Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Corso di laurea in Ingegneria Informatica

Realizzazione di un sistema di raccomandazione in Matlab

Relatore: Prof. Paolo Cremonesi
Correlatore: Prof. Roberto Turrin

Tesi di laurea di
Niccolò Olivieri Achille
Matr. 724811

Anno Accademico 2009/2010

A mio padre

Sommario

Al giorno d'oggi la sempre più vasta domanda di contenuti audiovisivi specifici sta obbligando i vari distributori di servizi quali *Video on Demand* e *IP Television* a utilizzare algoritmi di raccomandazione per aiutare l'utente nella fase di scelta.

Nel corso degli anni è stata creata una gran quantità di algoritmi in grado di effettuare una raccomandazione e quest'enorme offerta ha reso la vita incredibilmente difficile a tutti coloro che han dovuto tenere aggiornate le varie piattaforme di suggerimenti e test sull'efficacia degli stessi. Ed è proprio questa fase di test che più preoccupa in quanto è computazionalmente molto impegnativa.

La ricerca della *semplificazione* e dell'*estendibilità* è l'obiettivo che ci si pone in questa tesi: per un momento si vuole interrompere la ricerca di nuovi algoritmi o nuove metodologie di test, in favore di una reingegnerizzazione di tutto il sistema di raccomandazione, rendendolo più leggibile anche ai non addetti ai lavori e soprattutto facilmente estendibile per i nuovi algoritmi che verranno sicuramente ideati negli anni a venire.

Il tutto viene realizzato appoggiandosi a quell'incredibile strumento che è Matlab, in modo da tralasciare l'ideazione di strutture dati complesse, lasciando tutta la complessità al linguaggio di programmazione. Matlab, infatti, implementa delle funzioni in grado di calcolare in maniera molto rapida complesse strutture matematiche, che altrimenti richiederebbero una fase di programmazione molto più lunga.

Ringraziamenti

Che dire, anche la laurea specialistica è conclusa, e questo lavoro di tesi segna la fine della mia avventura universitaria. Ripenso a tutti coloro che hanno contribuito a questo "successo" ed è con grande piacere che li voglio ricordare.

In primis mi sento di *dover* ringraziare i miei compagni di peripezie, **Cecio**, **Luca** e **Andre**, in quanto mi hanno sopportato e supportato in tutti questi anni dentro e fuori le mura del Poli. Inutile dire che senza le giornate di studio collettivo "alba-tramonto" non avrei neanche raggiunto il traguardo della laurea triennale. Un ringraziamento particolare va senz'altro a Cecio che ci ha sempre fornito un pasto caldo, una casa calda (soprattutto in estate) e tanta elettricità per le sedute notturne! Ma non voglio dimenticare il sette-e-mezzo a Cervinia (e soprattutto quel che successe dopo) o gli *invisibili* di Ponza (Andre, dobbiamo ancora spiegarti come ci si ferma!). Ora basta, almeno per un paio di mesi (nel caso di Cecio *anni*) non voglio più vedervi!

Il secondo grazie va senz'altro alla mia famiglia, a **mia madre**, che seppur con minacce e grida ha sempre dimostrato fiducia nelle mie capacità (sarà stato vero amore o sfrenato desiderio di non dovermi più mantenere?), a **mia nonna** che pur di vedermi ingegnere mi ha spronato e blandito con lacrime e regali. Grazie anche a **mio padre** che ha desiderato con tutte le sue forze questa laurea e che purtroppo non è qui a condividere la mia gioia, e ai **miei fratelli** che si sono talmente impegnati a rendermi la vita impossibile in casa che ho considerato la laurea e il conseguente cambio di vita come l'affrancamento dalla caotica vita familiare.

Un ringraziamento particolarissimo va al **Professor Cremonesi**, che mi ha fatto appassionare ai sistemi di raccomandazione e che è riuscito a trovarmi una tesi "adatta" alle mie esigenze di tempi stringati, e a **Roby** che, votato alla santità, è riuscito a sopportarmi in questi mesi su Skype!

Okey, ultimati i ringraziamenti "obbligati" passiamo ora a tutte le persone che, oltre ai summenzionati tre grandi amici Dott. Ing., mi sono state vicine in questi anni. Voglio senz'altro iniziare dalla mia cugina adottiva, **Ko**, che con grande intuizione è sempre riuscita a farmi passare le notti in giro quando ero sotto esami

e a farmi perdere tempo su MSN quando dovevo studiare... ogni tanto mi chiedo perché non abbia già cestinato il suo indirizzo email e il suo numero di telefono! Deve esserci una spiegazione freudiana...

Ecco ora la mia famiglia allargata, le sorelline **Robi** e **Ro** che son riuscite a mantenermi sempre sulla retta via distraendomi dallo studio e facendomi spesso dimenticare i miei doveri: vogliamo ricordare una certa vacanza e un certo carrello della spesa (Robi) oppure le telefonate di mezzogiorno dal Poli (Ro)?

E come dimenticare i "colleghi", gli altri *furbacchioni* che insieme a me hanno scelto il Poli come propria morte intellettuale: **Zoid**, **Tig**, **Patrick**, **Pan**, **Monzio**, **Axel**, **Max**, **Botta** e tutti gli altri che in questi anni mi han sempre spronato ad arrivare alla fine ("se ce l'ha fatta lui ce la posso fare pure io"!!).

Ci sono poi tutti gli amici con cui si è condivisa buona parte del (poco?) tempo in cui non si studiava: a cominciare da tutti coloro che ho conosciuto tramite internet e che sono un punto fermo nei miei momenti di svago: dai più recenti: **Gossip**, **Nilu**, **Vergat**, **Kim**, **Chri**, **i Dadi**, **Khitty**, **Death**, **Galac**, **Khoruss**, **Eliset**, **Chime** e **i sanremesi**, ma in fondo in fondo anche Amy, Moghe, Ares, Taki, It, Bio, Ayaion, Peppe, Marco, Ste, Maicol ecc ecc per arrivare a **la Fede**, **Viò**, **Laz**, **il Fra**, **la Vale**, **il Deb**, **Carmine**, **Marco**, **Khai**, **Arwin** e a tutto il vecchio (vecchissimo) gruppo dei GDR.

Ultimi ma non ultimi tutti i cretini del Fashion che son sempre riusciti a convincermi a fare serata il venerdì sera (e anche molti sabati): **i fratellini**, **Edo**, **Gerry**, **tutti gli ex-Zacca** (siete troppi!) e tutti gli altri che in questi anni han popolato il nostro ritrovo fisso!

E anche a tutti gli altri che ora sto omettendo ma che han popolato il mio mondo di studente poco modello ma sufficiente coscienzioso, un grazie di cuore da chi oggi è consapevole di chiudere un capitolo piacevole della sua vita per aprirne un altro ben più oneroso.

Indice

Elenco delle figure

Elenco dei codici

Capitolo 1

Introduzione

Servizi di e-commerce e di Internet Television (più brevemente IPTv) hanno in comune un grandissimo database, da poter offrire all'utente, di oggetti acquistabili nel caso di portali di e-commerce e di contenuti audiovisivi visualizzabili attraverso il Video on Demand (più brevemente VoD) per quanto riguarda le televisioni sul web. E se questa particolarità può sembrare un vantaggio, c'è anche un retro della medaglia, una faccia che questi provider si trovano a dover affrontare ogni giorno in modo sempre più disperato: informare l'utente dell'esistenza di questo enorme database, rendendone semplice la consultazione e la ricerca. Ma cerchiamo di capire meglio di cosa si sta parlando: nel caso si sia mai visitato un portale di e-commerce con l'intento di fare acquisti, si sarà poi notato che il sistema, una volta visualizzata la scheda di un oggetto, suggerisce altri oggetti che potrebbero interessare, in quanto ricercati o visualizzati da chi ha acquistato prima di noi. Senza troppi giri di parole questa è una semplice **raccomandazione**, in quanto il sito ha in memoria lo storico dei suoi clienti e lo mette a nostra disposizione per consigliarci un ulteriore oggetto nella speranza che ci possa in qualche modo interessare.

Similmente funziona l'IPTv. In questo caso però si ha a che fare con risorse audiovisive che quindi devono sottostare ai gusti dell'utente. del quale il sistema registra lo storico[1] per poi suggerire la visione di un contenuto piuttosto che di un altro. Ma è proprio in questo momento che subentra l'ostacolo più grande, tipico

[1] In seguito esamineremo meglio cosa si intende per storico

proprio dei servizi di IPTv e VoD: il sistema di interfacciamento è infinitamente più semplice e meno funzionale di quello che si ha a disposizione quando l'utente si mette davanti al computer, ora è un semplice telecomando che ci permette di controllare il flusso di dati. Scorrere quindi attraverso pagine e pagine di contenuti può risultare lungo, ma soprattutto fastidioso per l'utente che desidera avere risultati veloci.

Da questa semplice panoramica è facilmente intuibile l'utilità dei **sistemi di raccomandazione**: sono in grado di suggerire a real time un numero limitato ma incredibilmente specifico di contenuti altamente *personalizzati*. Tutto ciò è ovviamente volto a rendere sempre più soddisfatto l'utente e, conseguentemente, incrementare le vendite dei prodotti.

Un sistema di raccomandazione è in grado di consigliare determinati contenuti in base a:

- il *profilo utente*, ovvero le singole preferenze, espresse in modo più o meno esplicito;
- le caratteristiche dei contenuti stessi (*modello degli item*).

Il profilo utente è lo storico delle interazioni tra l'utente e i contenuti. Vengono memorizzate sia le visualizzazioni di un determinato oggetto del catalogo, sia la valutazione (avvenuta quindi in modo esplicito) che gli è stata attribuita. In questo modo si crea una sorta di "carta di identità" dell'utente.

Il modello dei vari item riguarda le caratteristiche dei contenuti disponibili a catalogo. In base all'algoritmo scelto queste caratteristiche riguarderanno il contenuto dell'item oppure le preferenze indicate dalla comunità degli utenti. Nel primo caso parleremo di algoritmi *Content-Based*, nel secondo di algoritmi *Collaborativi*.

Fine ultimo del sistema di raccomandazione è restituire una lista di N valori (per questo chiamata **lista Top-N** in cui N solitamente è 10). Questa lista può essere anche filtrata in modo da controllarne la dinamicità in vari momenti.

In questa tesi si vuole quindi creare una piattaforma in grado di utilizzare i sistemi di raccomandazione non solo per generare i suddetti "consigli", ma anche per valutarne la bontà attraverso opportuni test. Il tutto considerando che

la tecnologia progredisce e nuovi algoritmi o metodi di test possono essere creati. Proprio per questo sono state studiate e verranno presentate alcune comode API per interfacciarsi ed estendere la piattaforma.

Il lavoro è suddiviso in nove capitoli.
Nel Capitolo due viene presentata una breve carrellata dei vari algoritmi utilizzati per le raccomandazioni e le tipologie di dati utilizzate da questi. Ci si soffermerà in particolar modo sulle basi della raccomandazione, in modo da capire poi il processo di reingegnerizzazione.

Nel Capitolo tre saranno trattate le metodologie di test utilizzate, soffermandosi sui risultati che forniscono.

Nel Capitolo quattro si illustrerà lo stato dell'applicazione prima della reingegnerizzazione, trattando in particolar modo la sua frammentazione sia dal punto di vista del codice, sia dal punto di vista dell'ideazione in modo da presentare meglio il problema che si è incontrato. Da qui la necessità di fermarsi nello sviluppo della piattaforma in modo da riorganizzare il lavoro.

Nel Capitolo quinto si tratterà la reingegnerizzazione vera e propria del sistema, analizzandone la struttura ma soprattutto l'idea di fondo che ne permette l'estendibilità.

Nel Capitolo sei si studierà la possibile integrazione in un sistema reale, mostrandone i vantaggi e anche dimostrando perché non possa considerarsi un qualcosa di assolutamente perfetto ed eterno.

Nel Capitolo sette infine si tireranno le somme di quanto è stato studiato, analizzando i possibili sviluppi futuri che l'applicazione può avere.

Capitolo 2

Algoritmi di Raccomandazione

In questo capitolo vengono presentati gli algoritmi usati per effettuare una raccomandazione, introducendo prima cosa sia nella pratica un sistema di raccomandazione e quali dati utilizzi.

2.1 Introduzione

I sistemi di raccomandazione nascono principalmente grazie all'avvento di tecnologie in grado di permettere la fruizione di servizi personalizzati: anche solo una decina di anni fa non si sarebbe mai pensato di poter richiedere al proprio provider un determinato contenuto multimediale. Ora questo è possibile grazie principalmente all'avvento della banda larga, per di più il cliente può dettare le condizioni e quindi richiedere un servizio sempre migliore.

Ma come avviene questa distribuzione di contenuti? L'IPTv è senz'altro il miglior mezzo per trasmettere questo tipo di richieste, in quanto, essendo parente diretto di Internet, permette ad ogni utente di ricevere un flusso personalizzato. La fruizione può quindi avvenire secondo due modalità differenti, e una terza prevede l'utilizzo delle prime due unite:

Streaming il contenuto viene visionato "in diretta", mentre lo si sta scaricando.

Download il contenuto è disponibile solo dopo averlo in locale. Questa modalità è molto comoda in quanto permette di utilizzare protocolli come il Torrent

e di accedere alla risorsa anche in un secondo momento, magari in modalità offline.

Download Streaming il contenuto è visionato come in streaming, ma alla fine della visione è disponibile in locale e quindi può essere rivisto in un secondo momento.

Questa continua evoluzione delle tecnologie genera un aumento della qualità richiesta: l'elenco dei titoli disponibili presso i servizi di VoD è in continua crescita, e con esso cresce anche la complessità della ricerca del contenuto, con la complicazione che il tutto dev'essere controllato tramite un semplice telecomando. Distinguiamo due tipi di ricerca:

Information Retrieval (IR) permette la ricerca del contenuto tramite una parola chiave. Questa ricerca avviene in maniera esatta, quindi i punti di debolezza sono la parola chiave e il sistema, in quanto un contenuto può essere stato memorizzato con un nome diverso da quello conosciuto dall'utente. Inoltre, condizione necessaria per la ricerca è la conoscenza, da parte dell'utente, della risorsa stessa, e questo rende la ricerca inutile.

Information Filtering (IF) permette un filtraggio delle informazioni in base alle preferenze dell'utente. Questo sistema, nato all'inizio degli anni '90, si sta ritagliando una fetta sempre più ampia di mercato in quanto non è necessario l'intervento dell'utente nella fase di ricerca, ma solamente in quella di valutazione del risultato. È da questo tipo di ricerca che sono nati i sistemi di raccomandazione, e negli ultimi anni stanno prendendo sempre più piede sia in ambito accademico sia in ambito commerciale in quanto si è potuto vedere che consigliare all'utente un oggetto per il quale nutre un certo interesse aumenta le vendite. Questi sistemi prendono come input il *profilo utente*, ovvero lo storico dell'utente all'interno del sistema: quali contenuti (in seguito chiamati *item*) ha visualizzato, per quali ha espresso una votazione, quali gli sono già stati consigliati e così via. Inoltre ogni item è descritto tramite un insieme in informazioni (chiamate *metadati*) che lo caratterizzano (per esempio un film può essere descritto dal regista, dal genere, dagli attori e via dicendo). Grazie a queste, poche, informazioni il sistema è in grado di

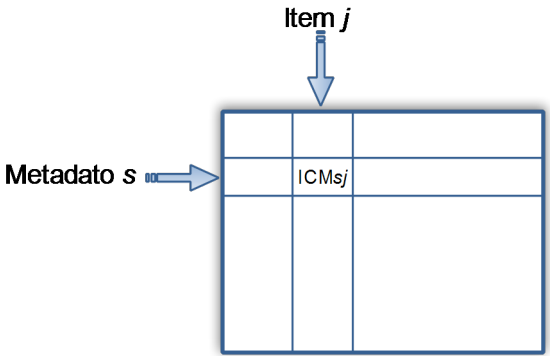

Figura 2.1: Matrice ICM

consigliare nuovi contenuti all'utente, sperando che questi suggerimenti siano di suo gradimento.

