Sara Mangili

Design, implementazione e valutazione di una piattaforma distribuita per la raccomandazione di contenuti su Twitter

Tesi di Laurea Magistrale

Sapienza
Università di Roma

Sapienza Università di Roma
Luglio 2014

SAPIENZA
UNIVERSITÀ DI ROMA

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in INGEGNERIA INFORMATICA

Indirizzo ARCHITETTURE E SISTEMI DISTRIBUITI

DESIGN, IMPLEMENTAZIONE E VALUTAZIONE DI UNA PIATTAFORMA DISTRIBUITA PER LA RACCOMANDAZIONE DI CONTENUTI SU TWITTER

Tesi di

Sara Mangili

Matricola: 801448

Relatore	**Controrelatore**
Prof. Leonardo Querzoni	Prof. Massimo Mecella

Sessione Laurea 18 Luglio 2014

Anno Accademico 2013/2014

Ai miei genitori

la parte riflessiva e la parte attiva del mio universo,

stella polare del mio cammino.

Al mio fratello "Dioscuro",

la mia parte complementare.

Ai miei due "fratelli adottivi".

Ringraziamenti

Quando si finisce un lungo viaggio, quando si arriva alla meta desiderata, nasce il bisogno di ricordare il cammino percorso e ringraziare chi ti è stato vicino nei momenti felici e in quelli di sconforto.

Sono le persone, infatti, a colmare di significato anche le tappe più difficili.

Vorrei ringraziare i miei genitori, che con pazienza mi sono stati vicini nei momenti di euforia e in quelli di sconforto trovando sempre le parole giuste per farmi andare avanti.

Vorrei ringraziare mio fratello, che ha riempito le mie giornate di sorrisi e di saggi consigli e che ha saputo trasformare ogni giorno in un momento indimenticabile.

Vorrei ringraziare il Prof. Leonardo Querzoni, che con la sua professionalità, la sua passione e la sua pazienza ha saputo ispirarmi e guidarmi durante tutto il percorso di tesi; un particolare ringraziamento anche all'Ing. Fabio Petroni per il supporto strategico fornito nelle fasi più critiche di lavoro.

Sara

Indice

Elenco delle figure

Introduzione

Negli ultimi anni si è assistito ad un incremento esponenziale delle informazioni presenti nella rete, grazie alla comparsa di nuovi sistemi quali i *siti di e-commerce* o i *social network* caratterizzati da un elevata densità dei contenuti a disposizione degli utenti.

In particolare i social network rappresentano un nuovo modo di fare informazione: in tali sistemi, infatti, gli utenti non sono più osservatori passivi dei contenuti presenti ma assumono un ruolo attivo, diventando i principali produttori di informazioni.

I contenuti presenti nella rete, perciò, assumono progressivamente un alto livello di personalizzazione e diventano un modello esatto degli interessi espressi da una comunità in uno specifico contesto temporale o spaziale di osservazione.

Il corretto utilizzo di queste informazioni può fornire numerosi vantaggi ad individui o società interessati alla promozione e vendita di prodotti, alla diffusione globale di contenuti o alla realizzazione di un processo di fidelizzazione degli utenti; la conoscenza degli interessi degli utenti di un particolare sistema, infatti, permette di applicare strategie per la predizione degli acquisti e per la raccomandazione di oggetti di interesse non facilmente individuabili.

I sistemi di *Information Filtering* devono essere in grado di affrontare queste nuove sfide, implementando algoritmi orientati alla personalizzazione dell'esperienza di navigazione degli utenti; a questo scopo, perciò, nascono i *Sistemi di Raccomandazione* il cui obiettivo è quello di costruire un modello efficace degli interesi degli utenti da cui poter derivare le informazioni necessarie alla generazione di un insieme di raccomandazioni affidabili.

I Sistemi di Raccomandazione implementati all'interno dei social

network tipicamente utilizzano le soluzioni architetturali ed implementative sviluppate per sistemi informativi strutturalmente differenti, quali i siti di catalogazione di musica o libri, caratterizzati da un insieme di utenti e item inferiore e da un flusso informativo contenuto; i sistemi realizzati, quindi, sono soggetti ad una serie di limitazioni legate all'architettura che li caratterizza e agli aspetti implementativi degli algoritmi utilizzati:

- *Basso livello di personalizzazione delle raccomandazioni*: tipicamente gli algoritmi di raccomandazione utilizzati all'interno di un social network hanno come obiettivo l'individuazione di un sottoinsieme di utenti associati ad un'alta probabilità di condivisione di informazioni di interesse. Tale strategia di raccomandazione, orientata all'utente piuttosto che al singolo messaggio pubblicato, impedisce la differenziazione tra i contenuti ricevuti dallo stesso utente, che vengono associati ad un valore di interesse *globale*; le raccomandazioni generate in tale contesto, infatti, forniscono una serie di risultati ad alto livello, basati sull'ipotesi semplificativa che il modello di interesse di un utente sia un'entità statica.

  I messaggi pubblicati da ogni singolo utente all'interno del sistema, in realtà, sono eterogenei e contraddistinti da un valore di interesse univoco per l'utente target che dipende da fattori sociali, temporali o spaziali; la scelta di utilizzare una specifica raccomandazione, perciò, e la conseguente ricezione dell'intero flusso di messaggi dell'utente selezionato, determina un aumento dell'overhead relativo ad informazioni non significative e un peggioramento dell'esperienza d'uso dell'utente. Tale effetti negativi risultano più presenti all'interno di social network fortemente orientati allo scambio di contenuti, come

Twitter, nei quali l'interesse primario è quello di ricevere esclusivamente contenuti idonei al propio profilo.

- *Mancata diversificazione dei contenuti e spazio di raccomandazione ridotto*: una delle strategie adottate all'interno dei social network per aumentare il grado di personalizzazione delle raccomandazioni generate è quella di utilizzare un modello di algoritmo di tipo *Content Based*, basato sull'analisi del contenuto e dei metadati associati ai messaggi pubblicati. Tale strategia permette di realizzare un sistema di filtraggio del flusso di informazioni diretto ad ogni singolo utente, limitando lo spazio di ricerca dei messaggi da inserire all'interno della lista di raccomandazione ai soli contenuti compatibili con il profilo di interessi dell'utente. L'efficacia degli algoritmi Content Based, tuttavia, dipende dalla disponibilità e dalla precisione dei metadati associati a ciascun messaggio: l'insieme dei tag caratterizzanti, infatti, viene analizzato per determinare la presenza nella rete di contenuti associati ad un profilo compatibile con quello dell'utente target.

