numero insufficiente di iterazioni. Gli esperimenti

da 11-V a 20-V sono il risultato dell'aggiunta al comando dei parametri `--num-iter=N`, che imposta il numero iniziale di iterazioni da effettuare, `--max-iter=N`, che imposta il numero massimo di iterazioni da effettuare, e `--find-iter=N`, che imposta ogni quante iterazioni l'algoritmo deve salvare le raccomandazioni.

Durante l'esecuzione del comando, sono state chiuse le sottomissioni alla challenge, per cui non è stato possibile ottenere un riscontro anche sul test set. I risultati riportati in tabella 5.4 come esperimento FIN-T sono le metriche calcolate dal sistema di sottomissione utilizzando l'intero set delle soluzioni, invece che solo una parte, come da specifiche(sezione 3.1.3).

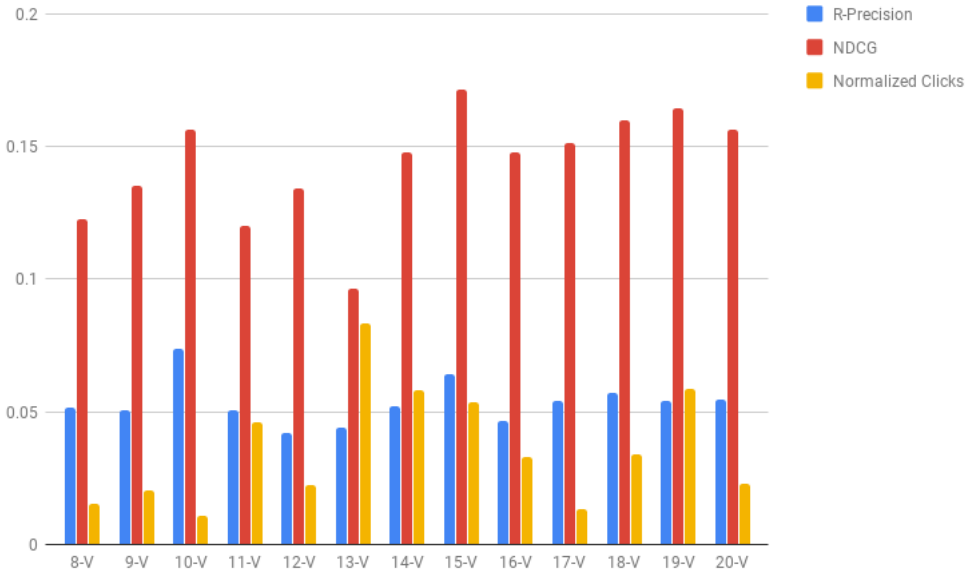

Figura 5.3: Misure per esperimento con attributi V2 - Validation Set

Experiment	Algorithm	Parameters	R-precision	NDCG	Clicks
8-V	WeightedBPRMF	num_factors=200	0.05171	0.12228	0.7644
9-V	WeightedBPRMF	num_factors=300	0.05062	0.13498	1.0219
10-V	WeightedBPRMF	num_factors=300 LearnRate=0.03 reg_i=0.003 reg_u=0.003	0.07346	0.15611	0.5302
11-V	WeightedBPRMF	num_factors=300 num_iter=15	0.05038	0.11981	2.2925
12-V	WeightedBPRMF	num_factors=300 num_iter=20	0.04199	0.13436	1.1053
13-V	WeightedBPRMF	num_factors=300 num_iter=25	0.04375	0.09648	4.1683
14-V	WeightedBPRMF	num_factors=300 num_iter=30	0.052	0.1477	2.8941
15-V	WeightedBPRMF	num_factors=300 num_iter=35	0.0641	0.1715	2.6778
16-V	WeightedBPRMF	num_factors=300 num_iter=40	0.04669	0.1475	1.6566
17-V	WeightedBPRMF	num_factors=300 num_iter=45	0.054	0.1512	0.6527
18-V	WeightedBPRMF	num_factors=300 num_iter=50	0.05709	0.16004	1.7028
19-V	WeightedBPRMF	num_factors=300 num_iter=55	0.05394	0.1641	2.937
20-V	WeightedBPRMF	num_factors=300 num_iter=60	0.05466	0.1561	1.1311