Nella trattazione verrà presa in considerazione solamente la ricerca IF in quanto quella IR non è di particolare interesse.

2.2 Dati utilizzati

Gli input utilizzati sono fondamentalmente di due tipi:

- La matrice ICM (**Item Content Matrix**) che contiene tutti i metadati dei vari item. Le colonne sono quindi gli item (in futuro sarà sempre l'indice j), mentre le righe sono le differenti parole (vedremo in seguito come si ricavano queste parole) che descrivono le proprietà di cui si vuol tenere traccia. Queste proprietà possono essere inserite o dal sistema (solitamente in fase di aggiunta dell'item) o dagli utenti collaborativamente: in questo modo non ricade tutto sulle spalle dell'azienda che mette a disposizione il servizio. Per contro, le informazioni potrebbero non essere sufficientemente precise in quanto la valutazione dell'utente è sempre soggettiva. Questa matrice viene utilizzata solamente negli algoritmi *Content-Based*.

- La matrice URM (**User Rating Matrix**) contiene lo storico di tutti gli utenti presenti nel sistema. Le righe rappresentano gli utenti (indicati con l'indice i), mentre le colonne rappresentano (come visto per la matrice ICM)

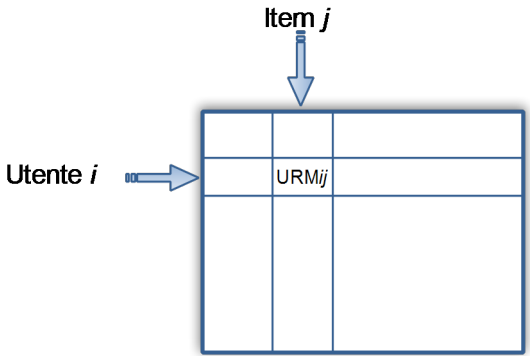

Figura 2.2: Matrice URM

gli item *j*. Ogni elemento della matrice sarà quindi la votazione che l'utente *i* ha dato all'item *j*. Questa tipologia di matrice si usa negli algoritmi *Collaborativi*.

Profilo Utente è una riga di questa matrice: rappresenta le preferenze di un utente.

Come output il sistema di raccomandazione può restituire due liste differenti:

Individual Scoring ovvero la votazione che l'utente darebbe ai vari item. In base all'algoritmo scelto questa votazione può essere assoluta o relativa.

Top-N Recommendation è l'elenco degli N item più vicini ai gusti dell'utente. Questa lista viene solitamente creata mettendo in ordine decrescente la lista precedente.

Ma come viene costruita la matrice URM? Nei paragrafi successivi si tratterà la raccolta dei vari rating.

2.2.1 Rating Espliciti

Quando un utente è consapevole di dare un voto a un contenuto si dice che questo voto è stato dato in modo esplicito. Solitamente si usa una scala numerica, per esempio tra 1 e 5 oppure tra -2 e $+2$. Il valore 0 significa che non vi è stata interazione tra l'utente e il sistema per quel determinato item. I valori più bassi

(intesi come inferiori rispetto a una soglia o predeterminata o calcolata a runtime) vengono considerati negativi ai fini della valutazione, mentre quelli alti positivi. Questa tipologia di rating è molto più comoda e genererà una raccomandazione molto più precisa delle altre tipologie. Sussistono però dei problemi che la rendono non sempre la più usata:

- la scala di valori varia da utente a utente in quanto è estremamente soggettiva;
- un utente può trovarsi a dare voti a caso: in questo caso risulterebbe molto più conveniente utilizzare una valutazione implicita, in quanto l'utente contribuisce solamente con la sua superficialità ad aumentare la complessità della raccomandazione;
- non tutti gli algoritmi di raccomandazione utilizzano le stesse scale, quindi può essere necessario riscalare le votazioni in modo da conformarle con quelle scelte dall'algoritmo.

2.2.2 Rating Implicito

Molto spesso il sistema non riesce ad avere delle votazioni direttamente dagli utenti, quindi si deve cercare un altro modo per far esprimere un giudizio. Il sistema utilizzato più di frequente è considerare la votazione in modo binario: 0 se l'item non è stato visto dall'utente e 1 viceversa. In questo modo si avrà un giudizio poco personale in quanto un utente potrebbe aver richiesto un item non per se stesso, oppure potrebbe averlo visto e non essergli piaciuto.

Purtroppo però questo tipo di rating è molto usato in quanto non sempre l'utente è disposto a "perdere tempo" nel dare i voti, e quindi bisogna cercare di ricavarli altrove.

2.2.3 Cambiamenti all'interno di un dataset

Un sistema di raccomandazione non rimane stazionario, ma continua a mutare nel tempo. In particolare i nostri dati possono modificarsi in 3 modi:

- un utente i visiona un contenuto j e ne dà un voto (supponendo di avere un rating esplicito): nella matrice verrà modificato il valore URM (i, j) in modo da rappresentare il nuovo voto;
- un nuovo item viene inserito nel sistema: la matrice URM avrà quindi una colonna in più, inizializzata a zero per non alterare la raccomandazione degli altri item;
- un nuovo utente comincia ad usare il sistema: la matrice URM acquisterà una nuova riga, sempre inizializzata a zero in modo da non influenzare i suggerimenti dati agli altri utenti.

In un sistema reale questi cambiamenti avvengono continuamente, basti pensare a quanti nuovi utenti giornalieri o contenuti audiovisivi possa avere un sistema complesso come YouTube. Dal momento che si vuole mantenere un certo livello di affidabilità per i suggerimenti, bisognerà mantenere aggiornato il modello su cui si basano tutte le raccomandazioni. Questo è però un calcolo molto complesso dal punto di vista computazionale, quindi si dovrà arrivare a un compromesso tra efficienza ed efficacia.

2.3 Composizione di un Sistema di Raccomandazione

Il sistema deve essere sempre pronto a rispondere se interrogato, un utente non accetterebbe mai di dover aspettare che il sistema finisca un calcolo prima di servirlo. Proprio per questo viene organizzato in tre fasi:

Batch è la fase di creazione del modello: è impensabile lavorare ogni volta su tutti i dati a disposizione, quindi si genera a intervalli regolari un modello che possa poi permettere la costruzione della raccomandazione vera e propria. Questa fase è molto onerosa dal punto di vista computazionale e proprio per questo non viene effettuata ogni volta che cambiano i dati, ma solo quando il sistema è poco carico. Il tempo che intercorre tra le due creazioni di un modello deve tener conto della velocità con cui cambiano i dati: sistemi come YouTube non

possono permettersi di ricalcolare il modello una volta al mese in quanto i dati cambiano con una frequenza impressionante, anche se questo significa avere continuamente a che fare con enormi matrici. Fortunatamente queste matrici URM sono incredibilmente sparse in quanto l'utente interagisce solamente con una ridottissima parte del sistema, e quindi possono essere conservate in strutture dati più efficienti.

Real-Time una volta creato il modello il sistema può essere interrogato per avere una raccomandazione. Quest'operazione è molto più snella e deve anche essere molto più veloce, in quanto con una risposta tardiva l'utente potrebbe perdere fiducia nel sistema. La raccomandazione viene quindi calcolata prendendo come input il modello da una parte e il profilo utente dall'altra. Sta poi all'algoritmo essere abbastanza robusto da poter accettare anche utenti con un profilo aggiornato rispetto a quanto registrato nel modello oppure utenti nuovi e quindi con un profilo che non compare nel modello: si partirà dalle nuove informazioni fornite per generare l'opportuna lista di suggerimenti.

Antireshuffling interviene dopo che il sistema ha generato la raccomandazione: in diversi momenti il profilo utente preso in considerazione cambia in quanto possono essere state aggiunte delle valutazioni, o modificate altre. In questo caso l'utente si vedrà suggerire una lista leggermente diversa che può presentare dei problemi: da un lato una lista troppo simile alla precedente può non soddisfare l'utente perchè c'è il rischio che non ne evidenzi le novità sia riguardanti il profilo utente sia l'evoluzione del modello), dall'altro invece raccomandare un elenco di item troppo diverso da quello precedente potrebbe essere interpretato come se il sistema estraesse gli elementi da raccomandare con casualità e conseguente scarsa attendibilità. È necessario quindi riprocessare la nuova lista cercando di renderla né troppo dinamica né troppo statica: questo è proprio il compito dell'antireshufflin[1].

[1]Questo argomento verrà trattato molto brevemente in questo lavoro in quanto lo scopo principale non è quello di mostrare le caratteristiche degli algoritmi, ma fornire una piattaforma per sviluppare e testare tutto un sistema di raccomandazione. Per avere maggiori dettagli sull'argomento si rimanda ad altri lavori di tesi precedenti a questo.

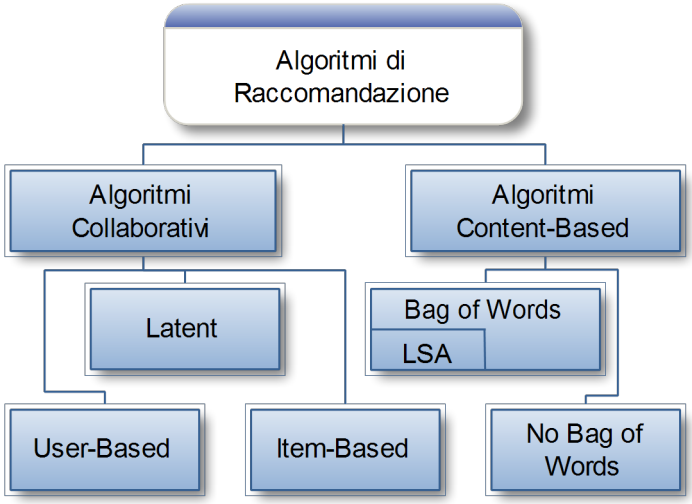

Figura 2.3: Classificazione degli algoritmi di Raccomandazione

2.4 Algoritmi di Raccomandazione

Una raccomandazione può essere effettuata in vari modi, tenendo conto di fattori diversi o, più semplicemente, ascoltando pareri differenti. Anche nella vita quotidiana si sa che un suggerimento non è mai univoco: diverse persone possono ragionare in modo diverso, e quindi condurre a conclusioni differenti. Identico è il mondo dei sistemi di raccomandazione, dove non c'è mai un suggerimento perfetto e assoluto, ma tanto modi diversi per arrivare a un parere. Questi modi diversi sono i vari **algoritmi di raccomandazione**.

Ogni algoritmo è caratterizzato da un certo input, un certo output, ma soprattutto un differente modo di utilizzare i dati per creare il proprio modello. Proprio per questo si vedrà che vi sono modelli più o meno accurati (verranno poi presentate anche le metriche per valutare quest'accuratezza).

Di seguito verranno analizzate le diverse famiglie di algoritmi, con un occhio particolare a quelle utilizzate nel seguito della trattazione per costruire la piattaforma desiderata.

2.4.1 Algoritmi con modello nello spazio latente

Molto spesso memorizzare tutte le preferenze degli utenti o tutte le caratteristiche di un item è un lavoro che richiede troppo spazio, e di conseguenza anche manipolare questi dati è troppo gravoso dal punto di vista computazionale. Proprio per questo si cerca di diminuire lo spazio utilizzato, avvalendosi del cosiddetto spazio latente. Gli algoritmi che ne fanno uso descrivono il dataset grazie a un insieme ridotto di caratteristiche, dette *features*, ricavate attraverso una decomposizione della matrice di partenza. Queste *features* possono rappresentare:

- l'interazione implicita tra l'utente e l'item nel caso di algoritmi collaborativi (in questo caso si attua una decomposizione della matrice URM);
- il contenuto implicito di ogni item, ovvero gli stem che rappresentano ogni elemento del sistema, nel caso di algoritmi Content-Based (ora invece si ha una decomposizione della matrice ICM).

Grazie a questo sistema è anche possibile ridurre notevolmente il rumore generato dall'algoritmo in quanto vengono eliminati i dati meno significativi ai fini della raccomandazione. Una delle tecniche per generare questo spazio latente è la decomposizione matematica SVD (Singolar Value Decomposition). Prende come input la matrice M di dimensioni $a \times b$ e la dimensione latente l, che rappresenta il numero di *features* che si vogliono considerare. Il risultato della decomposizione sono tre matrici:

U è una matrice unitaria di dimensioni $a \times a$;

S è una matrice diagonale di dimensioni $l \times l$ che rappresenta, attraverso valori sempre positivi, le *features*, ovvero i valori singolari della matrice M di partenza;

V' è la trasposta coniugata di una matrice unitaria di dimensioni $b \times b$.

In questo modo quindi si rafforzano i legami tra elementi simili della matrice di partenza, e si riescono anche a trovare dipendenze nascoste tra i diversi elementi.

2.4.2 Algoritmi Content-Based

Si basano sull'analisi delle proprietà dei diversi item, quindi all'utente verranno proposti sempre contenuti simili a quelli che egli stesso ha valutato positivamente. Rappresenta sicuramente il metodo più semplice e veloce per effettuare una raccomandazione. Esiste anche un retro della medaglia: dopo aver esaminato tutti i contenuti simili a quelli di suo gradimento, l'utente si ritroverà in sostanza tanti doppioni, o magari, come nel caso dei film, lo stesso elemento, semplicemente rivisitato o reinserito da parte del sistema. Inoltre si possono presentare delle ambiguità: tornando sempre all'esempio cinematografico, due film possono avere lo stesso regista, un cast simile e per questo essere proposti entrambi, pur essendo di due generi completamente differenti, col rischio che uno dei due non sia di nessun interesse per l'utente.

Per cercare di rimediare a questo inconveniente si può restringere l'analisi ad alcuni campi, per esempio non tenere in considerazione il regista, privilegiando altri elementi come il cast. In questo modo si possono proporre contenuti interessanti che l'utente non avrebbe né pensato né saputo ricercare.

Sicuramente si presentano molte problematiche con questo tipo di algoritmi:

- Le proprietà degli item devono essere inserite in modo preciso e leggibile dall'algoritmo (poi si vedrà come avviene il "parsing" delle stesse).
- Le proprietà possono essere sparse: in fase di inserimento sono state "sottovalutate" o ignorate talune proprietà, mentre per altri item le stesse sono ritenute molto importanti e specificate. È anche possibile che a seconda della persona fisica responsabile dell'inserimento dell'item nel sistema cambino le proprietà scelte: l'operatore A è molto scrupoloso riguardo al cast, mentre l'operatore B conosce tutte le sfumature dei generi, per questo introduce anche quelli di nicchia, sconosciuti ai più.
- Come detto in precedenza due elementi possono avere le stesse proprietà, e magari uno non è di gradimento all'utente: come fa in questo caso l'algoritmo a distinguerli e a capire quale consigliare e quale no?
- I suggerimenti sono sempre molto simili agli item già visti dall'utente: se ad esempio il fruitore non ha mai visto un film western, il sistema non arriverà

mai a consigliargli un film di quel tipo, a discapito di altri che con ogni probabilità gradirebbe di più.