  L'utilizzo di informazioni accessorie quali i metadati al'interno del processo di raccomandazione può causare due differenti problematiche: da un lato l'assenza di metadati o il loro errato settaggio può comportare l'esclusione di specifici contenuti dal flusso di raccomandazione, dall'altro la presenza di un insieme troppo specifico di informazioni può determinare la generazione di raccomandazioni caratterizzate da un basso livello di diversificazione.

- *Modelli statici degli interessi degli utenti*: i social network sono sistemi altamente dinamici, nei quali l'insieme dei dati a disposizione e i profili degli utenti e degli item evolvono ad un ritmo costante; in tali

sistemi, quindi, è possibile definire un modello temporale degli interessi espressi da ogni singolo individuo, legato principalmente ai trend di voto più recenti. La maggior parte dei sistemi di raccomandazione, tuttavia, utilizza algoritmi basati su approcci di tipo statico, nei quali l'insieme degli interessi espressi viene posto sul medesimo piano temporale; tale strategia di fatto limita la capacità degli algoritmi di identificare l'insieme di dati *temporalmente* significativi, generando così una lista di raccomandazioni indipendente dal fattore tempo e contenente un insieme di item associati ad interessi espressi lungo tutto l'arco temporale di attività del sistema. Il mancato utilizzo del fattore temporale all'interno degli algoritmi implementati, perciò, può causare l'inserimento nel flusso di raccomandazione di item associati ad interessi non più significativi o il cui contenuto è ormai obsoleto.

- *Elevati costi computazionali e di memorizzazione*: la natura statica degli algoritmi di raccomandazione implementati all'interno dei social network si riflette anche sulle strategie di calcolo e di gestione delle strutture dati necessarie al processo di raccomandazione. La maggior parte degli approcci utilizzati, infatti, prevede la memorizzazione di ogni informazione associata ai profili di utenti ed item collezionata nell'intero periodo di osservazione del sistema e il suo utilizzo per il calcolo completo delle matrici di similarità. Il costo computazionale degli algoritmi implementati e il costo di memorizzazione dei dati di interesse, quindi, crescono esponenzialmente all'aumentare delle dimensioni del sistema nel tempo.

La finalità di questo lavoro di tesi, perciò, è quella di sviluppare un approccio innovativo al problema della raccomandazione nell'ambito dei social network, realizzando un sistema le cui proprietà lo rendano adatto

all'implementazione all'interno di scenari caratterizzati da flussi informativi elevati e fortemente dinamici.

In particolare l'approccio di progettazione scelto è orientato al raggiugimento di due obiettivi principali: la costruzione di un algoritmo di raccomandazione ad alto livello di personalizzazione, basato sulla raccomandazione di singoli contenuti agli utenti, e la gestione ottimizzata del flusso informativo a disposizione, attraverso l'applicazione di strategie di aggiornamento incrementale delle strutture dati di supporto e di eliminazione delle informazioni superflue.

Tra i differenti social network esistenti, Twitter rappresenta lo scenario ideale all'interno del quale sviluppare tale tipologia di Sistema di Raccomandazione. Questo particolare social network, infatti, è caratterizzato da una struttura orientata principalmente alla condivisione di contenuti secondo un approccio simile a quello dei *microblog*; la pubblicazione continua di informazioni, ciascuna associata ad una finestra temporale di interesse limitata, e l'elevata dimensione dello spazio degli utenti e degli item del sistema, perciò, richiedono la progettazione di algoritmi altamente flessibili e dinamici, in grado di adattarsi alle variazioni del pattern di interesse dei singoli utenti e di limitare il costo computazionale e di memorizzazione richiesto.

Il sistema realizzato nell'ambito della tesi, *Twitter Time-aware Recommendation System*, è un sistema di raccomandazione basato sull'analisi di informazioni di voto esplicite espresse dagli utenti; l'architettura progettata adotta un modello di tipo ibrido, costituito da due differenti componenti: una componente distributita rappresentata da un'estensione del browser *Chrome* installata sui nodi peer e una componente centralizzata rappresentata da un server centrale con funzioni di memorizzazione e

calcolo. L'estensione, in particolare, viene utilizzata per estendere le funzionalità di Twitter e consentire agli utenti l'assegnazione di un voto esplicito ai tweet visualizzati. L'algoritmo realizzato per generare le singole raccomandazioni utilizza un approccio di tipo *incrementale*, il cui scopo è quello di fornire un'esatta rappresentazione dell'evoluzione degli interessi degli utenti attraverso l'applicazione di una procedura di calcolo incrementale della similarità che ad ogni intervallo di osservazione si occupa di aggiornare le informazioni già presenti nel sistema, limitando il costo computazionale della generazione delle raccomandazioni; l'introduzione del fattore temporale nella fase di calcolo, inoltre, permette di progettare un modello flessibile degli interessi degli utenti, legato alle variazioni dei trend di voto.

Gli elementi caratteristici del sistema realizzato, quindi, come l'utilizzo di informazioni di voto esplicite, la presenza di strategie di raccomandazione temporali e la distribuzione di parte del carico computazionale sui nodi peer, cercano di fornire un valido strumento per il superamento dei limiti associati ai Sistemi di Raccomandazione classici, attraverso la definizione di un modello di raccomandazione orientato alla gestione dei singoli contenuti e alla riduzione delle risorse di calcolo impiegate.

L'organizzazione della tesi è la seguente. Nel Capitolo 1 viene presentata una rassegna dello stato dell'arte relativa ai Sistemi di Raccomandazione; in particolare, vengono analizzati le tre principali categorie di sistemi esistenti, i sistemi *Collaborative Filtering*, i sistemi *Content Based* e i sistemi *Ibridi*, evidenziando vantaggi e limitazioni di ciascun approccio. Nel Capitolo 2 viene presentata una rassegna dello stato dell'arte dei Sistemi di Raccomandazione *time-aware*. Nel Capitolo 3 viene presentato il sistema di raccomandazione *Twitter time-aware Recommendation System*, analizzando

le diverse scelte effettuate sul piano architetturale e sul piano implementativo. Nel Capitolo 4 vengono presentati i risultati ottenuti nella fase di test del sistema; in particolare i differenti elementi che caratterizzano il sistema di raccomandazione realizzato vengono valutati singolarmente, in modo da evidenziare i contributi in termini di prestazioni e di risorse utilizzate forniti al sistema finale. Nel capitolo conclusivo vengono sintetizzati i risultati ottenuti e viene proposto uno scenario di ricerca futuro.