Tabella 5.3: Esperimenti con attributi V2 su validation set

Experiment	Algorithm	Parameters	R-precision	NDCG	Clicks
4-T	WeightedBPRMF	n_factors=100	0.0035	0.008429	47.7178
5-T	WeightedBPRMF	n_factors=200	0.113835	0.193081	6.6104
6-T	WeightedBPRMF	n_factors=300	0.118044	0.19957	6.4832
7-T	WeightedBPRMF	n_factors=300 LearnRate=0.03 reg_i=0.003 reg_u=0.003	0.116735	0.19715	7.4658
FIN-T	WeightedBPRMF	n_factors=300 LearnRate=0.03 reg_i=0.003 reg_u=0.003	0.1158	0.1961	7.5041

Tabella 5.4: Esperimenti con attributi V2 su test (challenge) set

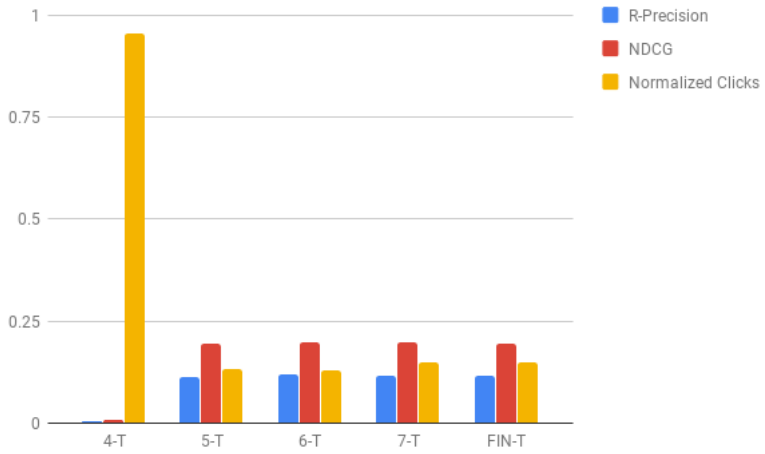

Figura 5.4: Misure per esperimento con attributi V2 - Test (challenge) Set

5.3 Risultati esperimenti con Spark MLlib

Affrontare il problema della partenza a freddo utilizzando i metodi scelti (sezione 4.4) si riduce alla scelta della dimensionalità del vettore delle feature e della dimensione dei cluster.

Le playlist del dataset hanno circa 93000 nomi diversi, risultato della combinazione di circa 24000 token. Per ridurre i tempi di calcolo, ho utilizzato Word2Vec per costruire un modello a dimensionalità ridotta del corpus dei messaggi. Non essendo il corpus stesso di grande dimensione, ho deciso di impostare come dimensione del vettore di uscita un valore di 30 ed utilizzare i nomi così trasformati per mettere a punto il numero di cluster, con l'obiettivo di migliorare le metriche ottenute nell'esperimento 1-P (tabella 5.1). Il numero di cluster, in questo caso, deve essere tale che la totalità delle playlist che li compongono contenga almeno 500 tracce distinte e tale da poter raggruppare il training set in maniera significativa per quanto concerne i gusti degli utenti. A tal scopo ho iniziato con un numero di cluster relativamente basso, se rapportato al dataset (esperimento C-1 tabella 5.5). L'aumento del numero di cluster porta anche ad un aumento delle metriche, probabilmente dovuto al fatto che con un maggior numero di cluster si hanno un minor numero di elementi per cluster, quindi una maggiore granularità nella divisione per nomi.