- Gli utenti nuovi non riceveranno mai una raccomandazione adeguata in quanto il sistema non ha elementi per fornirgliela, mentre quelli con poche preferenze si vedranno suggeriti più volte contenuti identici. Solamente nel caso di profili più "vissuti" si avranno suggerimenti migliori.

La trattazione di questa tipo di algoritmo verterà su quello definito **diretto**. Possiamo suddividerlo in diverse fasi, ognuna con la sua funzione specifica:

Tokenizzazione è l'operazione che prende come input le diverse proprietà dell'item e cerca di elaborarne dei token, in modo che possano essere analizzati più velocemente. Un token è solitamente una singola parola, ma può essere anche una serie di parole come nel caso del nome e del cognome di un attore. Quest'operazione viene effettuata per isolare i termini che in seguito verranno riconosciuti come "importanti" per la buona riuscita dell'algoritmo.

Eliminazione delle Stop Words avviene subito dopo la suddivisione in token: sono rimossi tutti quegli elementi inutili come ad esempio gli articoli, le congiunzioni e tutte quelle parole che non risultano essere importanti ai fini della raccomandazione.

Stemmizzazione modifica parole come sostantivi e aggettivi, eliminando prefissi e suffissi. In questo modo parole simili avranno la stessa valenza per l'algoritmo. Ad esempio le parole "corsa", "correre" e "corridore" saranno tutte riconducibili allo stesso *stem* "cor". Il risultato di quest'operazione è il cosiddetto BoW (**Bag Of Words**).

Assegnazione di un peso ai metadati è l'operazione successiva: si effettua un'analisi statistica per calcolare il peso che ogni stem o token ha nei confronti dell'item in cui appare: ad esempio la presenza di un cantante all'interno di un film (magari un cammeo) sarà molto meno importante rispetto alla sua presenza all'interno di un concerto. Si procede quindi al calcolo del TF-IDF (Term Frequency - Inverse Document Frequency). Il

Term Frequency è un semplice numero che rappresenta le occorrenze di una parole nelle proprietà di un item. L'Inverse Document Frequency è sempre un numero, l'inverso del numero di occorrenze di un certo token all'interno di tutto il catalogo. Entrambi i pesi possono essere normalizzati per il totale degli elementi del BoW. Da questi pesi se ne ricava uno solo, che viene usato come il valore dell'elemento della matrice ICM.

Si genera quindi uno spazio vettoriale dove gli assi sono i diversi steam che appaiono dentro alla matrice ICM. Gli item vengono rappresentati come vettori in questo spazio. Per completare la raccomandazione si deve disegnare il vettore dell'utente: viene calcolato come la somma dei voti espressi, moltiplicati per la colonna della matrice ICM (quindi per l'item) relativa. Gli item da consigliare saranno poi quelli col coseno dell'angolo compreso tra vettore utente e vettore proprio più alto.

2.4.2.1 LSA Cosine

Questo algoritmo rappresenta lo stato dell'arte per quanto riguarda quelli Content-Based. Si basa su un unico concetto base: l'aggregazione di tutte le parole che compaiono (o non compaiono) all'interno di un testo, stabilisce un insieme di vincoli di reciprocità, che determinano una similarità di significato tra le parole stesse, e permette di raggruppare quelle simili. Utilizza, come l'algoritmo Sarwar che tratteremo in seguito, la decomposizione SVD della matrice ICM.

Generazione del modello avviene applicando anzitutto una decomposizione SVD sulla matrice ICM, usando una specifica dimensione latente k[2]. Si ottengono quindi tre matrici: U_k, S_k e V_k. La nuova ICM_k, data dal prodotto $S_k * V_k'$ e che verrà usata come modello dell'algoritmo, è pulita dal rumore e integra già le diverse correlazioni tra gli stem.

Generazione del suggerimento ha come input la matrice URM, il profilo utente e il modello generato in precedenza. Ogni item viene rappresentato nello spazio creato dagli stem del modello. Quindi viene calcolata la similarità tra il vettore dell'utente e quelli degli elementi che non compaiono nel

[2]La definizione del parametro k è comunemente effettuata in modo empirico.

profilo tramite il coseno dell'angolo compreso tra i due vettori: più è alto, più l'elemento considerato si avvicina ai gusti dell'utente e quindi il rispettivo contenuto sarà raccomandato.

2.4.3 Algoritmi Collaborativi

A differenza della tipologia vista in precedenza, questi algoritmi si basano solamente sui gusti della comunità di utenti. Simulano quindi una normale richiesta di suggerimenti tra due esseri umani: se l'utente A ha un amico con gusti simili, sarà sufficiente chiedergli un consiglio per ritrovarsi con una raccomandazione conforme ai gusti di A. Si basano su due assunzioni fondamentali:

Vicinanza tra utenti in quanto utenti con gusti simili voteranno sempre gli elementi in maniera simile;

Vicinanza tra item poiché gli item "simili" saranno votati sempre in maniera simile.

Proprio per questo due contenuti sono valutati simili non tanto quando sono simili tra loro, ma quando la comunità li vota in modo simile.

A differenza degli algoritmi che si basano sul contenuto, quelli collaborativi possono portare al cosiddetto **effetto serendipity** (sorpresa): mentre nel primo caso l'utente comprende sempre il perché di una raccomandazione, ora può rimanere stupito dalla stessa, in quanto gli può essere consigliato un elemento fuori dai suoi schemi, ma che è stato votato da soggetti dai gusti simili ai suoi. Proprio qui ha origine uno dei vantaggi maggiori degli algoritmi collaborativi, ovvero il non basarsi su altre informazioni se non i giudizi della comunità.

Certamente sussistono anche molti limiti a questo tipo di approccio:

- quando un nuovo utente entra nel sistema non può essere associato a nessun altro utente, quindi non avrà a disposizione una raccomandazione;
- all'aggiunta di un nuovo item, esso dovrà essere anzitutto votato da qualcuno in quanto, altrimenti, non verrà associato a nessun item preesistente e quindi non verrà raccomandato;

- l'accuratezza della raccomandazione è legata alla sparsità della matrice URM: più essa è piena di informazioni, più sarà preciso il suggerimento;
- utenti con gusti particolari saranno associati con molta difficoltà ad altri utenti, e quindi riceveranno suggerimenti poco accurati.

All'interno degli algoritmi Collaborativi possiamo effettuare un'ulteriore suddivisione:

User-Based si basano sull'idea di stabilire un rapporto di similitudine tra i diversi utenti della comunità, in modo da costruire una matrice $m \times m$ (dove m è il numero degli utenti). Questi algoritmi hanno problemi per quanto riguarda le performance nella costruzione del modello, in quanto una comunità può avere decine o centinaia di migliaia di utenti iscritti. Un altro tipico problema si ha quando si vuole inserire un nuovo utente: bisogna infatti ricreare completamente il modello, sempre con le problematiche che l'utente nuovo si porta dietro (come visto prima).

Item-Based è la tipologia duale di quella precedente: ora si vuole costruire una matrice $n \times n$ (con n uguale al numero degli item) che metta in relazione i vari elementi, in modo da trovare quelli simili tra loro. Ogni elemento della matrice è quindi il grado di somiglianza tra l'elemento della riga i e quello della colonna j (i numeri sulla diagonale sono ovviamente uguali a 1). Molto spesso questi valori vengono normalizzati, in modo ad averli compresi tra -1 e 1 nel caso di rating espliciti, e tra 0 e 1 nel caso di rating impliciti. Per eseguire una raccomandazione poi è sufficiente moltiplicare il vettore del profilo utente per la matrice, in modo da ricavare un nuovo vettore con i voti dei singoli item. Una modifica a questo algoritmo è la cosiddetta KNN, in cui la matrice modello viene prima processata ponendo a 0 gli elementi più bassi, lasciandone quindi soltanto K. Questo avviene in quanto gli elementi eliminati non influirebbero nella lista finale, ma creerebbero solamente un rumore che quindi andrebbe a risultare fastidioso per la classifica degli altri elementi.

Per quanto riguarda le prestazioni, la creazione del modello negli algoritmi Item-Based sarà più rapida in quanto, solitamente, il numero di utenti è di gran

lunga superiore a quello di contenuti. Inoltre influisce anche la frequenza con cui un utente o un elemento vengono introdotti nel sistema: anche in questo caso, generalmente, gli utenti nascono con più frequenza dei nuovi elementi[3]. Inoltre l'avere un modello Item-Based risulta essere anche comodo quando si aggiunge un nuovo elemento al sistema: dal momento che questo, prima di essere raccomandato, dovrà raccogliere un certo numero di voti, non è necessario che il modello venga ricreato istantaneamente, quindi non c'è la frenesia che ci sarebbe per il modello User-Based dove ad ogni aggiunta bisogna assolutamente creare un nuovo modello.

2.4.3.1 Item-Item Cosine KNN

Rientra nella categoria degli algoritmi Item-Based. Ogni elemento del sistema è visto come un vettore nello spazio degli utenti: la similitudine tra due vettori è rappresentata come il coseno dell'angolo tra essi compreso:

$$sim(i,j) = cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{\|\vec{i}\|^2 \times \|\vec{j}\|^2} \tag{2.1}$$

Questo coefficiente sarà uguale a 1 nel caso in cui due elementi siano uguali, e a 0 nel caso due elementi non abbiano voti in comune.

Nella fase di raccomandazione, invece, il calcolo da effettuare sarà molto più semplice:

$$Raccomandazione = ProfiloUtente \cdot Modello \tag{2.2}$$

Questa sarà una lista di voti: nel caso di dataset esplicito si avrà la predizione del voto che l'utente avrebbe dato a quello specifico elemento, mentre nel caso di un dataset implicito sarà la probabilità con cui l'utente potrebbe aver visto quel contenuto ma non aver dato una votazione. In seguito il sistema ordinerà questi voti in ordine decrescente e ne estrarrà i primi N, generando quindi la tanto attesa Top-N da suggerire all'utente stesso.

Il parametro KNN (*K Nearest Neighbor*) indica il numero di elementi da tenere in considerazione in fase di creazione del modello, quando vengono trovati i coef-

[3]Tutte queste ipotesi valgono in condizioni standard: sistemi come YouTube **non** sono standard visto che secondo l'ultima stima ogni minuto vengono caricate 35 ore di contenuti audiovisivi.

ficienti di similarità tra i diversi item. Si vanno infatti a utilizzare solamente i K elementi più vicini a quello in fase di analisi. L'eliminazione di tutti gli altri elementi si basa sulla convinzione che essi generino solamente rumore. Questo parametro ha il suo fondamento nella dimensione del dataset: più è piccolo più sarà precisa la raccomandazione, ma un parametro troppo piccolo rispetto alla matrice potrebbe eliminare troppo rumore, e quindi danneggiare la raccomandazione stessa che andrebbe a consigliare solamente elementi troppo vicini a quelli di partenza. Quando K tende a infinito, l'algoritmo perde la sua proprietà KNN e si definisce solamente Item-Item Cosine.

2.4.3.2 Item-Item DR

Questo algoritmo (dove DR sta per Direct Relation) è simile al precedente. L'unica differenza è che la similarità tra due elementi della matrice viene calcolata come il numero di occorrenze in cui entrambi gli item sono presenti tra tutti i profili utenti. La fase di raccomandazione, poi, è sempre una semplice moltiplicazione tra il vettore utente e la matrice del modello.

2.4.3.3 Sarwar

Sempre della famiglia degli Item-Based, questo algoritmo utilizza la decomposizione SVD sulla matrice URM per ridurre lo spazio fisico occupato. Infatti si utilizzerà solamente una delle tre matrici calcolate tramite la decomposizione. La matrice di similarità tra elementi viene calcolata con i seguenti passaggi:

1. si riceve come input la matrice URM e un parametro ls che rappresenta la dimensione dello spazio latente in cui costruire la matrice modello;

2. si applica la decomposizione SVD alla matrice URM $(m \times n)$, utilizzando lo spazio latente ls: si ottengono quindi le matrici U $(m \times ls)$, S $(ls \times ls)$, V' $(ls \times n)$. Quest'ultima matrice V' rappresenta il modello dell'algoritmo e, come si può facilmente verificare, è di dimensioni minori rispetto a tutte quelle $n \times n$ utilizzate in precedenza;

3. si moltiplica $V \times V'$ in modo da ottenere la matrice di similarità (il modello vero e proprio): questa viene creata solamente per i calcoli, in quanto per la memorizzazione si utilizza sempre solo V'.

Infine per ottenere la raccomandazione si effettua sempre la solita moltiplicazione del vettore del profilo utente per la matrice di similarità, ottenendo un coefficiente per ogni elemento. Ordinando poi in ordine decrescente i coefficienti si avrà la lista Top-N da suggerire.

2.4.3.4 Asymmetric SVD

Ideato da Yehuda Koren, questo algoritmo[9] Item-Based è stato il vincitore della competizione Netflix Prize, e quindi si è imposto come lo stato dell'arte per quanto riguarda gli algoritmi collaborativi. È in grado di lavorare sia con dataset impliciti sia espliciti.

Per ogni elemento i non presente nel profilo dell'utente u viene calcolato un rating per mezzo della seguente formula:

$$r_{ui} = b_{ui} + q_i^T(|R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - b_{uj})x_j + (|N(u)|)^{-\frac{1}{2}} \sum_{j \in N(u)} y_j) \tag{2.3}$$

Analizziamo ora nel dettaglio la formula:

$b_{ui} = \mu + b_u + b_i$ dove μ è la media dei rating del dataset, mentre b_u e b_i sono le distanze (*bias*) rispettivamente dei ratings dell'utente u e della media dei ratings ricevuti dall'item i rispetto alla media totale μ;

$|R(u)|$ è la cardinalità dell'insieme dei ratings *espliciti* dell'utente u;

$|N(u)|$ è la cardinalità dell'insieme dei ratings *impliciti* dell'utente u;

q è il vettore degli elementi i non votati dall'utente;

x è il vettore degli elementi j che hanno ricevuto un rating esplicito;

y è il vettore degli elementi j che hanno ricevuto un rating implicito.

Questo algoritmo è stato poi leggermente modificato per adattarlo anche a scenari differenti da quello del Netflix Prize:

- alcuni parametri che intervenivano durante la creazione del modello sono stati cambiati in quanto si verificava overfitting sul bias degli utenti e degli elementi, portando quindi la raccomandazione ad essere sempre identica;
- durante la raccomandazione si verifica un ulteriore apprendimento sul profilo dell'utente: per questo è stata leggermente modificata questa fase per poter utilizzare anche dei profili creati ad hoc e non necessariamente facenti parte della matrice URM.

Capitolo 3

Metodologie di Test

Lo studio dei sistemi di raccomandazione dipende in grandissima parte dalla buona qualità dei risultati che si ottengono: il fine ultimo, infatti, è quello di fornire suggerimenti all'utente per invogliarlo ad acquistare contenuti multimediali. Se i suggerimenti risultano essere adeguati e quindi l'utente dà fiducia al sistema , si incrementeranno sia gli acquisti sia la soddisfazione generale. Se invece l'utente si accorge che le raccomandazioni fornite non risultano essere attendibili, si creerà un'insoddisfazione nei confronti del sistema, e di conseguenza diminuirà il numero degli acquisti.