Capitolo 1

Stato dell'arte per i Recommender Systems

I Recommender System [1] [2] [3] [4] [5] [6] rappresentano una sottoclasse dei sistemi di *Information Filtering* [7], il cui obiettivo principale è quello di ridurre l'information overhead presente nella rete attraverso metodi automatici e semiautomatici per la rimozione di informazioni ridondanti o non attinenti restituite a seguito di una query dell'utente. In particolare i Sistemi di Raccomandazione analizzano i *profili* degli utenti per restituire un insieme di oggetti di interesse o per predire il comportamento futuro dell'utente all'interno del sistema; tali profili vengono costruiti attraverso l'analisi di informazioni esplicitamente rilasciate dall'utente come voti o feedback sugli oggetti osservati o attraverso l'analisi di informazioni implicite come le abitudini di navigazione, le abitudini di acquisto, le parole chiave associate agli oggetti di interesse.

La capacità di tali sistemi di profilare con efficacia l'utente e di fornire contenuti altamente personalizzati li ha resi nel tempo adatti a numerosi contesti di applicazione; se, infatti, i primi Sistemi di Raccomandazione

vengono applicati all'interno di siti musicali per fornire agli utenti liste di brani di interesse, con il passare degli anni vengono introdotti nei siti di e-commerce, nei siti dedicati a libri, film o musica, fino ad arrivare a comparire all'interno dei social network, sotto forma di utenti da seguire o di pagine da leggere.

I siti di e-commerce rappresentano i sistemi che riescono a trarre numerosi benefici dall'introduzione dei sistemi di raccomandazione. La generazione di una lista di oggetti di interesse basata sulle interazioni (acquisti o visualizzazioni) effettuate nelle sessioni precedenti dall'utente, infatti, riesce da un lato ad aumentare il numero di oggetti venduti dal sito invogliando gli utenti ad effettuare acquisti multipli e dall'altro ad aumentare la fidelizzazione degli utenti fornendo una esperienza di navigazione personalizzata. La precisione con cui vengono generate le liste di raccomandazione permette, inoltre, di inserire nelle raccomandazioni di ciascun utente oggetti difficilmente rintracciabili attraverso la semplice navigazione ma compatibili con il profilo memorizzato; tale caratteristica permette di sfruttare appieno il principio della *long tail* [8], raccomandando agli utenti oggetti globalmente meno popolari ma più adatti ai gusti del singolo. Le liste di raccomandazione, infine, permettono di determinare in maniera efficace l'insieme di item più venduti e consentono, quindi, una gestione ottimizzata dei processi di produzione e stoccaggio degli oggetti.

Nel 2004 Herlocker [9] [10] identifica un insieme di funzioni orientate alla soddisfazione dell'utente finale per le quali un Recommender System può essere introdotto in un sistema:

Annotation in Context: il Recommender System viene integrato in un sistema esistente per analizzare il contesto delle informazioni presenti ed associare a ciascun oggetto una predizione che indichi all'utente se il

contenuto è di interesse o meno, filtrando eventualmente i messaggi con predizione peggiore; nell'ambito dei Web Recommender le predizioni effettuate vengono utilizzate per aggiungere alle informazioni esistenti dei link a pagine od oggetti di interesse per lo specifico utente.

Trovare Item di interesse: il Recommender System deve essere in grado di generare una lista di raccomandazione che contenga oggetti che soddisfino i gusti degli utenti, eventualmente associati al livello di gradimento di ciascun utente.

Trovare tutti gli item di interesse: in alcuni ambiti l'utente è interessato a ricevere tutti i risultati positivi associati ad una particolare ricerca e non il risultato filtrato che mostra solo gli item con predizione maggiore; in questi casi il Recommender System deve riuscire ad operare mantenendo basso il *false negative rate*.

Raccomandare una sequenza di item: nei siti che gestiscono brani musicali, libri o notizie il Recommender System deve essere in grado di generare una sequenza di oggetti che siano tutti di interesse per l'utente e non limitarsi a consigliare un solo oggetto alla volta.

Just Browsing: in molti sistemi gli utenti utilizzano i Recommender Systems allo scopo di visualizzare oggetti inusuali di interesse attraverso le raccomandazioni ricevute, senza avere intenzione di acquistare gli oggetti proposti; in queste situazioni l'utente viene soddisfatto maggiormente dalla presenza di una interfaccia di navigazione semplice e chiara, piuttosto che dalla precisione delle raccomandazioni ricevute.

Trovare Recommender Credibili: molti utenti non riescono ad accettare passivamente le raccomandazioni fornite dal sistema e cercano di

provare l'efficacia degli algoritmi di raccomandazione variando le informazioni di profilo fornite e verificando in che modo le liste di oggetti consigliati variano; in alcuni casi i Recommender Systems ottimizzati per restituire all'utente solo oggetti di interesse sicuramente non noti possono essere considerati inefficaci dall'utente finale in quanto non inseriscono nelle raccomandazioni oggetti che probabilmente l'utente già conosce ma non ha votato.

L'utente finale di un Sistema di Raccomandazione rappresenta solo uno dei parametri in gioco per generare una lista di raccomandazione o delle predizioni efficaci ; in particolare l'input utilizzato dalla maggior parte dei sistemi di raccomandazione è costituito da tre macro-insiemi:

- *Utenti*: gli utenti finali del Sistema di Raccomandazione per i quali vengono generate le liste di raccomandazioni o le predizioni.
- *Item*: gli oggetti consigliati agli utenti nelle liste di raccomandazione.
- *Voti*: i voti espressi dagli utenti sul set di item noti.

Ogni sistema gestisce e mantiene tali macro-insiemi in maniera differente, in base alla tipologia di algoritmo di raccomandazione utilizzato. Generalmente gli utenti e gli item vengono associati ad un profilo costituito per gli utenti da informazioni demografiche, gusti personali, voti espressi, abitudini di navigazione o di acquisto e per gli item da categorie di appartenenza, anno di produzione, popolarità globale. Tali profili vengono utilizzati, in combinazione con i voti espressi, per generare l'output del Sistema di Raccomandazione, costituito in alcuni casi da liste di raccomandazione e in altri da un insieme di predizioni per un set di oggetti.

Formalmente si può, quindi, definire un Recommender System come [1]:

- C: insieme di tutti gli utenti
- S: insieme di tutti gli item
- U: funzione utilità che misura il grado di utilità di uno specifico item per l'utente

$$U : C \times S \rightarrow R \tag{1.1}$$

L'insieme dei possibili item di interesse per un utente in alcuni casi può essere costituito da migliaia o milioni di item quindi è necessario scegliere per ciascun utente l'item o il set di item che massimizza la funzione di utilità:

$$\forall c \in C, s_c^{'} = \underset{s \in S}{argmax}\, u(c, s) \tag{1.2}$$

I voti espressi dagli utenti, perciò, rappresentano il mezzo più efficace per calcolare le utilità di ogni singolo item per lo specifico utente; le modalità con cui tali voti vengono utilizzati per produrre le raccomandazioni finali determinano la tipologia di Recommender System implementata.