Utilizzare per il clustering l'algoritmo Bisecting K-Means non porta a miglioramenti delle metriche, mentre si ha un miglioramento significativo utilizzando come vettore di input per l'algoritmo di clustering un vettore contenente le TF-IDF dei token, quindi utilizzando interamente le informazioni sui nomi. Ciò è fattibile, a livello computazionale, grazie al basso numero di token totali del corpus. Di contro, è interessante notare come i risultati peggiorino in maniera significativa utilizzando n-grammi sulle parole che compongono i nomi delle playlist. Ciò potrebbe essere causato dal fatto che, essendo il numero dei token totali non molto alto ed essendo i nomi delle playlist composti da poche parole, utilizzare n-grammi introdurrebbe del rumore. Inoltre, nel caso in esame, i nomi delle playlist sono composti da poche parole, rendendo quindi fattibile soltanto la composizione di digrammi. Di conseguenza i digrammi risultanti sono, in alcuni casi, i nomi stessi delle playlist di partenza, rendendo quindi inutile il passaggio precedente di tokenizzazione.

Experiment	Algorithm	Parameters	R-precision	NDCG	Clicks
C-1	Word2Vec + K-Means	vector_size=30, num_clusters=1008, max_iter=25	0.02064	0.00983	40.902
C-2	Word2Vec + K-Means	vector_size=30, num_clusters=2008, max_iter=25	0.03689	0.01696	37.726
C-3	Word2Vec + K-Means	vector_size=30, num_clusters=3018, max_iter=25	0.05092	0.02344	36.359
C-4	Word2Vec + K-Means	vector_size=30, num_clusters=4018, max_iter=25	0.06685	0.03035	33.737
C-5	Word2Vec + K-Means	vector_size=30, num_clusters=5018, max_iter=25	0.08206	0.03743	31.284
C-6	Word2Vec + K-Means	vector_size=30, num_clusters=6018, max_iter=25	0.09383	0.04413	30.288
C-7	Word2Vec + K-Means	vector_size=30, num_clusters=7038, max_iter=25	0.10214	0.04718	28.767
C-15	Word2Vec + Bisecting K-Means	vector_size=27, num_clusters=2959, max_iter=25	0.04624	0.02099	37.025
C-16	Word2Vec + Bisecting K-Means	vector_size=27, num_clusters=2977, max_iter=25	0.04579	0.02104	36.057
C-17	Word2Vec + Bisecting K-Means	vector_size=27, num_clusters=5035, max_iter=25	0.07271	0.03338	33.363
C-18	Word2Vec + Bisecting K-Means	vector_size=27, num_clusters=5232, max_iter=25	0.07439	0.03384	33.067

Tabella 5.5: Esperimenti per cold start su validation set con Word2Vec

Experiment	Algorithm	Parameters	R-precision	NDCG	Clicks
C-8	TF-IDF + K-Means	Num_clusters=5018, max_iter=27	0.07374	0.03382	30.768
C-9	TF-IDF + K-Means	Num_clusters=6018, max_iter=27	0.08472	0.03813	29.08
C-10	TF-IDF + K-Means	Num_clusters=7018, max_iter=27	0.09765	0.04532	28.255
C-11	TF-IDF + K-Means	Num_clusters=8018, max_iter=27	0.11103	0.05094	26.67
C-12	Ngrams + TF-IDF + K-Means	ngrams_size=2, num_clusters=5018, max_iter=27	0.05771	0.02717	41.03
C-13	Ngrams + TF-IDF + K-Means	ngrams_size=2, num_clusters=6018, max_iter=27	0.05792	0.02813	41.267
C-14	Ngrams + TF-IDF + K-Means	ngrams_size=2, num_clusters=7038, max_iter=27	0.05787	0.02842	40.894

Tabella 5.6: Esperimenti per cold start su validation set con TfIDF

Capitolo 6

Analisi dei vincitori

Alla conclusione della competizione le squadre partecipanti, per essere messe in lizza per il premio finale, hanno dovuto pubblicare il codice e rilasciare un resoconto lavoro effettuato e dell'approccio utilizzato per la risoluzione della challenge. Di seguito analizzo i lavori delle squadre che hanno conquistato il podio ed i diversi approcci utilizzati.