Ma come si fa a capire se un algoritmo raccomanda elementi in modo adeguato o meno? In questa circostanza viene in aiuto la statistica, mostrando delle metodologie di test che generano un metro di giudizio per ogni singolo test.

3.1 Hold Out

La più semplice in assoluto forma di test si ottiene prendendo una piccola percentuale del dataset (solitamente intorno al 3%) e facendola diventare il TestSet. La restante parte forma invece il TrainSet. Com'è facilmente intuibile in fase di testing si usa il TrainSet per creare il modello che verrà usato in seguito per stimare i risultati del TestSet.

Questo test ha il vantaggio di essere il più semplice di tutti, ma si dimostra anche molto inefficiente in quanto all'interno della percentuale usata come TestSet

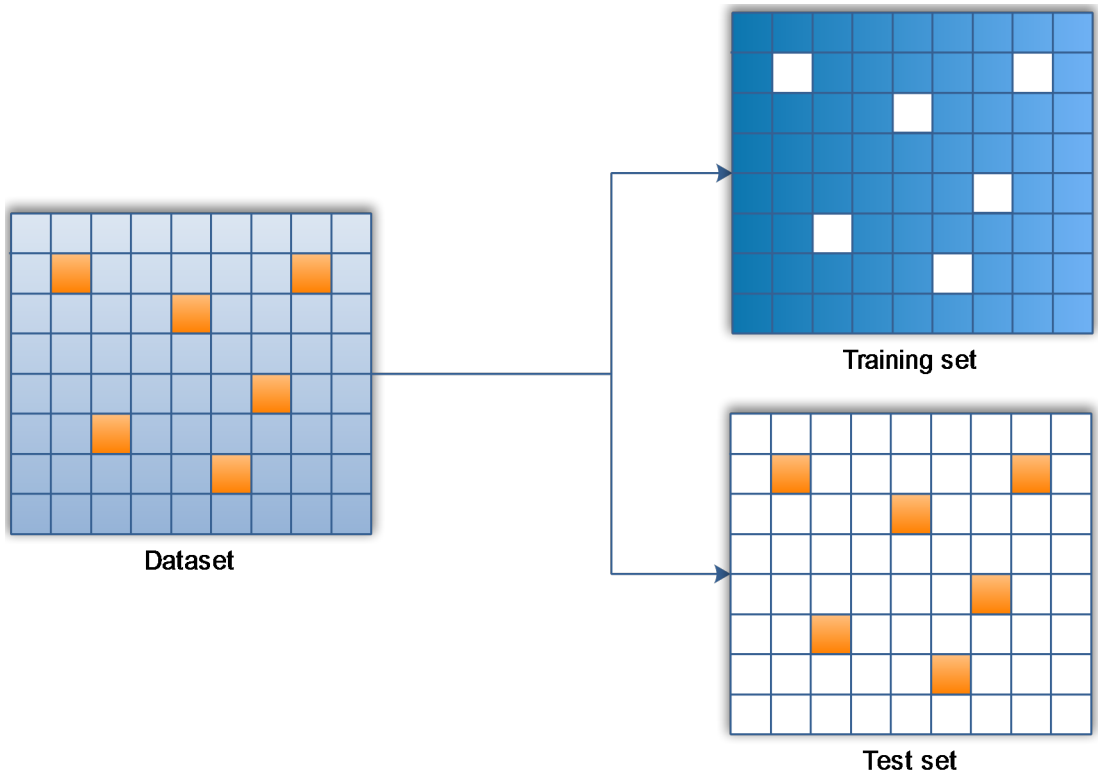

Figura 3.1: Metodologia HoldOut

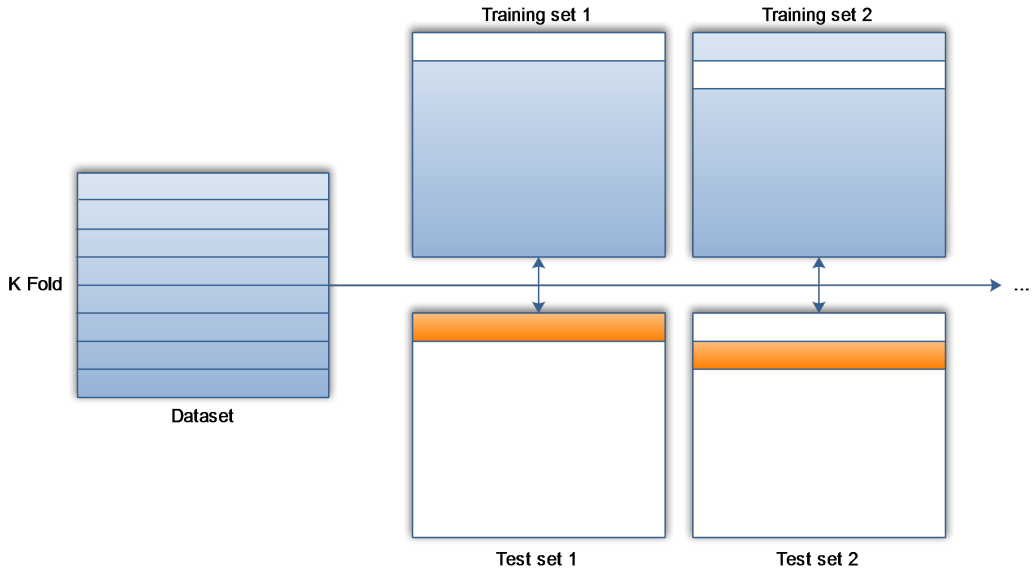

Figura 3.2: Metodologia K-Fold Cross Validation

si possono presentare eccezioni o casi particolari. Un rimedio a ciò può essere fornito dall'introduzione di filtri[1] durante la suddivisione del dataset, ma anche questo non sempre risolve i problemi. Ci si affida quindi ad altre metodologie per ottenere risultati più accurati, mentre questa viene utilizzata se si vuole una stima rapida (seppur imprecisa) del tutto, visto che gli altri metodi sono incredibilmente più complessi dal punto di vista computazionale.

3.2 K-Fold Cross Validation

Questa tipologia di test consiste nello stimare l'affidabilità di un'analisi in base a un campione indipendente. Il dataset viene suddiviso casualmente in K parti con la stessa cardinalità. Una alla volta queste parti diventano il campione di validazione, mentre le restanti $K-1$ sono il campione di training. Questo procedimento viene ripetuto K volte in modo che tutte le parti diventino, una alla volta, campione di validazione. Durante la validazione di una parte, le altre servono per il training,

[1] Si può ad esempio decidere di prendere in considerazione solamente i voti superiori una certa soglia nel caso di dataset espliciti, oppure utenti che han fornito almeno un certo numero di preferenze.

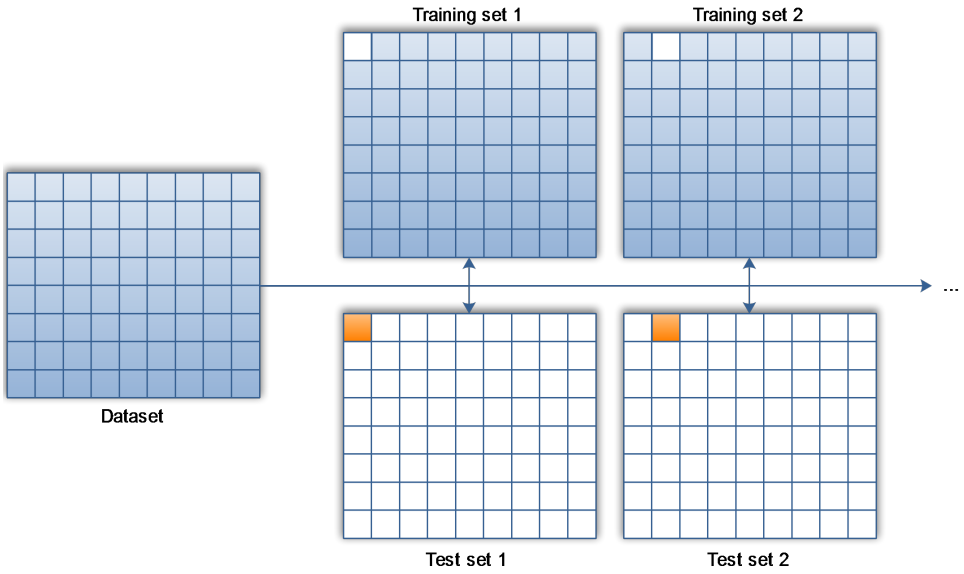

Figura 3.3: Metodologia Leave One Out

quindi per la creazione del modello. Questa validazione viene realizzata analizzando un singolo campione del TestSet alla volta, con un meccanismo chiamato Leave One Out (analizzato in seguito): in questo caso però il modello è già stato creato e si usa sempre quello.

I risultati ottenuti dopo tutti i campionamenti vengono riuniti per generare un unico risultato che fornisca la stima cercata. Il vantaggio di questo metodo è quindi che ogni parte del dataset faccia da validazione e da training, in modo da non generare overfitting. In letteratura a K viene solitamente attribuito il valore 10.

3.3 Leave One Out Cross Validation

Un caso particolare del metodo esaminato prima è il Leave One Out, dove in pratica si assegna a K il valore della cardinalità del dataset. Viene testato un solo valore alla volta, usando tutti gli altri per il training.

Questa validazione può essere considerata migliore di quella precedente in quanto si riesce a indicare con più precisione se esistono dei casi per cui l'al-

goritmo sbaglia, fornendo raccomandazioni errate. C'è anche da dire, però, che computazionalmente è molto più pesante in quanto il modello deve essere ricreato molte più volte rispetto al semplice K-Fold, quindi si cerca di usarlo solamente per problemi di piccole dimensioni.

3.4 Metriche

Dopo aver esaminato le varie tipologie di test si passa ora a capire esattamente come si può valutare un algoritmo e quali metri di giudizio si hanno per determinare se conviene o meno.

Si usano quindi delle metriche, valori che mostrano (su una scala relativa o assoluta) un giudizio su quanto è stato valutato dal test, in modo da riuscire a classificare in modo semplice e rapido i vari algoritmi. In seguito verranno mostrate le varie metriche, suddividendole in:

- Metriche di errore;
- Metriche di classificazione.

3.4.1 Metriche di errore

Servono normalmente per valutare la predizione di un rating[2]. Non sono molto utilizzate in quanto gli algoritmi correttivi influiscono solamente sulla posizione nella classifica ritornata dalla fase di raccomandazione, e non sulla valutazione stessa.

3.4.1.1 Mean Absolute Error (MAE)

Letteralmente è l'*Errore Medio Assoluto* e rappresenta la distanza media tra il valore predetto e quello reale. Questa distanza è, ovviamente, calcolata in valore assoluto:

[2]Il campo normale di utilizzo sono i rating espliciti in quanto viene valutata la distanza tra la predizione e il valore vero. Possono essere anche utilizzate in presenza di valutazioni implicite, ma in questo modo non riescono ad avere tutta la potenza di espressione che hanno nel caso di rating espliciti.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{r}_{ij} - r_{ij}| \tag{3.1}$$

3.4.1.2 Mean Squared Error (MSE)

Viene tradotto letteralmente come *Errore Quadratico Medio* ed è la differenza quadratica media tra i valori osservati e i valori attesi. Si calcola in modo simile al MAE:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{r}_{ij} - r_{ij})^2 \tag{3.2}$$

3.4.1.3 Root Mean Squared Error (RMSE)

È la radice quadrata dell'MSE e rappresenta la varianza interna data dal rapporto fra la devianza interna e la numerosità totale:

$$RMSE = \sqrt{MSE} \tag{3.3}$$

3.4.2 Metriche di classificazione

3.4.2.1 Recall

Nella raccomandazione, un elemento può essere fondamentalmente di due tipi: rilevante per l'utente, che quindi si aspetta di trovarlo in cima alla lista, con un punteggio alto, oppure non rilevante per l'utente, che quindi si stupirebbe se fosse nella parte alta della lista. Proprio per questi motivi ogni elemento raccomandato può essere classificato in uno dei seguenti modi:

True Positive (TP) item consigliati dal sistema e rilevanti per l'utente, ovvero quelli che giustamente ricoprono una posizione alta nella classifica;

False Positive (FP) item consigliati dal sistema, ma che non ricoprono un ruolo rilevante per l'utente; rappresentano gli errori "positivi" del sistema;

True Negative (TN) elementi che non sono consigliati dal sistema e che non sono rilevanti per l'utente, quindi quelli che giustamente hanno un rating basso;

False Negative (FN) elementi che non vengono raccomandati dal sistema, ma che l'utente vorrebbe vedersi consigliati.

La recall, una delle metriche più accurate per valutare le prestazioni di un algoritmo di raccomandazione, si basa su un'idea molto semplice: quanti elementi rilevanti sono stati consigliati tra tutti quelli interessanti per l'utente? In formule:

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

Proprio per questa sua caratteristica di considerare una percentuale del totale degli elementi viene definita una metrica di completezza che dipende molto dalla lunghezza della lista da raccomandare, quindi dal parametro N della lista Top-N: per "positivi" e "negativi" infatti si intendono quegli elementi che compaiono o meno nelle prime N posizioni della lista.

Com'è ovvio pensare, a valori alti di recall corrispondono algoritmi più accurati[3].

3.4.2.2 Precision

Anche questa è una delle metriche più accurate nella valutazione di un algoritmo di raccomandazione, ma, a differenza della recall, non misura la completezza, bensì l'esattezza e l'affidabilità dell'algoritmo, misurando infatti la percentuale di elementi consigliati che l'utente trova rilevanti, rispetto al numero di quelli raccomandati in totale[4]:

$$Precision = \frac{TP}{N} \tag{3.5}$$

Come si è visto per la recall, anche questa metrica dipende fortemente dal parametro N.

[3]Il valore massimo della recall sarà 1 quando non esistono "false negative", mentre sarà 0 nel caso pessimo, ovvero quando non esistono "true positive".

[4]Nella formula seguente N è lo stesso di *Top-N*, rappresenta quindi il numero di elementi che rientrano nella raccomandazione visibile all'utente.

3.4.2.3 Expect Percentile Ranking (EPR)

La *Votazione Attesa Percentuale* è una misura della qualità che considera altro oltre alla lista Top-N: preso infatti un campione di raccomandazioni indica, percentualmente, la posizione in classifica in cui vengono consigliati gli elementi. Si calcola nel seguente modo:

$$EPR = \frac{\sum_{u,i} r_u^T i \times rank_u i}{\sum_{u,i} r_u^T i} \tag{3.6}$$

È quindi la somma delle posizioni in classifica di tutti gli item del campione preso in esame, diviso per la sua numerosità.

Più è basso il valore dell'EPR, più il sistema ha predetto gli elementi visti dall'utente nelle prime posizioni. Il 50% corrisponde a una raccomandazione effettuata prendendo i valori in modo casuale, e più si cresce più lo scenario peggiora.