1.1 Content Based Recommender Systems

I metodi di raccomandazione Content Based [11] [12] basano il proprio funzionamento sulla costruzione e sull'analisi dei profili di utenti e item. Tale metodologie utilizzano in parte l'approccio sviluppato all'interno dei sistemi di *Information Retrival*, in cui ciascun item viene associato ad un profilo costituito dall'insieme di parole chiave che meglio lo descrivono, e lo estendono associando anche agli utenti un profilo costituito dalle preferenze espresse esplicitamente e dagli item di interesse visualizzati in passato. L'aspetto chiave delle tecniche Content Based è quello di raccomandare a ciascun utente oggetti simili a quelli precedentemente visualizzati; per questo

motivo l'utilità di ciascun item presente nel sistema viene determinata in base a quanto il profilo dell'item si adatta al profilo costruito per l'utente.

Per determinare la similarità tra profili in contesti basati su keyword ed informazioni testuali, quali i sistemi Content Based, è possibile applicare metodologie di misura già sviluppate nell'ambito delle tecniche di *Information Retrival* come la funzione di peso *term frequency–inverse document frequency* [13]. Tale misura si basa sull'idea che il peso di un termine aumenti in maniera proporzionale al numero di volte che appare nel documento in esame e decresca in maniera inversamente proporzionale al numero di volte che appare negli altri documenti; in particolare la misura è esprimibile come il prodotto di due differenti termini [12]:

$$TF - IDF(t_k, d_j) = \underbrace{TF(t_k, d_j)}_{TF} \cdot \underbrace{log\frac{N}{n_k}}_{IDF} \tag{1.3}$$

dove N è il numero di documenti totali del sistema, n_k è il numero di documenti in cui il termine t_k compare almeno una volta e d_j è il documento in esame.

Il termine *term frequency* è pari a :

$$TF(t_k, d_j) = \frac{f_{k,j}}{max_x f_{z,j}} \tag{1.4}$$

dove il massimo è calcolato sulle frequenze $f_{z,j}$ di tutti i termini t_z che compaiono nel documento d_j.

La *TF-IDF*, perciò, permette di calcolare un valore di similarità tra profili o documenti che non dipende principalmente dall'insieme di termini di uso frequente condivisi ma dall'insieme di termini più rari e quindi più significativi che due item hanno in comune.

$$sim(d_i, d_j) = \frac{\sum_k w_{ki} \cdot w_{kj}}{\sqrt{\sum_k w_{ki}^2} \cdot \sqrt{\sum_k w_{kj}^2}} \tag{1.5}$$

$$w_{k,j} = \frac{TF - IDF(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} TF - IDF(t_s, d_j)^2}} \tag{1.6}$$

In alcuni scenari, quali la ricerca di paper scientifici, tale caratteristica consente all'utente di ricevere come raccomandazione un set di item legati allo stesso topic di interesse, sebbene in altri ambiti penalizzi la *diversificazione* delle raccomandazioni e impedisca all'utente di esplorare contesti differenti da quello attuale. La similarità calcolata tra gli item o gli utenti attraverso delle funzioni di utilità come la *tf-idf* viene sostituita in alcuni sistemi da tecniche di classificazione quali l'applicazione di *Classificatori Bayesiani*, la costruzione di *Alberi di Decisione* o di *Reti Neurali* o l'applicazione di *Tecniche di Clustering* [11].

1.1.1 Limitazioni

Le tecniche di raccomandazione Content Based presentano una serie di limitazioni che le rendono non utilizzabili in tutti gli scenari possibili [12].

Limited Content Analysis : le misure di similarità o di classificazione applicate in un tradizionale Sistema di Raccomandazione Content Based basano il proprio funzionamento sull'analisi e sul confronto dell'insieme di *keyword* che definiscono i profili di utenti e item; tale tipologia di analisi testuale non può essere applicata in contesti in cui gli oggetti di interesse sono costituiti da dati multimediali come immagini o stream video. Una soluzione a questo problema può essere individuata nell'assegnazione da parte degli utenti di una serie di metadati testuali agli item inseriti nel sistema o votati; tale strategia, però, non è sempre efficace in quanto non tutti gli utenti collaborano attivamente con i Recommender System e il costo di memorizzazione

di dati aggiuntivi può essere elevato in sistemi con risorse limitate. I Sistemi di Raccomandazione Content Based, inoltre, non riescono a distinguere tra item differenti quando questi sono descritti dalle stesse keyword né riescono a determinare all'interno di un set di item associati a keyword di interesse gli item di qualità superiore.

Over-specialization: i sistemi Content Based costruiscono le liste di raccomandazione selezionando l'insieme di item con similarità più alta rispetto al profilo dell'utente; tale strategia riduce la *diversificazione* degli item raccomandati che generalmente appartengono ad un insieme omogeneo di alternative. L'introduzione di un fattore di casualità nella generazione della lista di raccomandazione permette di superare tale limitazione.

New User Problem: per poter costruire un profilo preciso delle preferenze di un utente e generare raccomandazioni affidabili il sistema deve collezionare un insieme sufficientemente ampio di informazioni su ciascun utente; tali informazioni principalmente derivano dall' interazione che l'utente ha con il sistema, come la visualizzazione di determinati item o l'apprezzamento per un insieme di topic. Gli utenti entrati da poco nel sistema, perciò, non riescono ad ottenere raccomandazioni affidabili a causa dell'incompletezza del loro profilo.

1.2 Collaborative Filtering

I Sistemi di Raccomandazione Collaborative Filtering [14] [10] adottano una strategia di filtraggio delle informazioni che emula la tendenza naturale dell'utente a visualizzare gli item apprezzati dall'insieme di persone con gusti simili ai propri; in particolare, a differenza dei sistemi Content Based

che generano le raccomandazioni basandosi sulle preferenze espresse solo dall'utente in passato, i sistemi di tipo Collaborative Filtering utilizzano le opinioni tutti gli utenti presenti nel sistema.

Il primo sistema di Collaborative Filtering, Tapestry [15], viene sviluppato nel 1990 per introdurre nei Sistemi di Raccomandazione la possibilità di valutare qualitativamente gli item inseriti nella lista di raccomandazione differenziandoli in base all'ordine di importanza, caratteristica non presente nei classici Sistemi di Raccomandazione Content Based. Tale sistema implementa un database di informazioni testuali a cui ciascun utente può associare una serie di metadati aggiuntivi, come autore, data del messaggio e opinione/voto, che vengono successivamente utilizzati nell'ambito delle ricerche per restituire risultati affidabili. Tapestry si basa su una metodologia di raccomandazione di tipo *pull based*: ogni utente deve attivamente analizzare i metadati associati a ciascun item di interesse, in particolare le opinioni inserite dagli altri utenti, per costruire la propria lista di raccomandazione.