6.1 Squadra Avito

La squadra Avito, che ha conquistato il terzo posto nella classifica finale, combina un approccio basato sul contenuto ed un approccio collaborativo in un sistema di raccomandazione ibrido basato su due fasi[30]:

1. Selezione delle tracce candidate tramite fattorizzazione di matrici.
2. Inclusione di attributi aggiuntivi e raccomandazione finale.

La fase di selezione delle tracce candidate la squadra ha utilizzato la libreria LightFM (vedi sezione 3.2.2), utilizzando la funzione di costo WARP[36], per allenare due modelli differenti.

Il primo modello, definito CS(candidate selection), è stato allenato utilizzando una semplice matrice delle interazioni utente-oggetto, in questo caso una matrice in cui ogni riga rappresenta una playlist ed ogni colonna una traccia. Ogni elemento della riga viene posto al valore 1 se la traccia corrispondente compare nella playlist, 0 altrimenti. Da questo

modello, per ogni playlist e traccia da raccomandare, viene generato un punteggio ottenuto dalla somma dei bias di playlist e traccia e del prodotto interno tra i vettori contenete gli attributi latenti di playlist e traccia.

Il secondo modello, definito CS_{text}, è stato allenato utilizzando le 2000 parole più frequenti ottenute dai nomi delle playlist. Il punteggio ottenuto per ogni playlist e traccia da raccomandare, in questo caso, il bias ed il vettore degli attributi latenti della playlist sono dati dalla somma dei bias delle singole parole e la somma dei vettori degli attributi latenti delle singole parole.

Agli attributi ottenuti tramite lo studio con LighFM, cioè lo score, i bias ed i vettori degli attributi latenti, vengono aggiunti per ogni traccia della playlist, media, mediana, minimo e massimo sulle tracce totali ed altre statistiche e attributi, tra cui il numero di artisti e album unici per la playlist ed il numero di tracce seme ed omesse.

Questo processo viene effettuato su due set di validazione e sul set di test. Dai due set di validazione, per ogni traccia candidata, si aggiunge un flag impostato ad 1 se la traccia è anche rilevante, cioè presente nelle tracce omesse, 0 altrimenti. Utilizzando l'algoritmo XGBoost[10] viene allenato un modello utilizzando il primo dei due set di validazione ed utilizzando il secondo per massimizzare l'area sotto la curva ROC. Il modello ottenuto viene utilizzato per classificare le tracce del test set e queste vengono ordinate.

L'approccio utilizzato dalla squadra Avito è un approccio che, rispetto a quello da me utilizzato, tende a dividere maggiormente il compito della raccomandazione, affidando diversi aspetti della raccomandazione a sezioni diverse.

Per quanto riguarda la parte di raccomandazioni con l'insieme di tracce seme non vuoto, è stato utilizzato un approccio simile a quello da me utilizzato, utilizzando un approccio collaborativo e consigliando un numero di tracce superiore a quelle richieste, che sono state però poi ulteriormente ordinate in base a potenziale rilevanza ottenuta tramite un modello ad albero. Interessante è come la squadra Avito ha risolto il problema della partenza a freddo, cioè quello delle playlist senza tracce seme, allenando un secondo modello ad hoc.

6.1.0.0.1 Specifiche tecniche hardware La squadra "Avito" ha eseguito codice sviluppato su una macchina Debian con processore Intel(R) Xeon(R) E5-2697 v3 e 256GB di RAM.

6.2 Squadra "hello world!"

La squadra "hello world!", che ha conquistato il secondo posto, ha invece proposto un approccio ibrido basato su un nuovo modello che prende in considerazione anche il contenuto di playlist e tracce, definito da loro *multimodal collaborative filtering* model, basato sull'utilizzo di due componenti principali[37]:

1. Un autoencoder che sfrutta le sia le proprietà delle playlist e che delle tracce per trovare una rappresentazione in termini di proprietà latenti.