3.4.2.4 Average Reciprocal Hit Rank (ARHR)

È una misura della qualità che esamina la posizione in cui un elemento viene raccomandato all'interno della lista stessa. Si ripensi alla precision e la si ridefinisca come il rapporto tra una raccomandazione ritenuta importante per l'utente (ovvero un TP), chiamata *hit* e il numero n di raccomandazioni:

$$HitRate = \frac{\#Hits}{n} \tag{3.7}$$

Quando l'Hit Rate è uguale a 1, l'algoritmo è riuscito a raccomandare tutti gli elementi rilevanti. Non viene però presa in considerazione la posizione in classifica in cui viene consigliato l'elemento: non c'è differenza se quello meno rilevante è stato suggerito per primo mentre quello più rilevante compare all'ultimo posto.

Superiamo questa limitazione utilizzando la metrica ARHR che dà un punteggio a ogni hit, sulla base della posizione nella lista. Definito h il numero di hit che avvengono nelle posizioni $p_1, p_2, \ldots, p_N$:

$$ARHR = \frac{1}{n} \sum_{i=1}^{h} \frac{1}{p_i} \tag{3.8}$$

Agendo in questo modo gli hit nelle posizioni alte della classifica valgono di più di quelli nelle posizioni basse. Se gli elementi compaiono tutti in prima posizione, allora il valore dell'ARHR coincide con quello dell'Hit Rate. Quando invece gli hit avvengono nelle posizioni basse della classifica l'ARHR diminuisce fino ad arrivare al valore limite di $\frac{HR}{n}$.

Capitolo 4

Analisi dell'applicazione e descrizione del problema

La trattazione va ora a prendere in esame la piattaforma di raccomandazione com'era prima di questo lavoro di tesi, esaminandone attentamente i difetti che poi hanno portato alla volontà di ripartire da zero per costruire una piattaforma più dinamica e agile.

4.1 La partenza

Alla creazione della piattaforma certo non si pensava che risultasse qualcosa di così complesso: si sentiva solamente la necessità di implementare qualche algoritmo di raccomandazione, più per esercizio che per altro: non si era presa in esame una possibile integrazione dei vari algoritmi, per questo furono creati completamente diversi gli uni con gli altri[1] e ognuno con i suoi parametri.

Procedendo col lavoro si evidenziarono i limiti di queste strutture, e soprattutto i limiti di MatLab che seppur semplificando molte operazioni, non riesce a raggiungere le velocità tipiche del C e di altri linguaggi di programmazione più a basso livello. Fu così che le funzioni più impegnative vennero implementate in C,

[1] Come esempio basti citare il nome del parametro con cui veniva passato il modello già creato: in ogni algoritmo che implementa questa possibilità, nemmeno in tutti, si usa un nome diverso che riflette le caratteristiche del determinato modello.

riuscendo a garantire per lo meno standard prestazionali di gran lunga superiori a quelli precedenti.

Continuava a rimanere, però, sempre un qualcosa di slegato e fine a se stesso, lontano dal suscitare il ben che minimo interesse.

4.2 NetFlix e i test

Finalmente arrivò dall'esterno una spinta per iniziare a dare un senso a tutti gli algoritmi inseriti e funzionanti: un provider di IPTv statunitense indisse una competizione per trovare l'algoritmo di raccomandazione migliore al mondo. Da tutte le migliori università e centri di ricerca iniziarono ad arrivare tonnellate di algoritmi che vennero processati da NetFlix[1] stesso per verificarne la bontà.

Alla fine vinse la competizione Yehuda Koren col suo algoritmo "Asymmetric SVD" (cfr. 2.4.3.4), ma il sistema in oggetto di questo lavoro di tesi iniziò a implementare tutte quelle metodologie di test che hanno permesso a NetFlix di valutare i vari lavori.

Nacque così l'idea di organizzare i vari algoritmi, che comunque erano ancora disuniti e privi di un filo conduttore che ne guidasse l'implementazione, per lo meno in directory e nel numero di input. Così facendo ogni metodo di test poteva utilizzare qualsiasi algoritmo per creare il modello e quindi la raccomandazione da valutare.

Contemporaneamente all'implementazione dei test si cercò anche si sistemare gli algoritmi già visti per poterli migliorare e adattare alla nuova piattaforma: vennero infatti creati degli studi per sistemare alcuni algoritmi particolarmente complessi, in modo da semplificarli e renderli più adatti alla piattaforma che ormai era stata prodotta, seppur ancora adatta solamente a chi ci lavora quotidianamente.

4.3 Le novità

Iniziarono finalmente ad arrivare anche le prime novità apportate alla piattaforma: prima fra tutte l'antireshuffling (cfr. 2.3) che introduceva una logica nuova nella creazione delle liste di raccomandazione.

Quest'aggiunta, però, veniva implementata scontrandosi contro la frammentarietà dell'opera fino ad allora prodotta, quindi era stata creata una struttura incredibilmente fragile di funzioni che necessitavano di parametri incredibilmente specifici per funzionare: utilizzare queste funzionalità risultava praticamente impossibile se prima non si veniva sottoposti a una sessione di training.

Man mano furono anche aggiunte una serie di metodologie di test differenti, prese in letteratura e implementate in modo sommario, al solo fine di ottenere determinati risultati ma senza la necessità di legare le varie strutture di dati o i risultati stessi.

4.4 Lo stop

Era questo il momento giusto per fermarsi e riflettere: per quanto tempo poteva ancora essere interessante una struttura del genere, completamente frammentata e in cui ogni riga di codice era fine a se stessa? Quanto poteva servire una piattaforma simile, dove comparivano tanti risultati che se considerati contemporaneamente risultavano inutili in quanto slegati?

Si decise quindi di fermare, almeno in parte, la ricerca e focalizzare l'attenzione sul rifacimento dell'intero sistema, in maniera più intuitiva e soprattutto avendo sott'occhio l'intero lavoro: non si pensava più di dover solamente costruire alcuni algoritmi come esercizio, ma c'era la necessità di permettere la scalabilità della struttura, la facile integrazione di nuovi algoritmi che potessero essere presi facilmente dal sistema e utilizzati attraverso semplici API.

Anche le metodologie di test dovevano essere raccolte e adattate al sistema, in modo che ne risultassero facili sia l'ampliamento sia la modifica.

Ormai si era arrivati al punto di impiegare più tempo per comprendere la piattaforma che per implementare nuove funzionalità o studiarne i risultati.

4.5 Altri esempi esterni a questo progetto

Per cercare di capire meglio come organizzare il lavoro e soprattutto su quali aspetti agire, si è guardato anche ad altri progetti realizzati in altre parti del mondo. Tra

tutti quelli trovati, si è deciso di citarne solamente alcuni in quanto costituiscono un campione esemplare.

4.5.1 Open Source Recommender System Software Workshop

Viene inserito questo progetto[13] per primo in quanto costituisce il miglior tentativo (anche se poi si è rivelato il peggiore visto che è stato cancellato quasi subito) di realizzare una piattaforma opensource dedicata ai sistemi di raccomandazione.

L'idea di fondo degli sviluppatori che ebbero l'idea era creare una comunità di studenti e professionisti intorno a questa piattaforma, in modo da non dipendere sempre da quanto sviluppato dalle aziende: la loro idea era che le soluzioni a pagamento fossero troppo poco trasparenti, e quindi si rivelassero alla lunga non interessanti da un punto di vista puramente didattico.

Proprio per questo si volle creare un progetto aperto a tutti con la speranza che, grazie al web e alla disponibilità degli utenti, si riuscisse a dare alla luce una valida alternativa alle piattaforme proprietarie, rendendo i sistemi di raccomandazione un qualcosa di più facilmente capibile e utilizzabile. Inoltre una soluzione del genere poteva essere accolta positivamente anche dagli sviluppatori e dalle aziende del settore, che l'avrebbero presa come base da ampliare: fa sempre comodo avere delle fondamenta comode e indipendenti per svilupparci sopra le proprie idee.

Sfortunatamente tutti questi buoni propositi non ebbero seguito in quanto la prima conferenza sull'argomento venne cancellata in quanto non vi erano iscritti, e dopo poco tempo, anche l'intero progetto morì dal momento che la comunità sognata non si era creata. A un anno dalla data della conferenza, l'unica traccia che rimane di questo lavoro è il **Duine Framework**[8], lavoro dei tre ideatori del workshop, che però risulta essere ancora a livello di bozza: ideato per essere il punto di riferimento per i sistemi di raccomandazione, appare con una struttura estremamente pesante ed elaborata. Al suo interno implementa solamente un paio di algoritmi e una sola metodologia di test. Inoltre anche di questo progetto non si hanno notizie da più di un anno e mezzo.

4.5.2 Apache Mahout

Direttamente dall'*Apache software foundation* si è cercato di realizzare una libreria[6] in grado di gestire l'apprendimento automatico. In questo modo i sistemi di raccomandazione sono una delle sue possibili applicazioni: è sufficiente includere gli algoritmi visti all'inizio di questa trattazione, e la libreria è in grado di raccomandare item di svariati tipi.

Un'altra caratteristica molto importante di questo progetto, è la facilità con cui gestisce enormi carichi di dati: la struttura di base è **Hadoop**[5], sempre di Apache, che permette un *mapping* incredibilmente efficace dei dati, in modo da non risentirne l'aumento.

In modo nativo vengono gestiti sia gli algoritmi Content-Based, sia quelli Collaborativi.

Questo progetto nacque per vari motivi:

- Si voleva anzitutto creare una comunità intorno al *machine learning* in quanto tutti gli altri progetti esistenti non riuscivano a raccogliere molti sviluppatori indipendenti disposti a condividere le proprie idee e i propri risultati;
- Non vi era una buona documentazione in rete che desse un'idea del lavoro che è stato fatto fin qui, ma ognuno era convinto di dover reinventare la ruota: si è voluto creare un punto di partenza per tutti;
- Nessun prodotto garantiva un'elevata scalabilità in quanto la maggior parte, essendo sviluppati a livello accademico, venivano usati a scopi puramente didattici.

Col passare del tempo vennero assorbiti progetti minori. Uno tra tutti è Taste[10], nato nel 2006 come piattaforma di test per la competizione Netflix[2], è un insieme di classi Java che permette creare modelli e liste di suggerimenti in base a matrici URM fornite. I limiti di quest'applicazione furono evidenti fin dall'inizio: le strutture matematiche non erano gestite alla perfezione e soprattutto aveva problemi con le grandi quantità di dati tipiche di Netflix: proprio per questo motivo si decise la fusione con Mahout e il conseguente abbandono del progetto.

Capitolo 5

Reingegnerizzazione dell'applicazione

Si va ora a esaminare in dettaglio la reingegnerizzazione dell'intera piattaforma, studiando più da vicino le idee da cui si è partiti ma soprattutto l'organizzazione e la strutturazione del lavoro, in modo da rendere il sistema il più facilmente fruibile possibile, senza trascurarne l'espandibilità.

5.1 Le premesse

Nella reingegnerizzazione del sistema si è voluto partire da zero e costruire quindi gli oggetti necessari per le funzionalità di raccomandazione vera e propria e per i vari test.

La soluzione più semplice e immediata è stata ideare due oggetti separati in grado di interagire tra di loro:

Recommendation gestisce la fase di suggerimento attraverso la creazione del modello, la generazione della lista e infine il reshuffling e l'esportazione dei risultati. Deve poter utilizzare qualsiasi algoritmo in grado di generare una lista di raccomandazioni.

Test si occupa della fase di testing, richiamando funzioni che implementano i diversi metodi, in modo da eseguire un test completo in pochi passi, fornendo

anche tutte le parametrizzazioni necessarie. Non deve occuparsi della fase di generazione della lista di suggerimenti, ma solamente della coordinazione delle diverse liste ottenute.

Il procedimento viene complicato dal fatto che gli algoritmi di generazione del modello e le metodologie di test sono molto diverse tra di loro, quindi devono essere implementati separatamente. Ma proprio questa diversità può anche risultare un punto di forza, in quanto si possono già creare le fondamenta per un'espansione futura dell'applicazione.

Ogni oggetto deve essere inteso come un *workflow*, ovvero come un flusso che si occupa di generare completamente la raccomandazione o il test. L'utente deve poter interagire solamente con un'istanza dell'oggetto, senza perder tempo nella gestione di istanze multiple che, oltre a consumare un'enorme quantità di memoria, rendono l'esperienza dell'utente poco pratica, visto che è proprio per sopperire alla mancanza di praticità che ci si è fermati per riorganizzare il tutto.

5.2 La struttura

Nella piattaforma in esame sono fondamentalmente due le parti parametriche: l'algoritmo di raccomandazione, che si occupa sia della creazione del modello sia della generazione della lista di suggerimenti, e la metodologia di test. Per questo motivo si sono intesi i due oggetti (Recommendation e Test) come dei *wrapper* che essenzialmente vanno a recuperare altre funzioni in specifiche zone del filesystem a seconda delle scelte fatte: ad esempio se si vuole effettuare un test "Leave One Out" con un algoritmo "Asymmetric SVD" è sufficiente recuperare le funzioni che gestiscono il "Leave One Out" e quelle che implementano l'algoritmo voluto e quindi fonderle per generare i risultati.

La fase di recupero dell'implementazione cercata dev'essere molto semplice e intuitiva, proprio per questo si è deciso di adeguare la struttura delle directory che contengono i vari sorgenti (cfr. Figura 5.1):

Algorithms raccoglie, uno per cartella, i vari algoritmi di raccomandazione. All'interno di queste cartelle sono presenti due file Matlab:

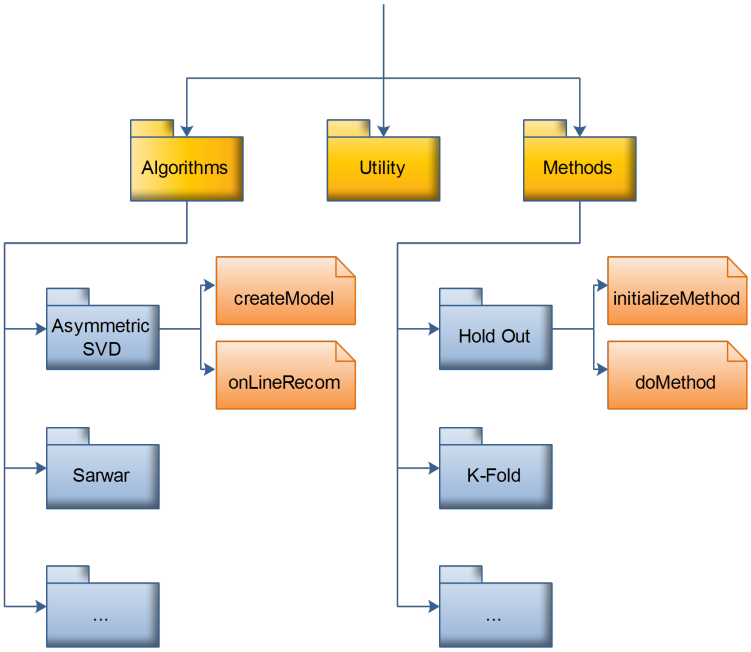

Figura 5.1: Struttura del filesystem

createModel si occupa della creazione del modello, prendendo come input la matrice URM (di dimensione $m \times n$) e una struttura di parametri aggiuntivi che servono per specificare alcune opzioni dei vari algoritmi. Tra questi parametri deve esserci *Path*[1] che specifica il percorso della root della struttura dati e può anche esserci il parametro *Model*: se è settato la funzione ritorna subito il contenuto di questo parametro, senza effettuare ulteriori controlli. Quest'opzione è stata aggiunta in quanto molto spesso la creazione del modello è computazionalmente talmente onerosa da rendere impossibili certi test, quindi questi vengono eseguiti utilizzando più volte il medesimo modello e abbreviando quindi i tempi di calcolo. L'output di questa funzione sarà il modello della raccomandazione, che verrà poi gestito dall'oggetto Recommendation stesso.