Un approccio completamente opposto basato su una strategia di raccomandazione di tipo *push based* viene implementato in Lotus Notes da Maltz ed Ehrlich: ogni utente individua le persone interessate ad un set di specifici item in base alle preferenze espresse da questi precedentemente e si fa carico di inviare o segnalare personalmente tali item [16].

Sia la strategia di raccomandazione *push based* che la strategia di raccomandazione *pull based* vengono implementate in una categoria di Recommender System definiti *attivi*, in cui ciascun utente deve conoscere la rete di utenti in cui è inserito, insieme alle loro preferenze e all'affidabilità delle loro opinioni, per poter generare o ricevere raccomandazioni attendibili; la rapida crescita del numero di utenti e del numero di item appartenenti

ai Recommender System, tuttavia, ha richiesto nel tempo lo sviluppo di un approccio *passivo* per la generazione delle raccomandazioni, che prevede lo svolgimento di un ruolo da intermediario da parte del sistema stesso, attraverso l'acquisizione delle informazioni sulla rete e sui profili presenti.

I Sistemi di Raccomandazione di tipo Collaborative Filtering assolvono a due differenti funzioni:

- **Predizione dei voti:** il Sistema di Raccomandazione cerca di predire l'opinione che un utente ha rispetto ad un item non ancora visualizzato; tale opinione viene espressa mediante un valore numerico che rappresenta il voto previsto per l'utente.
- **Generazione della lista di raccomandazione:** il Sistema di Raccomandazione determina per l'utente gli item con il voto previsto più alto e li inserisce in una lista di raccomandazione personalizzata.

Per poter eseguire tali compiti i sistemi Collaborative Filtering utilizzano l'insieme di opinioni espresse sotto forma di voto da ciascun utente sul set di item visualizzati. In genere la rappresentazione dei voti degli utenti dipende dalla natura del Sistema di Raccomandazione in esame: la maggior parte dei sistemi adotta una scala di voto discreta che permette di esprimere il livello di apprezzamento dell'utente per un particolare item, anche se esistono Sistemi di Raccomandazione che utilizzano una strategia di voto unario 0/1 per indicare la visualizzazione o meno di uno specifico item o l'applicazione di una tipologia di voto binario 1/-1 che indica due opinioni opposte come il *"like"* e il *"dislike"*.

All'interno del Sistema di Raccomandazione i voti vengono rappresentati mediante una matrice di tipo utente/voto: ciascuna entry i,j della matrice

contiene un valore numerico che rappresenta, rispetto alla scala scelta, il voto espresso dall'utente *i-esimo* per l'item *j-esimo*.

	i_1	i_2	...	i_j	...	i_m
u_1	...	...	...	...	...	...
u_2	...	...	...	...	...	...
...	...	...	...	...	...	...
u_i	...	...	...	$r_{i,j}$	...	...
...	...	...	...	...	...	...
u_n	...	...	...	...	...	...

Tabella 1.1: Matrice Utenti-Item

In mancanza di voti per uno specifico item la relativa entry viene impostata al valore 0; per questo motivi nei sistemi reali, in cui la dimensione del set di utenti e la dimensione del set di item è elevata, la matrice dei voti è una matrice altamente sparsa.

I sistemi di tipo Collaborative Filtering basano gli algoritmi di predizione o di raccomandazione sulla costruzione di un set di utenti con abitudini di voto o preferenze simili rispetto all'utente attivo; in quest'ottica la matrice dei voti rappresenta la struttura dati fondamentale attraverso la quale determinare il *grado di similarità* tra ciascun utente.

Le tecniche di analisi della matrice dei voti e di trasformazione dei valori numerici contenuti in predizioni o raccomandazioni varia in base al modello di sistema di Collaborative Filtering implementato. Generalmente è possibile suddividere tali sistemi in due differenti categorie: modelli di tipo *memory based* e modelli di tipo *model based*.

1.2.1 Memory Based

I Sistemi di Raccomandazione Collaborative Filtering di tipo *Memory Based* [1] [17] [18] [19] hanno la caratteristica di mantenere in memoria le informazioni associate ad ogni utente, item o voto presente nel sistema; tali informazioni costituiscono la base di conoscenza su cui l'algoritmo di predizione o di raccomandazione lavora. Idealmente i sistemi di tipo *memory based* devono essere in grado di generare l'insieme di predizioni in maniera efficiente processando tutte le informazioni contenute nella matrice utenti/item. I Sistemi di Raccomandazione reali, tuttavia, sono caratterizzati da una elevata dimensione dello spazio degli utenti e degli item che rende estremamente costosa la fase di lettura ed analisi della matrice utenti/item. Per superare tali limitazioni mantenendo elevata l'affidabilità delle predizioni e delle raccomandazioni generate gli algoritmi di tipo *memory based* basano il proprio funzionamento sull'idea di determinare un grado di similarità tra gli utenti che permetta di estrapolare dalla matrice dei voti le informazioni associate ai soli utenti simili all'utente attivo.

Similarità

La similarità tra utenti può essere interpretata come una misura di distanza che esprime il livello di vicinanza tra i gusti degli utenti del sistema e tipicamente viene calcolata considerando l'insieme di item votati in comune da coppie di utenti. Ciascun Sistema di Raccomandazione può applicare una qualsiasi funzione di distanza per il calcolo delle similarità anche se le misure più frequentemente adottate sono il coefficiente di similarità di Jaccard, il coseno di similarità e il coefficiente di correlazione di Pearson [20].

Coefficiente di Jaccard: Il coefficiente di similarità di Jaccard rappresenta la misura di similarità più semplice da calcolare. Data

una coppia di utenti x e y, il valore del coefficiente di Jaccard associato è pari alla dimensione dell'intersezione dell'insieme di item votati divisa per la dimensione dell'unione.

$$sim(x, y) = J(x, y) = \frac{|I_x \cap I_y|}{|I_x \cup I_y|} \tag{1.7}$$

Coseno di similarità: Il coseno di similarità è una misura di similarità sviluppata nel campo dell'*Information Retrival* per confrontare coppie di documenti attraverso i loro vettori di frequenza dei termini. Nell'ambito dei Recommender System viene modificata sostituitendo i documenti con gli utenti del sistema e i vettori dei termini con i voti espressi; ciascun utente, quindi, viene associato ad un vettore di voti in uno spazio *m-dimensionale*, dove m è il numero di item covotati dagli utenti per i quali si calcola similarità. Il valore di similarità ottenuto è pari al coseno dell'angolo tra tali vettori ed assume un valore compreso tra *0* (perfetta dissimilarità) e *1* (perfetta similarità) nel caso in cui il range di voti esprimibili sia esclusivamente positivo e tra *-1* e *1* nel caso in cui il sistema consenta l'utilizzo di voti negativi.

$$\begin{aligned} sim(x, y) = cos(\vec{x}, \vec{y}) &= \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \times \|\vec{y}\|} \\ &= \frac{\sum_{j \in I_x \cap I_y} r_{x,j} r_{y,j}}{\sqrt{\sum_{j \in I_x} r_{x,j}^2} \sqrt{\sum_{j \in I_y} r_{y,j}^2}} \end{aligned} \tag{1.8}$$

Nei sistemi in cui è presente la possibilità di esprimere un voto compreso in un range di valori discreti tale formula deve essere modificata per tenere conto delle differenti scale di voto utilizzate dagli utenti; in molti casi, infatti, è possibile individuare insiemi di utenti che tendono a utilizzare voti medi *alti* rispetto al range di voti assegnabili e insiemi di utenti che tendono a utilizzare voti medi *bassi*.