2. Una rete neurale convoluzionale su caratteri (charCNN) per trovare le proprietà latenti che legano le playlist ai loro titoli.

I due modelli vengono poi combinati utilizzando una semplice combinazione lineare dei due vettori di uscita. I pesi utilizzati, detti w_{items} e w_{text}, vengono definiti dinamicamente in base al numero di oggetti $N([p, a_p])$ ed all'importanza del titolo $I(T_p)$:

$$w_{item} = \frac{N([p, a_p])}{N([p, a_p]) + I(T_p)}, w_{title} = \frac{I(T_p)}{N([p, a_p]) + I(T_p)}. \tag{6.1}$$

Utilizzando i pesi così definiti, le raccomandazioni finali vengono influenzate maggiormente dalla charCNN al diminuire delle tracce seme della playlist e maggiormente dall'autoencoder all'aumentare delle tracce seme.

La squadra "hello world!", a differenza degli approcci utilizzati dalle altre squadre e da me, non propone un recommender basato su l'utilizzo di tecniche collaborative o una combinazione di collaborative e basate sul contenuto, ma crea un modello ex novo (vedi sezione 2.6), che utilizza fortemente reti neurali, proponendo un nuovo tipo di recommender ibrido.

6.2.0.0.1 Specifiche tecniche hardware La squadra "hello world!" ha eseguito codice sviluppato in un ambiente Python Anaconda utilizzando Tensorflow su di una macchina con 4 Nvidia GTX 1080Ti.

6.3 Squadra "vl6"

La squadra "vl6", che ha conquistato il primo posto, propone un approccio in due fasi combinando diversi tipi di approcci collaborativi e tecniche di deep learning.

La prima fase utilizza un modello basato su WRMF per fornire 20.000 tracce candidate per ogni playlist. Per ogni playlist candidata vengono calcolati dei punteggi utilizzando una rete neurale convoluzionale sugli attributi latenti, un modello UserKNN(sezione 2.5.1.1) ed un modello ItemKNN(sezione 2.5.1.2). Questi punteggi, la loro combinazione lineare e gli attributi playlist-traccia estratti dell'algoritmo WRMF, vengono utilizzati come input per la seconda fase.

La seconda fase dell'algoritmo consiste nell'ordinamento delle tracce, utilizzando gli attributi ricevuti dalla prima fase, utilizzando l'algoritmo XGBoost[10] per allenare un modello ad albero. Il modello viene allenato utilizzando un massimo di 20 playlist rilevanti fra quelle candidate e 20 playlist non rilevanti.

Molto interessante è come la squadra "vl6" ha gestito la partenza a freddo, cioè le playlist con insieme di tracce seme vuoto. Hanno creato una matrice delle occorrenze dei nomi concatenando playlist e tracce. Ogni riga della matrice corrisponde ad una playlist ed una traccia e le colonne ai nomi delle playlist. Ogni riga, quindi, nel caso delle playlist, è composta da un 1 nella colonna corrispondente al nome, 0 altrimenti. Ogni raga corrispondente ad una traccia, invece, presenta un conteggio delle playlist a cui appartiene, per nome. Alla matrice così ottenuta è stata poi applicata una SVD(vedi 2.5.2.1) per ottenere una rappresentazione delle relazioni latenti fra i nomi delle playlist, le playlist stesse e le tracce.

Anche l'approccio utilizzato dalla squadra "vl6", divide il compito della raccomandazione utilizzando un sistema differente per il caso di partenza a freddo. In maniera simile al sistema da me utilizzato, anche la squadra "vl6", durante la prima fase del loro recommender, l'algoritmo di MF utilizzato fornisce un numero di tracce consigliate maggiore di quello richiesto. Nel loro caso viene ritornata una quantità di tracce di molto maggiore, che poi nella seconda vase vengono ordinate e successivamente sfoltite. Nel caso del reccommender da me utilizzato, invece, la quantità di tracce ritornate è poco maggiore del necessario, vengono solamente rimosse le tracce presenti nelle tracce seme e la lista viene troncata.