[1]Questo parametro viene inserito in automatico dall'oggetto Recommendation.

```
Model = createModel(URM, Parameters)
```

Codice 5.1: Funzione createModel

onLineRecom si occupa di generare la lista di suggerimenti. Prende come input il modello creato in precedenza, il profilo utente (un vettore riga di lunghezza n) e infine, come già visto per la createModel, una struttura di parametri aggiuntivi per modificare alcuni comportamenti dell'algoritmo. Come nel caso della createModel, anche qui deve esserci il parametro *Path*. L'output di questa funzione è un vettore colonna di lunghezza m con i rating dei vari item.

```
List = onLineRecom(UserProfile, Model, Parameters)
```

Codice 5.2: Funzione onLineRecom

Ogni algoritmo ha le sue particolarità e per questo il sistema non interviene mai sul modello della raccomandazione, ma si limita a passare il dato dalla funzione di creazione a quella di generazione della lista. Solamente i parametri di ingresso e la lista finale sono "standardizzati", tutto il resto è specifico da algoritmo ad algoritmo.

Methods contiene le diverse metodologie di test che risultano divise in due funzioni:

initializeMethod è la funzione che viene lanciata quando si vuol testare un algoritmo con un certo metodo. Contiene tutte le dovute inizializzazioni come preparare la matrice di Train e quella di Test. Al suo interno verrà richiamato il doMethod. Avrà come parametri di input la matice URM, la matrice di Probe su cui fare il Train del test, l'algoritmo di raccomandazione che si vuole studiare e infine i vari parametri che si vogliono utilizzare, quelli del metodo in sè, quelli della createModel e infine quelli dell'onLineRecom. L'output sarà un oggetto con al suo interno i risultati del test secondo una determinata metrica (principalmente RMSE).

```
Result = initializeMethod(URM, URMProbe,
    Algorithm, MethodParameters, ModelParameters,
    OnLineParameters)
```

Codice 5.3: Funzione initializeMethod

doMethod contiene tutte le operazioni tipiche della metodologia di test. Per funzionare a dovere ha bisogno che gli siano passati tutta una serie di parametri che la inizializeMethod ha il compito di preparare. In alcuni metodi è possibile richiamare una funzione doMethod di un altro metodo di test. Gli input di questa funzione sono l'algoritmo da testare, la matrice URM, la matrice di Train, il set dei test positivi e di quelli negativi e infine i parametri della createModel e dell'onLineRecom. L'output è un array con i test positivi e quelli negativi.

```
[PositiveTests, NegativeTests] =
    doMethod(Algorithm, URM, URMTrain,
    PositiveTestSet, NegativeTestSet,
    ModelParameters, OnLineParameters)
```

Codice 5.4: Funzione doMethod

Questa suddivisione è stata introdotta in quanto è possibile che un metodo utilizzi parte di un altro metodo per svolgere alcune funzioni.

Utility contiene tutte quelle funzioni che servono a far funzionare i vari algoritmi e le funzioni di test. Al suo interno si possono trovare funzioni per la normalizzazione delle matrici, per il reshuffling e infine la libreria C che si occupa di velocizzare le operazioni più onerose. Si è scelto di inserire tutti i file in questa cartella per fare ordine, cosa che prima dell'intervento in oggetto a questo lavoro di tesi non era minimamente pensabile.

5.3 Gli oggetti

Si passa ora all'esame dettagliato dei due oggetti principali del lavoro di reingegnerizzazione, *Recommendation* e *Test* che rappresentano anche i soli punti di

contatto tra il sistema e l'utente: infatti tutte le interazioni avverranno attraverso questi wrapper mediante opportune API. Verranno anche presentati i metodi principali di questi oggetti, in modo da fornire un esempio di utilizzo.

5.3.1 Recommendation

Quest'oggetto è stato ideato partendo da una semplice idea: ogni istanza deve rappresentare un'esperienza di raccomandazione, quindi l'utente deve creare solamente un oggetto e continuare ad usarlo memorizzando al suo interno dati finché non ha finito la raccomandazione completa. In questo modo l'utente non si trova ad avere memorizzate troppe istanze dell'oggetto stesso, col risultato di risparmiare memoria e soprattutto rendere ordinato il workspace.

L'utente deve inoltre poter scegliere tranquillamente quale algoritmo utilizzare, e quindi non doversene più interessare: è l'oggetto che andrà a costruire il modello e creerà la lista di raccomandazioni secondo quel determinato algoritmo. È anche possibile modificare l'algoritmo scelto, ma poi sarà necessario ricreare il modello in quanto potrebbero nascere delle incompatibilità, basti pensare a due algoritmi che utilizzano modelli differenti.

Altra particolarità di quest'oggetto è non salvare la matrice URM: essa viene tenuta in memoria nell'ambiente base di Matlab, in modo da non pesare all'interno dell'oggetto. In questo modo l'accesso risulta essere più rapido. Si è voluto cercare di portare su Matlab il concetto dei puntatori del linguaggio C, ma non esistendo un costrutto del genere si è iniziato a lavorare differenziando i workspace. A tal scopo all'interno dell'oggetto viene salvato solamente il nome della matrice, che risiede fisicamente nell'ambiente di base.

Infine si è cercato di memorizzare tutti i risultati delle raccomandazioni eseguite: in questo modo si può accedere rapidamente nel caso di Reshuffling e di test in cui si vogliono avere a disposizione le ultime liste create. Inoltre quest'operazione non è computazionalmente troppo onerosa in quanto si tratta semplicemente di vettori colonna, seppur di grandi dimensioni.

Si prosegue quindi con l'analisi dei diversi metodi e attributi di quest'oggetto.

5.3.1.1 Attributi Privati

Sono stati protetti in scrittura poiché non risulta essere d'interesse modificarne il valore. Sarà invece importante potervi accedere in lettura (soprattutto a uno dei tre).

Model è il modello utilizzato. Viene creato da un apposito metodo. Teoricamente potrebbe essere protetto anche in lettura in quanto non è di enorme interesse conoscerne la forma, ma visto che per alcune fasi di debug può risultare interessante leggerne il contenuto, si preferisce lasciarlo pubblico in lettura.

Path è una stringa contenente il percorso assoluto della root della nostra struttura di cartelle. Serve alle varie funzioni per includere le diverse cartelle da cui prendere file utili (come la cartella delle Utility). Viene impostato dal costruttore dell'oggetto e in seguito non viene più modificato. Viene aggiunto a tutti i parametri e quindi passato in tutte le funzioni di creazione del modello e creazione della lista.

Result è un array contenente un oggetto per ogni raccomandazione. Questo oggetto è organizzato nel seguente modo:

Result.List è la lista vera e propria, così come esce dalla funzione di raccomandazione. Contiene tutti i voti dati ai vari elementi.

Result.Ranking è un array contenente le chiavi degli elementi di Result.List ordinati in modo decrescente, quindi il primo valore è la chiave dell'elemento col voto maggiore.

Result.UserProfile è il profilo utente utilizzato per generare la raccomandazione. Nel caso in cui il risultato provenga dalla funzione di antireshuffling e non da una raccomandazione vera e propria, è un array contenente i due profili utenti associati alle liste su cui è stato applicato l'antireshuffling.

5.3.1.2 Attributi Pubblici

I seguenti attributi, possono essere soggetti a modifiche da parte dell'utente. Proprio per questo sono pubblici, benché la loro modifica sia soggetta a vincoli, anche molto stringenti per quanto riguarda uno di essi.

Algorithm è l'algoritmo che si utilizza per generare la raccomandazione. È una stringa e deve corrispondere alla cartella in cui sono contenuti i file con le funzioni utilizzate. Non è necessario memorizzare tutti i nomi, poiché in caso d'errore il sistema è in grado di stampare schermo il nome di tutti gli algoritmi disponibili. Questa proprietà può essere modificata solamente prima di creare il modello della funzione: in caso contrario verrà stampato un messaggio d'errore e l'algoritmo rimarrà il medesimo.

URM è una stringa che corrisponde al nome della variabile, nell'ambiente *base*, che costituisce la matrice URM. Come per l'algoritmo, anche questa stringa non può essere modificata dopo la creazione del modello. Si occupa in automatico di caricare le variabili indicate quando la stringa passata corrisponde a un file nel workspace.

5.3.1.3 Metodi

In questa fase di esamineranno i metodi che permettono all'utente di interagire con l'oggetto per creare il modello, la raccomandazione e per altre funzionalità. Tutti i metodi che verranno presentati sono pubblici in quanto non c'è alcuna necessità di creare metodi privati poiché, come ripetuto più volte in questa trattazione, l'oggetto non è nient'altro che un wrapper. Per completezza è opportuno informare il lettore che in seguito i parametri opzionali per i metodi saranno posti tra tramite parentesi quadre.

Recommendation è il costruttore dell'oggetto. Prende come parametri il nome della matrice URM da utilizzare e il nome dell'algoritmo. L'assegnazione del nome della variabile contenente la matrice e dell'algoritmo sono soggetti a limitazioni nel caso in cui non esista o la variabile o il file contenente la matrice URM, oppure non esista la cartella indicata come algoritmo da utilizzare. Nel caso queste assegnazioni non vadano a buon fine, l'oggetto viene

comunque creato, ma con i campi summenzionati vuoti. Sarà poi compito dell'utente andare a riempirli prima di creare il modello.

```
R = Recommendation(URM, Algorithm)
```

Codice 5.5: Funzione costruttrice dell'oggetto Recommendation

createModel serve per creare il modello della funzione. Come è facilmente intuibile, controlla innanzitutto che siano settati l'algoritmo da utilizzare e la matrice URM. Come input prende una struttura di parametri che poi saranno passati alla funzione di creazione vera e propria. Dal momento che molte funzioni di creazione del modello necessitano di altre funzioni "ausiliarie", alla struttura di parametri viene aggiunto *Path*. E, una volta creato, il modello viene memorizzato nell'attributo privato dell'oggetto.

```
createModel([Parameters])
```

Codice 5.6: Metodo per la costruzione del modello

onLineRecom si occupa della creazione e della memorizzazione della lista di raccomandazioni. Ha due parametri in ingresso: il vettore col profilo utente e la solita struttura di parametri da usare nella funzione di generazione dei suggerimenti. Come visto per la createModel, anche qui viene aggiunto il parametro indicante il percorso sul filesystem. L'output del metodo è la lista contenente le votazioni di ogni singolo elemento. Nell'oggetto viene inserita una voce all'attributo *Result*, contenente tutte le informazioni viste al punto 5.3.1.1.

```
List = onLineRecom(UserProfile[, Parameters])
```

Codice 5.7: Metodo per la creazione della lista di raccomandazioni

antiReshuffling si prende in carico l'esecuzione dell'antireshuffling su due risultati ottenuti con la onLineRecom. I parametri di ingresso sono tre: il primo indice dell'array Result contenuto nell'oggetto (alternativamente può essere specificato un profilo utente su cui eseguire la raccomandazione), il secondo

indice (anche qui si può specificare un profilo utente) e infine una struttura di parametri. Si può scegliere di non specificare tutti gli input operando quindi nei seguenti modi:

- se non vengono passati i parametri si esegue la funzione tra gli ultimi due risultati;
- se si specifica un solo indice (o vettore utente) viene eseguito l'antireshuffling tra questo e l'ultimo;
- se si specifica solo una struttura di parametri si considerano gli ultimi due risultati e si passano i parametri specificati;
- se vengono passati due indici (o vettori utente) si esegue l'antireshuffling tra quei due valori;
- se vengono passati un indice e una struttura di parametri si produce come risultato la combinazione del risultato indicato e dell'ultimo, passando alla funzione i parametri.

L'output del metodo è la nuova lista di raccomandazioni, inoltre viene memorizzata nell'attributo Result la lista più altre informazioni, come visto nel già citato punto 5.3.1.1.

```
List = antiReshuffling([Index1[, Index2[,
    Parameters]]])
```

Codice 5.8: Metodo per la funzione di antireshuffling

export si occupa, dell'esportazione dei risultati su un file di testo. Viene utilizzata la convenzione CSV[2] utilizzando un # per separare i diversi valori e una | per i campi. La funzione prende come input il file su cui memorizzare e gli indici (che possono essere passati sia come valori, sia come array) dei risultati da esportare. Se non viene specificato l'indice del risultato da esportare, la funzione agirà sull'ultimo creato.

[2]Coma Separated Values, ovvero valori separati da una virgola o più in generale da un limitatore.

```
export(File[, Index])
```

Codice 5.9: Metodo per l'esportazione dei risultati

5.3.2 Test

Quest'oggetto si occupa di testare un algoritmo secondo diverse metodologie. Il test di per sé viene pensato come qualcosa di automatico, in cui l'utente può cambiare i parametri in modo da effettuare una procedura adatta alle proprie esigenze.

Come già visto per le raccomandazioni, anche il test è un oggetto di cui basta anche solo un'istanza per poter operare: i risultati vengono salvati, secondo diverse metriche, all'interno dell'oggetto stesso, così che si possano eseguire diverse prove con algoritmi differenti o addirittura con metodologie differenti.

Ovviamente il suo funzionamento è legato a quello dell'oggetto Recommendation poiché tutte le raccomandazioni sono affidate a quest'ultimo. Attraverso un passaggio di parametri adeguato, l'utente può intervenire in qualsiasi punto della raccomandazione, in modo da avere il pieno controllo e gestire, ad esempio, spazi latenti differenti per più prove con lo stesso metodo.

5.3.2.1 Attributi Privati

Analogamente all'oggetto Recommendation, anche qui gli attributi privati sono riservati a informazioni che l'utente non deve modificare, ma che ugualmente possono essere letti.

Path è il percorso assoluto dell'M-file contenente l'oggetto in questione. Serve, come visto in precedenza, per includere le funzioni ausiliarie e i metodi di testing. Viene settato in automatico in fase di inizializzazione dell'oggetto.

Result è il contenitore dove vengono memorizzati tutti i risultati dei vari test fatti tramite l'oggetto istanziato. È un array di struct dove ogni voce rappresenta una metrica (quella più usata è l'RMSE, ma ne esistono altre).

5.3.2.2 Attributi Pubblici

Queste proprietà dell'oggetto, diversamente dalle precedenti, possono essere modificate. Gli attributi, questi devono sottostare a vincoli per la buona riuscita dell'algoritmo di test, come verrà più esplicitamente esaminato nelle singole spiegazioni.

Algorithm è l'algoritmo utilizzato per generare le raccomandazioni e i modelli. Deve essere uguale al nome della cartella in cui sono memorizzate le sue funzioni. Nel caso in cui non ci si ricordi del nome esatto, il sistema stamperà a video l'elenco degli algoritmi disponibili.

Method rappresenta la metodologia di test scelta. Similmente all'algoritmo, anche questa stringa deve coincidere con una determinata cartella. Anche per questo attributo non è necessario ricordare tutti i nomi perché l'oggetto stamperà a video tutte le metodologie disponibili.