L'*adjusted cosine similarity* cerca di superare tale problema modificando la formula originale della misura di similarità e introducendo nel calcolo il valore del voto medio espresso da ciascun utente; tale valore viene sottratto al voto di ogni item in modo da normalizzare le scale di voto usate.

$$\begin{aligned} sim(x,y) = cos(\vec{x},\vec{y}) &= \frac{\vec{x}\cdot\vec{y}}{\|\vec{x}\|\times\|\vec{y}\|} \\ &= \frac{\sum_{j\in I_x\cap I_y}(r_{x,j}-\overline{r_x})(r_{y,j}-\overline{r_y})}{\sqrt{\sum_{j\in I_x}(r_{x,j}-\overline{r_x})^2}\sqrt{\sum_{j\in I_y}(r_{y,j}}-\overline{r_x})^2} \end{aligned} \tag{1.9}$$

Correlazione di Pearson: Il coefficiente di correlazione di Pearson utilizza un approccio simile a quello sviluppato per l'*adjusted cosine similarity* ma sostituisce nella formula il voto medio espresso dall'utente con il voto medio assegnato all'item considerato.

$$sim(x,y) = \frac{\sum_{j\in I_x\cap I_y}(r_{x,j}-\overline{r_j})(r_{y,j}-\overline{r_j})}{\sqrt{\sum_{j\in I_x}(r_{x,j}-\overline{r_j})^2}\sqrt{\sum_{j\in I_y}(r_{y,j}}-\overline{r_j})^2} \tag{1.10}$$

Le differenti misure di similarità possono essere applicate nei Sistemi di Raccomandazione per individuare similarità tra utenti o similarità tra item; nel primo caso si parla di sistemi di tipo *user based* mentre nel secondo caso si parla di sistemi di tipo *item based* [21] [22].

Sistemi User Based

Nei sistemi *User Based* [2] la funzione di similarità viene applicata all'insieme degli utenti del sistema attraverso il confronto degli item votati in comune da coppie di utenti.

$$sim(x,y) = \frac{\sum_{j\in I_x\cap I_y}(r_{x,j}-\overline{r_j})(r_{y,j}-\overline{r_j})}{\sqrt{\sum_{j\in I_x}(r_{x,j}-\overline{r_j})^2}\sqrt{\sum_{j\in I_y}(r_{y,j}}-\overline{r_j})^2} \tag{1.11}$$

I valori di similarità così ottenuti vengono utilizzati all'interno dell'algoritmo di predizione/raccomandazione come peso per i voti espressi dagli utenti sullo specifico item; in tale modo l'algoritmo riesce ad assegnare a ciascun voto un peso proporzionale al grado di similarità esistente tra gli utenti [10].

$$pred(u,i) = \frac{\sum_{n \in Users} sim(u,n) \cdot r_{ni}}{\sum_{n \in Users} sim(u,n)} \tag{1.12}$$

Nei sistemi reali, caratterizzati da milioni di utenti, il calcolo dei coefficienti di similarità per l'intero insieme di utenti può risultare computazionalmente costoso e generare notevoli rallentamenti nell'esecuzione degli algoritmi di raccomandazione/predizione. Una delle strategie tipicamente adottate per superare tale limitazione si basa sull'osservazione che l'insieme dei peer di un qualsiasi sistema non varia drasticamente nel tempo e quindi è possibile utilizzare dei valori precalcolati dei coefficienti di similarità per generare raccomandazioni affidabili. I Sistemi di Raccomandazione, perciò, calcolano i valori di similarità per l'intero set di utenti ad intervalli regolari nel tempo ed utilizzano negli algoritmi di predizione i valori ottenuti nell'esecuzione più recente dell'algoritmo di calcolo della similarità.

L'applicazione delle differenti tecniche di calcolo della similarità può generare valori inconsistenti in situazioni nelle quali il numero di voti presenti nel sistema è ridotto e quindi l'algoritmo di similarità opera su una matrice altamente sparsa; sono state elaborate, perciò, varie strategie per estendere la matrice dei voti e migliorare le tecniche standard di calcolo della similarità [17] [19] [18].

Significance Weighting: nei Sistemi di Raccomandazione possono verificarsi casi in cui coppie di utenti hanno votato in comune un set estremamente ridotto di item; le similarità calcolate in queste situazioni

risultano scarsamente significative e possono generare predizioni non affidabili. Per poter superare tale limitazione si associa ad ogni similarità un *fattore di significance* che dipende strettamente dal numero di item covotati dagli utenti e che permette di diminuire il peso dei valori di similarità calcolati su un insieme di item più piccolo di una dimensione k prefissata per il sistema in esame [23] [24].

$$sim^{'}(x,y) = \begin{cases} sim(x,y) \cdot 1 & \text{se } n > k \\ sim(x,y) \cdot \frac{n}{k} & \text{se } n < k \end{cases} \tag{1.13}$$

dove n è il numero di item covotati dagli utenti x e y.

Default Voting: una tecnica alternativa al significance weighting che permette di aumentare la densità della matrice dei voti e quindi rendere più affidabile il valore di similarità calcolato per coppie di utenti consiste nell'inserire nella matrice utenti/item un insieme di voti di default per item non ancora votati; tipicamente il valore scelto per tali voti corrisponde alla media di voto dell'utente considerato.

Inverse User Frequency: tipicamente nei Sistemi di Raccomandazione esiste un insieme di item votati positivamente o negativamente dalla maggior parte degli utenti; nel calcolo dei coefficienti di similarità tali item *"popolari"* contribuiscono con lo stesso peso rispetto agli item meno popolari votati in comune dagli utenti. Nei casi in cui gli insiemi di item covotati da coppie di utenti sono costituiti da un numero elevato di item *"popolari"*, i valori di similarità non riescono a cogliere i reali interessi degli utenti e rappresentano, invece, una tendenza comune di voto. Per superare tale limitazione è possibile applicare un coefficiente di *inverse user frequency* che, moltiplicato per il valore originario del

voto espresso su un item, ne diminuisce il peso all'aumentare del numero di utenti che hanno votato l'item stesso.