6.3.0.0.1 Specifiche tecniche hardware La squadra "vl6" ha condotto gli esperimenti su di un server Ubuntu con due processori Intel(R) Xeon(R) E5-2620, 256GB di ram ed una scheda video Nvidia Titan V.

Capitolo 7

Conclusioni

I sistemi di raccomandazione sono ormai diventati un prodotto fondamentale e necessario di qualunque azienda offra una piattaforma per la fruizione e/o condivisione di contenuto. Dove prima le tecniche di information retrieval utilizzate tramite la ricerca del contenuto da parte dell'utente erano il solo metodo di interazione con l'utente, subentrano i sistemi raccomandazione che tramite tecniche di analisi della storia pregressa dell'utente, riescono a creare un modello dei suoi gusti e diventano tramite primario per la fruizione di contenuti, mentre la ricerca diretta diventa un tramite secondario.

I sistemi di raccomandazione sono diventati particolarmente importanti per piattaforme basate su contenuti multimediali, come musica o video. Nello specifico per Spotify, piattaforma basata su contenuto audio, la continuazione automatica di playlist, focus della RecSys challenge 2018. è diventata parte fondamentale del loro servizio. La continuazione automatica di playlist è un servizio messo a disposizione da Spotify per estendere la durata delle playlist create dall'utente, introdotto sia per aumentare il tempo di permanenza degli utenti, sia perchè la piattaforma viene utilizzata mediamente in situazioni in cui non risulta ottimale selezionare una nuova playlist dopo quella appena ascoltata, ad esempio durante una festa o un viaggio.

Il mio lavoro, nell'ambito della RecSys challenge 2018, è stata la progettazione di un sistema di raccomandazione destinato alla continuazione automatica di playlist, tramite l'utilizzo della libreria MyMediaLite per la progettazione di un sistema di raccomandazione basato su approccio ibrido che prenda in considerazione le informazioni le relazioni fra

playlist e tracce, considerando anche le proprietà delle tracce stesse, e l'utilizzo della libreria MLlib di Spark per tentare di risolvere il problema della partenza a freddo utilizzando un approccio basato sul contenuto e sulle proprietà delle playlist.

L'opportunità di utilizzare il più grande dataset di playlist mai rilasciato, con un milione di playlist ed oltre due milioni di tracce, si è rivelata una grande sfida durante le fasi scelta delle libreria da utilizzare e di sviluppo con MyMediaLite, date le limitazioni in termini di memoria del sistema utilizzato.

I risultati raggiunti, per quanto non impressionanti a livello globale, li ritengo più che soddisfacenti considerando la mia scelta di partecipare ad una competizione globale di tale portata senza una squadra con cui poter costantemente confrontare idee e la strumentazione da me utilizzata. Sarebbe stato possibile migliorare i risultati modificando la struttura di raccomandazione utilizzata, utilizzando fin da subito un recommender dedicato alle playlist con insieme di tracce seme vuoto. Inoltre avrei potuto definire una misura di rilevanza degli artisti e degli album all'interno della playlist ed ordinare, per le playlist con insieme di tracce seme non vuoto, le tracce fornite in base ad una combinazione lineare delle due misure prima di troncare la lista delle tracce fornite.

Appendice A

Struttura json Million Playlist Dataset

Ogni file del MPD contiene un dizionario json strutturato come segue:

info Il campo info è a sua volta un dizionario contenente informazioni generali sulla porzione di dataset. È stato ignorato durante le fasi di analisi del MPD.