URM è il nome della variabile da usare per la matrice URM. Se il nome specificato è quello di un file *.mat* o *.mm*, il sistema andrà in automatico a leggerlo dal filesystem e lo importerà nel workspace.

URMProbe è la matrice usata per il testing vero e proprio. Viene memorizzata solamente la stringa corrispondente e quindi caricata ogni volta dalla memoria.

5.3.2.3 Metodi

A differenza dell'oggetto Recommendation, Test non ha molti metodi al suo interno: è facilmente intuibile che per generare una raccomandazione necessitano diversi passi, mentre è sufficiente far girare un test per avere dei risultati. Avremo quindi solamente i metodi indispensabili. Questa scelta è anche suggerita dal fatto che s'è voluto semplificare al massimo l'oggetto, lasciando tutta la complessità alle funzioni che gestiscono le metodologie di testing e soprattutto alla scelta dei parametri. Sarà poi l'utente a utilizzare più volte lo stesso oggetto per avere test particolari.

Test è il costruttore dell'oggetto. Prende come parametri di input le due matrici da utilizzare, URM e Probe, l'algoritmo per la raccomandazione e il metodo di test. Nel caso in cui una di queste assegnazioni non sia corretta (ad esempio si sceglie un algoritmo inesistente) l'oggetto viene creato ugualmente, ma manca dell'attributo settato. Successivamente quando si cercherà di utilizzare un altro metodo verrà restituito un messaggio d'errore.

```
T = Test(URM, URMProbe, Algorithm, Method)
```

Codice 5.10: Funzione costruttrice dell'oggetto Test

fire si occupa di far partire il metodo di test, quindi di lanciare la funzione initializeMethod contenuta nella cartella *Method* scelta. Prende in ingresso i tre diversi tipi di parametri: quelli del metodo di test, quelli della funzione che crea il modello e quelli della funzione che genera la raccomandazione vera e propria. Questi tre input sono tutti facoltativi in quanto si può optare per l'utilizzo dei parametri di default. L'output della funzione sarà il risultato ottenuto, che verrà anche salvato come ultima voce dell'array Result, attributo dell'oggetto istanziato.

```
Result = fire([MethodParameters[, ModelParameters[,
    OnLineParameters]]])
```

Codice 5.11: Metodo per lanciare il test

5.4 Use Cases

La semplicità d'uso è stato il filo conduttore di tutta l'implementazione della piattaforma in esame. Proprio per questo motivo vengono ora presentati i casi d'uso che rispecchiano una fruizione tipica dell'applicazione.

5.4.1 Generazione della lista di raccomandazione

Per generare un suggerimento bisogna interagire con l'oggetto Recommendation. I passaggi da svolgere sono pochi:

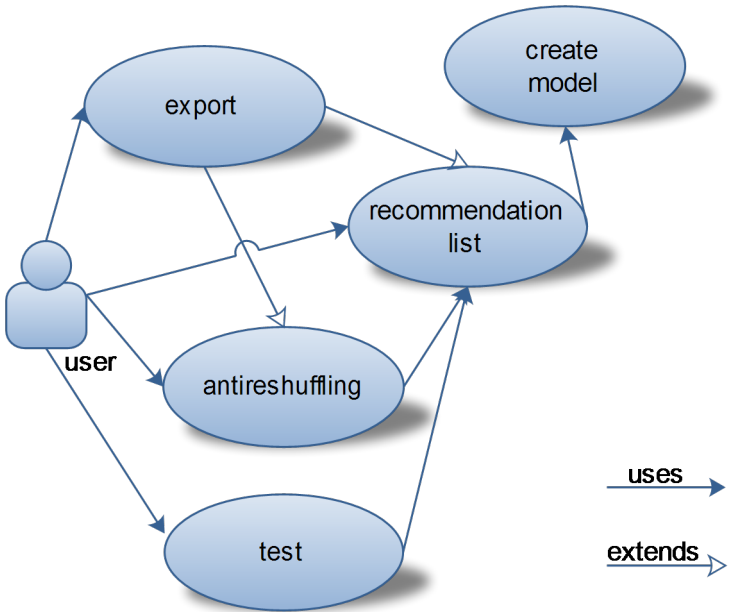

Figura 5.2: Gli use cases dell'applicazione

- si istanzia l'oggetto passando come parametri del costruttore la matrice URM e l'algoritmo scelto. Il sistema alloca lo spazio in memoria per l'oggetto e quindi controlla che i due input siano corretti; devono essere validi sia il nome della variabile che rappresenta la matrice, sia l'algoritmo di raccomandazione. Qualora non si verifichino queste condizioni, viene restituito un messaggio d'errore: l'oggetto è già stato istanziato e quindi bisogna procedere a settare gli attributi direttamente, senza ripassare dal costruttore. Quando i valori inseriti risultano essere idonei il sistema procede alla memorizzazione degli stessi.

- l'utente richiama il metodo che crea il modello. Il sistema va a cercare sul filesystem la funzione corrispondente e quindi la esegue, andando poi a memorizzare il risultato nell'attributo Model dell'oggetto.

- attraverso un profilo utente viene richiesta la generazione della lista di raccomandazione che successivamente è salvata nell'array Result in modo da poter essere riutilizzata in seguito.

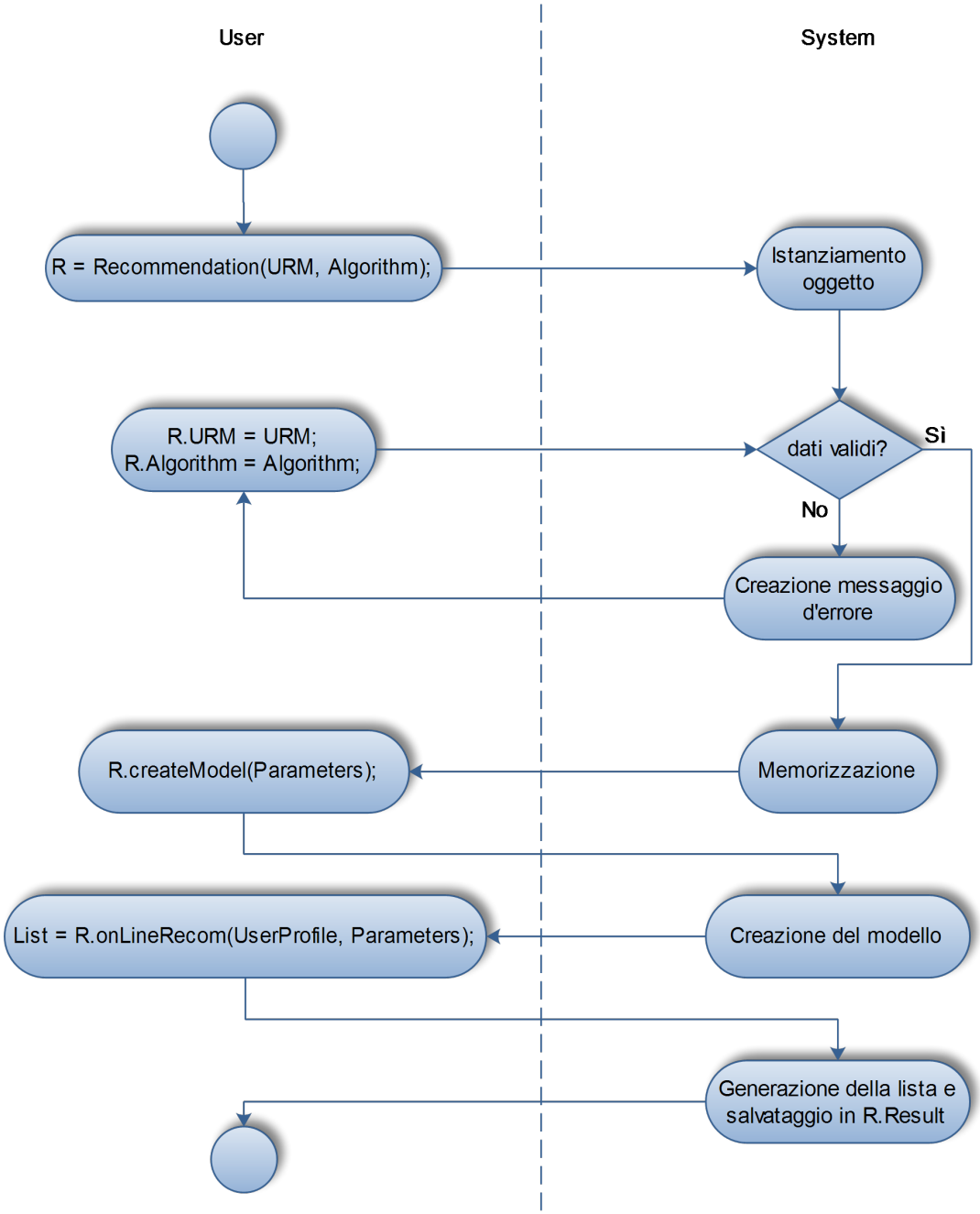

Figura 5.3: Use Case: generazione della lista di raccomandazione

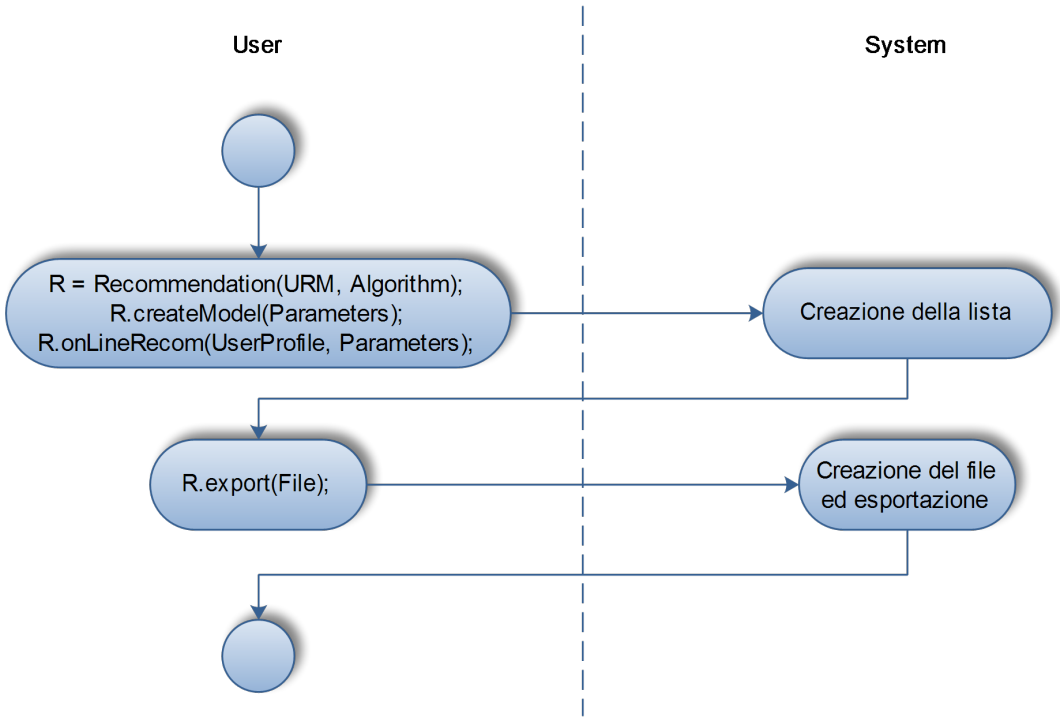

Figura 5.4: Use Case: esportazione della raccomandazione

5.4.2 Esportazione della raccomandazione

Ogni lista di suggerimenti può essere esportata in formato testale. Per far ciò è sufficiente:

- creare la raccomandazione con i passaggi visti in precedenza.
- richiamare il metodo di export indicando su quale file debba essere memorizzata la stringa rappresentante la lista. Si può anche specificare quale dei tanti risultati esportare mediante l'uso di un array di indici.

5.4.3 Antireshuffling

Si esamina ora la creazione di una lista di raccomandazioni tramite la tecnica di antireshuffling. Per far ciò bisogna prima generare due diverse liste di suggerimenti:

- si costruisce la prima lista inserendo un profilo utente di un momento prefissato. La lista conseguente viene memorizzata nell'array Result.

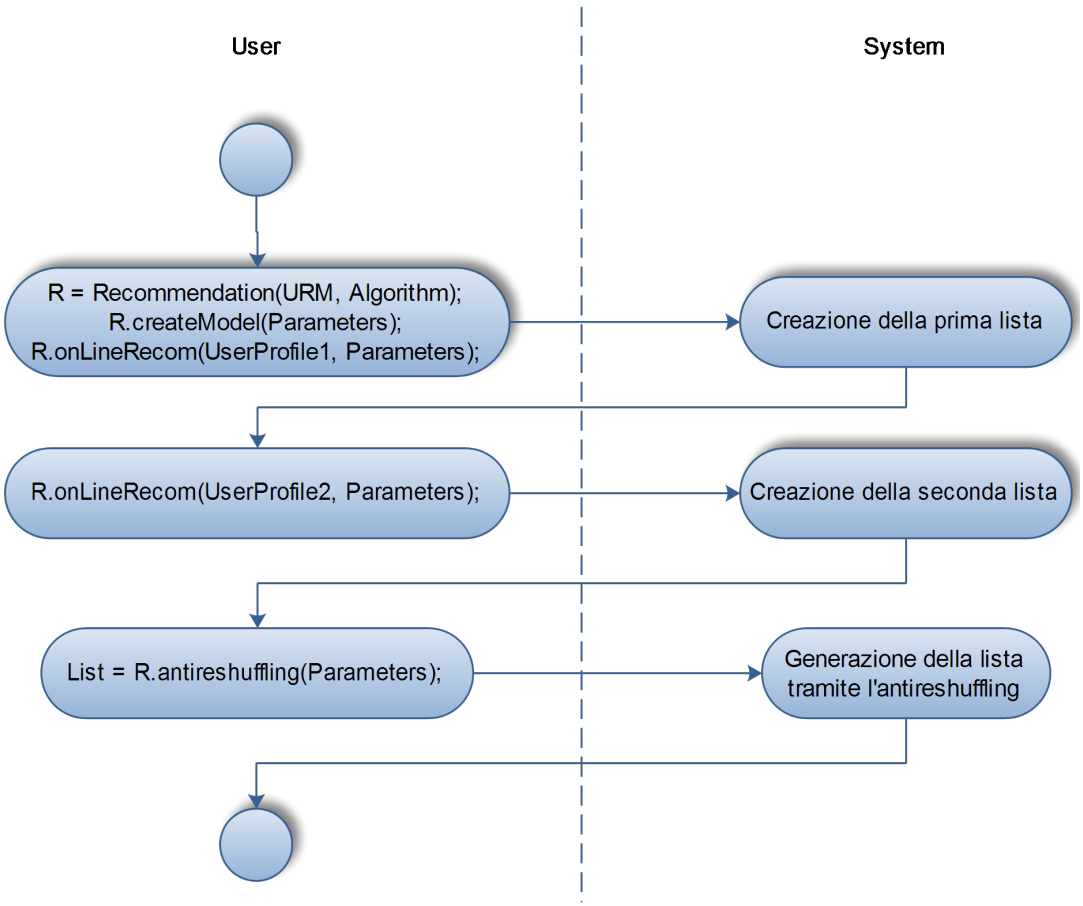

Figura 5.5: Use Case: antireshuffling

- si inserisce un secondo profilo utente che genera una nuova lista di raccomandazioni. Anche questa viene automaticamente salvata nell'attributo Result dell'oggetto.