L'inverse user frequency viene definito come $log\frac{n}{n_i}$ dove n è il numero totale di item del sistema e n_i è il numero di utenti che hanno votato l'item i; quando tutti gli utenti hanno votato l'item i-esimo il coefficiente, e quindi il voto, assumono valore 0.

Case amplification: negli algoritmi di raccomandazione/predizione ciascun voto contribuisce alla predizione finale in maniera proporzionale al peso (similarità) associato; nei casi in cui numerosi voti siano associati ad utenti con valori di similarità bassi o trascurabili si introduce nel voto finale un *fattore di rumore* che può diminuire l'affidabilità dell'algoritmo. Per ridurre tale rumore ed assegnare maggiore importanza ai voti associati a similarità più elevate è possibile utilizzare un *coefficiente di amplificazione* ρ che riesce a rendere trascurabile il contributo di voti associati a similarità più basse, mantenendo inalterato il contributo dei voti associati a similarità elevate.

$$w'_{x,y} = w_{x,y} \cdot |w_{x,y}|^{\rho-1} \tag{1.14}$$

Sistemi Item Based

Nei sistemi *Item Based* [2] [25] i valori di similarità vengono determinati attraverso il confronto dei voti assegnati dagli utenti a coppie di item *i,j.*

$$sim(i,j) = \frac{\sum_{u \in U_i \cap U_j}(r_{u,i} - \overline{r_i})(r_{u,j} - \overline{r_j})}{\sqrt{\sum_{u \in U_i}(r_{u,j} - \overline{r_j})^2}\sqrt{\sum_{u \in U_j}(r_{u,j} - \overline{r_j})^2}} \tag{1.15}$$

L'algoritmo di predizione/raccomandazione definisce, quindi, il voto previsto per l'item i-esimo come media dei voti espressi dall'utente attivo

sui restanti item moltiplicati per il relativo coefficiente di similarità [10] .

$$pred(u,i) = \frac{\sum_{j \in ratedItems(u)} sim(i,j) \cdot r_{ui}}{\sum_{j \in ratedItems(u)} sim(i,j)} \tag{1.16}$$

Un limite dei sistemi item based è rappresentato dalla dimensione del modello di computazione che è pari al quadrato del numero di item presenti; tale dimensione nei sistemi caratterizzati dall'introduzione continua di nuovi item causa un peggioramento progressivo delle performance. Una possibile soluzione a questo problema consiste nel mantenere in memoria solo le coppie di item caratterizzate da un numero di voti maggiore di una soglia prefissata k; ciò permette di ridurre sensibilmente la dimensione della matrice di similarità ma può determinare l'impossibilità di effettuare predizioni per alcuni item.

Sistemi Neighborhood based

Sia i sistemi di tipo User Based che i sistemi di tipo Item Based implementano algoritmi di raccomandazione le cui performance in termini di occupazione di memoria e tempi di processamento sono strettamente legati alla dimensione dell'insieme degli utenti e alla dimensione dell'insieme degli item; in alcuni scenari in cui sono noti a priori tali parametri è possibile adottare la tipologia di sistema legata all'insieme di dimensione minore per migliorare le performance, applicando algoritmi di tipo User Based quando il numero di utenti è limitato e costante o algoritmi di tipo Item Based quando il numero di item è trascurabile rispetto al numero di utenti.

Una strategia generale applicabile per ridurre la complessità della fase di predizione consiste nel selezionare all'interno della matrice dei voti i soli contributi associati al sottoinsieme di utenti/item con similarità più elevate rispetto all'utente/item attivo; tali tecniche vengono definite *neighborhood*

based e si basano sull'idea che i contributi più significativi per la generazione di una predizione provengano dagli utenti con gusti più vicini a quelli dell'utente attivo. I modelli di tipo neighborhood based suddividono gli algoritmi di predizione/raccomandazione in tre fasi distinte:

- Nella prima fase viene applicata una funzione di distanza per determinare i coefficienti di similarità tra gli utenti del sistema; le formule tipicamente utilizzate sono il calcolo del coefficiente di Pearson in sistemi di tipo User Based o il coseno di similarità in sistemi di tipo Item Based.

- Nella seconda fase per ogni utente attivo vengono selezionati i k utenti con valori di similarità più alti che andranno a costituire il neighborhood; il coefficiente k varia da sistema a sistema e tipicamente viene settato ad un valore pari alla radice quadrata del numero di utenti nei sistemi di tipo User Based o degli item nei sistemi di tipo Item Based.

- Nella terza fase viene calcolato il voto previsto di un item per l'utente attivo attraverso una combinazione pesata dei voti degli utenti appartenenti al neighborhood:

$$r_{ui} = \frac{\sum_{v \in N(u,i)} sim(u,v) \cdot r_{vi}}{\sum_{v \in N(u,i)} sim(u,v)} \tag{1.17}$$

L'elemento chiave che determina l'affidabilità delle raccomandazioni o predizioni generate è la corretta selezione degli utenti all'interno del neighborhood; la dimensione e la tipologia dell'insieme di utenti scelti influenza, infatti, le performance dell'algoritmo di calcolo della predizione e la correttezza del risultato. In generale è possibile utilizzare due differenti

tecniche per stimare la dimensione del neighborhood di utenti: la *correlation thresholding* e la *best-n-neighbors* [26].

La **Correlation Thresholding** prevede di limitare gli utenti inseriti all'interno del neighborhood a quelli con un valore di similarità superiore ad un fissato valore di soglia. Tale tecnica è sensibile al valore di soglia scelto: un valore troppo alto riduce sensibilmente il numero di utenti scelti e possono verificarsi situazioni in cui la dimensione ridotta dell'insieme di neighbors impedisce la predizione di voto per determinati item; un valore troppo basso, al contrario, elimina i benefici dell'utilizzo di un sottoinsieme di utenti per la predizione dal momento che la dimensione di tale insieme può risultare elevata.

La tecnica **Best-n-neighbors** prevede la selezione, per ciascun utente attivo, degli n utenti con similarità maggiore. Tale tecnica in generale offre buone performance per l'algoritmo di predizione ma deve essere applicata selezionando un valore n ragionevole: valori troppo alti, infatti, introducono un fattore di rumore nel modello a causa della presenza nel neighborhood di un insieme consistente di utenti non particolarmente significativi ma associati ad un valore di similarità intermedio; valori troppo bassi, invece, possono determinare predizioni non affidabili per utenti il cui neighborhood è costituito principalmente da utenti con similarità estremamente basse.