- **slice** Il campo slice indica la porzione di playlist presente all'interno del file. Deve essere uguale al range riportato nel nome del file. Ad esempio 2000-2999.
- **version** Indica la versione del MPD, il cui valore dovrebbe essere sempre `v1`.
- **generated_on** Contiene un timestamp del momento in cui la slice è stata creata.

playlists Il campo playlists contiene un array di 1000 elementi, in cui ogni elemento è playlist. Ogni playlist è a sua volta un dizionario:

- **pid** L'identificativo della playlist all'interno del MPD. È un numero intero compreso tra `0` e `999.999`.
- **name** Contiene il nome della playlist.
- **description** Contiene la descrizione della playlist. Questo campo è opzionale e può non essere presente in alcune playlist.

modified_at Contiene un timestamp in del momento in cui la playlist è stata modificata l'ultima volta, in secondi dalla unix epoch.

num_artists Contiene il numero di artisti distinti che hanno composto le tracce contenute nella playlist.

num_albums Contiene il numero di album distinti a cui appartengono le tracce contenute nella playlist.

num_tracks Contiene il numero di tracce contenute nella playlist.

num_followers Contiene il numero di utenti che seguono la playlist, escluso il creatore.

num_edits Contiene il numero di sessioni di modifica della playlist. Una sessione consiste di una finestra di due ore all'interno della quale tutte le modifiche effettuate sono considerate una modifica unica.

duration_ms La durata totale della playlist in millisecondi.

collaborative Indica se la playlist è stata segnata dal creatore come playlist collaborativa. In questo caso più utenti sono autorizzati a modificarla.

tracks Contiene un array con le informazioni sulle tracce contenute nella playlist. Ogni elemento di questo array è a sua volta un dizionario:

track_name Contiene il nome della traccia

track_uri Contiene l'URI di Spotify della traccia. È una stringa di formato `spotify:track:{id}`

album_name Contiene il nome dell'album a cui la traccia appartiene.

album_uri Contiene l'URI di Spotify dell'album a cui la traccia appartiene. È una stringa di formato `spotify:album:{id}`

artist_name Contiene il nome dell'artista principale che ha composto la traccia.

artist_uri Contiene l'URI di Spotify dell'artista principale che ha composto la traccia. È una stringa di formato `spotify:artist:{id}`

duration_ms Contiene la durata in millisecondi della traccia.

pos Contiene la posizione della traccia nella playlist. Le posizioni partono da zero.

Appendice B

Struttura json Million Playlists Challenge Set

All'interno del file è contenuto un dizionario json composto da tre campi:

date La data in cui il challenge set è stato generato.

version La versione del challenge set.

playlist Un array contenente le 10.000 playlist incomplete della challenge. Ogni playlist è a sua volta un dizionario:

- **pid** Contiene l'identificativo della playlist.
- **name** Contiene il nome della playlist. Si tratta di un parametro opzionale e può essere assente in alcune playlist.
- **num_holdouts** Contiene il numero di tracce che sono state omesse dalla playlist.
- **num_samples** Contiene il numero di tracce seme, cioè il numero di tracce che non sono state omesse dalla playlist.
- **num_tracks** Contiene il numero totale di tracce all'interno della playlist.
- **tracks** Un array contenente la lista delle tracce contenute nella playlist. Tale array può essere vuoto. Ogni traccia contenuta nell'array è costituita come nel MPD(vedi appendice A).

Bibliografia

[1] *Telegram Bot Python library.* Available at: https://github.com/python-telegram-bot/python-telegram-bot.

[2] *C# Language Specification, 5th edition*, December 2017. https://www.ecma-international.org/publications/standards/Ecma-334.htm.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[4] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, March 2012.

[5] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform resource identifier (uri): Generic syntax. Technical report, 2004.

[6] Robin Burke. Knowledge-based recommender systems. In *ENCYCLOPEDIA OF LIBRARY AND INFORMATION SYSTEMS*, page 2000. Marcel Dekker, 2000.

[7] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, Nov 2002.

[8] Òscar Celma. *The Recommendation Problem*, pages 15–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. Recsys challenge 2018: Automatic music playlist continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 527–528, New York, NY, USA, 2018. ACM.