- si richiama quindi la funzione di antireshuffling che, prese le ultime due liste generate, calcola la risultante. Questa viene sia restituita come output della funzione, sia memorizzata nell'array Result per essere eventualmente utilizzata in seguito.

5.4.4 Esecuzione di un test

In questo caso pratico si usa l'oggetto Test. Anche in questo caso l'interazione col sistema è molto semplice e intuitiva:

- prima di tutto si inizializza l'oggetto Test passando come parametri le due matrici URM e Probe, il nome dell'algoritmo di raccomandazione che si vuole utilizzare e la metodologia scelta. Il sistema quindi allocherà lo spazio in memoria e verificherà la correttezza dei dati: le matrici devono esistere nell'ambiente di base (o alternativamente come file nel filesystem), l'algoritmo e la metodologia devono essere validi. Nel caso in cui almeno uno di questi parametri non sia corretto viene restituito un messaggio d'errore e l'utente deve provvedere alla correzione manuale dell'attributo. Quando non ci sono più problemi, ogni attributo viene memorizzato.

- si richiama il metodo che fa partire il test, passando come input i vari parametri da utilizzare per la metodologia di test, per la creazione del modello e per la generazione della lista di suggerimenti. Il sistema processerà tutto quanto e, dopo aver svolto l'intero test, creerà un oggetto contenente i risultati memorizzandolo nell'array Result.

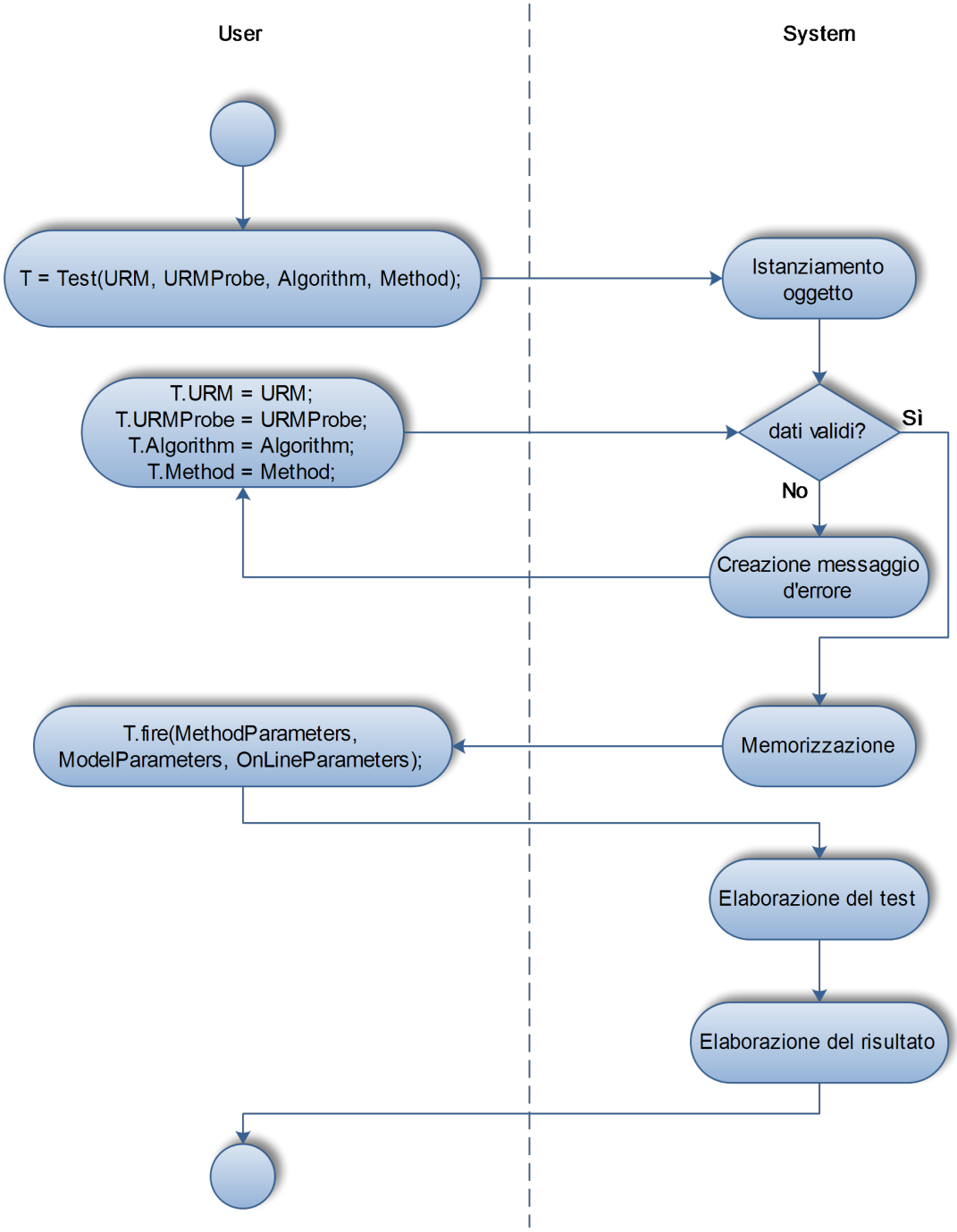

Figura 5.6: Use Case: esecuzione di un test

Capitolo 6

Integrazione in un Sistema Reale

Tutto quanto visto finora non è mai stato messo a confronto con un sistema reale: potrebbe praticamente servire una piattaforma simile fuori dall'ambito accademico? Quali sono le sue limitazioni e quali i suoi punti di forza?

Si cercherà ora di sottoporre questo sistema alle diverse esigenze che si possono incontrare al di fuori di un contesto di testing.

6.1 Integrazione

Il principio cardine di tutta la piattaforma è la facilità di implementazione: la struttura a oggetti permette una gestione semplice di tutto il flusso di dati in quanto non vengono allocate troppe variabili per memorizzare i vari stati dell'applicazione.

Anche i metodi sono stati pensati per rendere intuitivo l'utilizzo: non ne esistono di superflui, ognuno ha una sua precisa funzione. Tramite essi si può accedere a tutte le funzionalità dell'oggetto, senza dover avere una conoscenza approfondita degli attributi e dei parametri.

L'utente avanzato, invece, sarà interessato a tutti i diversi parametri per agire direttamente sul metodo: per esempio, la creazione di un modello può avvenire in modo differente in base alle opzioni che vengono passate. Questa gestione personalizzata può influire positivamente sulla riuscita di una particolare raccomandazione e proprio per questo durante l'implementazione si è cercato di tenere sempre in considerazione tutti i possibili parametri in modo da lasciare all'utente l'opzione

di variare alcune condizioni. Esemplificativo è il passaggio del modello ai metodi che lo creano: in questo modo la funzione ritorna subito evitando calcoli inutili.

6.2 Ampliamento e API

In campo informatico l'evoluzione è rapidissima ed è estremamente difficoltoso stare al passo con questo susseguirsi di aggiornamenti e nuove idee. Proprio per questo motivo la piattaforma è stata creata a moduli: ogni algoritmo di raccomandazione e ogni metodologia è una funzione a se stante, richiamata dal metodo, sulla quale non ci sono particolari controlli, a parte richiedere un output e certi parametri di input. Questa soluzione è stata adottata per consentire una libertà totale all'ampliamento del sistema: implementare un nuovo algoritmo è facilissimo e l'operazione viene riconosciuta subito dall'oggetto.

Analogamente s'è scelto di non creare costrutti rigidi per i parametri dei singoli metodi: si possono passare anche oggetti incredibilmente complessi, sta poi alla funzione chiamata saperli gestire. Esemplare da questo punto di vista è il modello della raccomandazione: all'oggetto non interessa come venga creato o come sia strutturato. È solo memorizzato e quindi passato alla funzione che genera la lista di suggerimenti. Quest'assenza di controllo rende possibile la gestione della struttura dati da parte dell'implementatore, che per questo motivo non è più costretto a "nascondere" i propri oggetti dentro strutture articolate come si era soliti fare in alcuni punti della vecchia applicazione.

6.3 Risorse Computazionali

La costruzione del modello è una fase computazionalmente molto impegnativa: il calcolatore deve analizzare e operare su migliaia di dati e il motore di Matlab non ne è sempre all'altezza. Le maggiori difficoltà si sono riscontrate con le matrici, in quanto la maggior parte degli algoritmi le analizza una riga alla volta: il programma, al contrario, le memorizza per colonne, e quindi risulta molto dispendioso estrarre un singolo elemento da una colonna. Proprio per questo una prima ottimiz-

zazione si è verificata introducendo le matrici trasposte, ma ancora in alcune fasi il sistema risulta essere troppo lento.

Le operazioni più lunghe sono state implementate in *C* in quanto Matlab supporta l'esecuzione di funzioni in questo linguaggio in modo quasi nativo (è sufficiente disporre di un compilatore sulla propria macchina). In questo modo il tempo di esecuzione di alcuni metodi è ridotto drasticamente, rendendo possibile anche la fase di testing (che richiede svariate creazioni di un modello) in tempi ragionevoli.

Nonostante questo, però, risulta ancora remota la possibilità di poter utilizzare una piattaforma Matlab in un sistema reale: si tratta di un linguaggio di livello troppo elevato e quindi troppo lento per poter accogliere le enormi quantità di dati dei sistemi attualmente in circolazione (IPTv e servizi di VoD). L'idea più semplice è eseguire un porting in C, ma al momento non risulta essere di particolare interesse in quanto si perderebbe tutta la comodità nella gestione dei dati numerici che la programmazione Matlab fornisce. Per ottenere strutture simili in altri linguaggi bisogna disporre di una conoscenza estremamente approfondita, mentre il principio su cui si basa la piattaforma è proprio la semplicità.

Senz'altro questa problematica può rappresentare un buon punto di partenza per successivi lavori di ottimizzazione, dal momento che bisognerebbe mettere mano a funzioni che interagiscono a coppie, e quindi che non dipendono dall'output di qualcun altro.

Capitolo 7

Conclusioni

La reingegnerizzazione dell'applicazione Matlab per i sistemi di raccomandazione è stata una tappa obbligata nello sviluppo della piattaforma.

I benefici sono evidenti:

Miglioramento nella gestione della memoria in quanto prima dell'intervento le matrici venivano passate alle varie funzioni e memorizzate in modo completo. Ora invece si cerca di utilizzare un sistema simile ai puntatori di alcuni linguaggi di programmazione.

Riduzione dello spazio in memoria poiché sono stati eliminati un incredibile numero di file inutili che erano stati creati solo come prove o fini a se stessi. A dimostrazione di ciò si è passati da una memoria sul filesystem di oltre $4.5Gb$ a poco più di $31Mb$[1].

Informazioni più accessibili dal momento che i vari risultati vengono memorizzati all'interno degli oggetti che li hanno prodotti. In questo modo non è più necessario, da parte dell'utente, utilizzare numerose variabili poiché ne è sufficiente una per ogni oggetto istanziato.

Espandibilità per quanto riguarda i nuovi algoritmi e le nuove metodologie di test. Grazie all'utilizzo di apposite API è possibile implementare nuove fun-

[1]Gran parte dello spazio richiesto dall'applicazione prima di questo lavoro era occupato da enormi matrici URM e ICM. Nella nuova piattaforma non sono state incluse, ma al loro posto sono state inserite matrici molto più piccole, ma sempre esemplificative

zioni che gli oggetti vanno a recuperare in modo automatico, senza che l'utente debba tutte le volte specificare il percorso dei file che gli interessano.

Facilità d'uso dal momento che ci si basa su un esiguo numero di comandi con i quali si riesce a compiere praticamente qualsiasi tipo di operazione. Starà poi all'utente esperto disporre dei parametri come meglio crede per ottenere un diverso livello di interazione con l'algoritmo.

Esportazione per poter visualizzare i risultati graficamente o importarli in altri sistemi in modo, ad esempio, di valutare più facilmente certe statistiche non implementate in questa sede.

7.1 Sviluppi futuri

Come epilogo di questo lavoro vengono inserite delle possibilità di sviluppo che si sono presentate durante lo studio dell'applicazione. Non tutte sono prioritarie, anzi, alcune possono definirsi quasi inutili, ma per dovere di cronaca vengono qui menzionate:

- Realizzazione di un'interfaccia grafica per consentire l'uso dell'applicazione non solo da riga di comando. Questa potrebbe aprire la piattaforma, ma anche i sistemi di raccomandazione in generale, a un pubblico sempre più vasto, ma soprattutto non prettamente di addetti ai lavori. Un'interfaccia grafica poi permetterebbe anche di avere un'idea più precisa di quello che si sta realizzando, sia per quanto riguarda le liste di suggerimenti, sia per quanto riguarda i risultati dei test.

- Esportare i risultati dei test in grafici, in modo da visualizzarne meglio l'andamento. In questo modo si vuole offrire un'alternativa all'esportazione testuale che è già presente, ma quasi inutile in considerazione della mole di risultati. Inoltre la codifica CSV adottata mal si presta a rappresentare enormi array di valori in quanto bisogna creare troppe convenzioni per i separatori dei dati.

- Una migliore implementazione degli algoritmi, in modo da considerare le matrici a colonne e non più a righe. Questo porterebbe a un incremento notevole nelle performance: l'ideale sarebbe implementare tutte le funzioni in C e quindi costruire in Matlab solamente "l'interfaccia". Parallelamente a questo lavoro, però, non sono stati condotti studi che ne verificassero l'effettivo miglioramento anche per operazioni brevi.

Riferimenti bibliografici

[1] Netflix. http://www.netflix.com/, . 34

[2] Netflix prize. http://www.netflixprize.com/, . 37

[3] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17:734–749, June 2005. ISSN 1041-4347. URL http://dx.doi.org/10.1109/TKDE.2005.99. http://dx.doi.org/10.1109/TKDE.2005.99.

[4] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22:143–177, January 2004. ISSN 1046-8188. URL http://doi.acm.org/10.1145/963770.963776. http://doi.acm.org/10.1145/963770.963776.

[5] The Apache Software Foundation. Hadoop, open-source software for reliable, scalable, distributed computing. http://hadoop.apache.org/, . 37

[6] The Apache Software Foundation. Mahout, a scalable machine learning and data mining library. http://mahout.apache.org/, . 37

[7] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3502-9. http://dx.doi.org/10.1109/ICDM.2008.22.

[8] Telematica Instituut/Novay. Duine framework. http://www.duineframework.org/. 36

[9] Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Trans. Knowl. Discov. Data*, 4(1):1–24, 2010. ISSN 1556-4681. http://doi.acm.org/10.1145/1644873.1644874. 21

[10] Sean Owen. Taste, open source recommendation engine in java. http://blogs.sun.com/plamere/entry/open_source_recommendation_engine_in. 37

[11] Matteo Restelli and Martino Zocca. Analisi e correzione dell'eccessiva dinamicità delle liste di raccomandazione. Master's thesis, Politecnico di Milano, 2010.

[12] Roberto Turrin and Paolo Cremonesi. Recommender systems for interactive tv. In *8th European Conference on Interactive TV and Video*, Tampere, Finland, June 2010.

[13] Mark van Setten, Jaap Reitsma, and Rogier Brussee. Open source recommender system software workshop. http://www.duineframework.org/osworkshop/index.html. 36