[10] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery*

and Data Mining, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[11] Microsoft Corporation. *C# Garbage Collector reference*. https://docs.microsoft.com/it-it/dotnet/standard/garbage-collection/memory-management-and-gc.

[12] Microsoft Corporation. *Common Language Runtime reference*. https://docs.microsoft.com/it-it/dotnet/standard/clr.

[13] Microsoft Corporation. *Language Integrated Query reference*. https://docs.microsoft.com/it-it/dotnet/standard/using-linq.

[14] Cui, Bei-Bei. Design and implementation of movie recommendation system based on knn collaborative filtering algorithm. *ITM Web Conf.*, 12:04008, 2017.

[15] Christian Desrosiers and George Karypis. *A Comprehensive Survey of Neighborhood-based Recommendation Methods*, pages 107–144. Springer US, Boston, MA, 2011.

[16] Nikhilesh Dholakia, Ruby Roy Dholakia, and Nir Kshetri. Internet diffusion. 2003.

[17] Python Software Foundation. *Python Language Reference, version 3.5*. Available at: https://docs.python.org/3.5/.

[18] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. MyMediaLite: A free recommender system library. In *Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011)*, 2011. https://www.ismll.uni-hildesheim.de/pub/pdfs/Gantner_et_al2011_MyMediaLite.pdf.

[19] Nicolas Hug. Surprise, a Python library for recommender systems. 2017.

[20] Fatima EL Jamiy, Abderrahmane Daif, Mohamed Azouazi, and Abdelaziz Marzak. The potential and challenges of big data-recommendation systems next level application. *arXiv preprint arXiv:1501.03424*, 2015.

[21] V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, April 1980.

[22] Maciej Kula. Metadata embeddings for user and item cold-start recommendations. In Toine Bogers and Marijn Koolen, editors, *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.*, volume 1448 of *CEUR Workshop Proceedings*, pages 14–21. CEUR-WS.org, 2015.

[23] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural*

Information Processing Systems 13, pages 556–562. MIT Press, 2001.

[24] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[25] Michael I Mauldin. Lycos: Design choices in an internet search service. *IEEE Expert*, 12(1):8–11, 1997.

[26] The million playlist dataset was built by the following researchers at Spotify: Ching-Wei Chen Cedric De Boom Jean Garcia-Gathright Paul Lamere James McInerney Vidhya Murali Hugh Rawlinson Sravana Reddy Romain Yon. The million playlists dataset. 2018. Million Playlist Dataset, official website hosted at https://recsys-challenge.spotify.com/.

[27] MLlib. *MLlib documentation*. https://spark.apache.org/docs/latest/mllib-guide.html.

[28] Juan Ramos. Using tf-idf to determine word relevance in document queries.

[29] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.

[30] Vasiliy Rubtsov, Mikhail Kamenshchikov, Ilya Valyaev, Vasiliy Leksin, and Dmitry I. Ignatov. A hybrid two-stage recommender system for automatic playlist continuation. In *Proceedings of the ACM Recommender Systems Challenge 2018*, RecSys Challenge '18, pages 16:1–16:4, New York, NY, USA, 2018. ACM.

[31] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[32] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[33] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[34] Guy Shani and Asela Gunawardana. *Evaluating Recommendation Systems*, pages 257–297. Springer US, Boston, MA, 2011.

[35] Shahab Saquib Sohail, Jamshed Siddiqui, and Rashid Ali. Classifications of recommender systems: A review. *Journal of Engineering Science & Technology Review*, 10(4), 2017.

[36] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770, 2011.

[37] Hojin Yang, Yoonki Jeong, Minjin Choi, and Jongwuk Lee. Mmcf: Multimodal collaborative filtering for automatic playlist continuation. In *Proceedings of the ACM Recommender Systems Challenge 2018*, RecSys Challenge '18, pages 11:1–11:6, New York, NY, USA, 2018. ACM.