Una delle limitazioni più evidenti dei sistemi neighborhood based è la generazione di liste di raccomandazione costituite per la maggior parte da insiemi di item popolari e la perdita, quindi, del fattore di personalizzazione richiesto dall'utente. Tale comportamento è causato dalla tendenza degli utenti a visualizzare e votare positivamente l'insieme di item con più alto indice di popolarità: questo fenomeno da un lato causa la generazione di valori di similarità basati esclusivamente su interessi generalmente diffusi e

non su reali affinità tra utenti e dall'altro determina l'assegnazione di pesi elevati agli item *"popolari"* che consentono loro di essere inseriti con alta probabilità in numerose liste di raccomandazione. Per poter aumentare, quindi, la diversificazione delle raccomandazione e cercare di rispecchiare realmente i gusti dei singoli utenti è stato introdotto in alcuni Sistemi di Raccomandazione il concetto di *"k-furthest neighbor"* [27]. L'idea alla base di tale approccio è quella di utilizzare una misura di dissimilarità tra utenti che permetta di identificare l'insieme di k utenti più dissimili rispetto all'utente attivo; una volta individuati tali utenti, l'algoritmo identifica l'insieme di item con voti più bassi e li raccomanda all'utente attivo. Tale approccio, sebbene determini un peggioramento delle performance degli algoritmi di raccomandazione in termini di precision e recall, risulta efficiente nell'identificare insiemi di item di interesse non individuabili mediante i classici algoritmi di nearest neighbors.

1.2.2 Model Based

I Sistemi di Raccomandazione di tipo *Model Based* [1] [17] [18] [19], a differenza dei sistemi di tipo memory based, utilizzano l'insieme di voti espressi dagli utenti per costruire un modello statistico di preferenze su cui generare le predizioni. La particolarità di tali sistemi è la suddivisione del processo di predizione/raccomandazione in due fasi distinte:

- **Fase offline di generazione del modello:** in questa fase l'insieme dei voti presenti nel sistema viene utilizzato per costruire il modello statistico utilizzato come base per le predizioni; tale fase richiede un tempo di computazione elevato che ne impone l'esecuzione offline.

- **Fase online di predizione:** il modello costruito nella fase precedente viene utilizzato per generare la predizione per l'utente attivo.

Il modello di predizione generato dall'insieme dei voti espressi dagli utenti può essere costruito applicando differenti tecniche, sia di tipo probabilistico che di tipo non probabilistico [17].

Modelli di clustering: l'insieme degli utenti del sistema viene suddiviso in differenti classi, ciascuna rappresentante una tipologia di interesse in comune tra gli utenti che ne fanno parte; all'interno della medesima classe le preferenze espresse sui vari item sono indipendenti. Il modello di probabilità adottato per la distribuzione degli utenti nelle differenti classi in base all'insieme di voti espressi è un classificatore bayesiano nella sua formulazione naïve [28]:

$$Pr(C = c, v_1, ..., v_n) = Pr(C = c) \prod_{i=1}^{n} Pr(v_i \,|C = c\,) \tag{1.18}$$

I voti contenuti nel database del sistema vengono utilizzati per determinare il valore dei parametri del modello, come l'insieme di probabilità di appartenenza ad una determinata classe o l'insieme di probabilità condizionate.

Reti Bayesiane: ciascun item presente nel sistema viene rappresentato da un nodo all'interno di una rete bayesiana. Lo stato di ciascun nodo rappresenta l'insieme dei possibili voti assegnabili all'item; in caso di dati insufficienti, un nodo item può essere associato ad uno stato di tipo *"no vote"*. Ogni nodo della rete è associato ad un insieme di nodi padre che rappresentano i migliori predittori per il suo voto; le probabilità condizionate per tale nodo vengono rappresentate mediante un albero di decisione [5].

Modelli a fattori latenti e tecniche di fattorizzazione delle matrici: tali modelli di sistema si basano sull'idea di poter rappresentare gli insiemi di utenti e item e la loro similarità attraverso un set di caratteristiche sottostanti dei dati non note a priori [29] [30] [31] [32] [33] [34]. In particolare gli utenti e gli item del sistema vengono associati a due vettori di caratteristiche w_u,$h_i \in \mathbb{R}^k$ il cui valore viene determinato in modo che il loro prodotto interno $w_u^T h_i$ approssimi la preferenza nota $r_{u,i}$. Le tecniche di fattorizzazione delle matrici richiedono un elevato sforzo computazionale che può essere in parte mitigato attraverso strategie di calcolo incrementale dello spazio dei fattori latenti.

1.2.3 Vantaggi e limitazioni

I sistemi di tipo Collaborative Filtering [18], a differenza dei sistemi di tipo Content Based, sono sistemi di tipo flessibile che trovano applicazione in numerosi scenari; l'utilizzo di funzioni di predizione non legate ai metadati testuali associati agli item ma solo ai voti espressi dagli utenti, infatti, permette l'introduzione di un livello di raccomandazione di tipo collaborativo al di sopra di qualsiasi tipologia di sistema preesistente. La scelta di implementare un Sistema di Raccomandazione è determinata non solo dall'esigenza di fornire esplicitamente insiemi di raccomandazioni personalizzate agli utenti o di adattare il contenuto dei siti alle loro abitudini di navigazione ma anche dall'esigenza di fornire un sottoinsieme di servizi più complesso. Attraverso un Sistema di Raccomandazione, infatti, gli utenti possono individuare un set di item nuovi associati alle opinioni espresse dagli altri utenti del sistema oppure possono identificare agevolmente l'insieme di utenti caratterizzati da interessi simili ai propri e associarsi, quindi, in gruppi di discussione omogenei; inoltre i Sistemi di Raccomandazione di tipo

Collaborative Filtering riescono ad identificare in maniera efficiente l'insieme di item di interesse per un intero gruppo di utenti o adatti ad un particolare obiettivo di ricerca.

L'efficacia delle raccomandazioni generate dai sistemi di tipo Collaborative Filtering dipende da differenti fattori, quali la presenza di un numero di voti per item sufficiente, la presenza di un insieme di utenti adeguato per stabilire relazioni di similarità, la possibilità di definire per ciascun utente un profilo degli argomenti di interesse preciso. I sistemi reali possono sperimentare nel tempo la mancanza di uno o più di questi fattori e sono quindi soggetti a differenti problematiche [18].

New User Problem: i sistemi di Collaborative Filtering generano l'insieme di predizioni/raccomandazioni utilizzando un modello di preferenze costruito attraverso i voti espressi dagli utenti. Un nuovo utente che entra nel sistema deve, quindi, esprimere un insieme consistente di voti prima di poter ricevere delle raccomandazioni affidabili. Una delle tecniche sviluppate per superare tale limitazione consiste nell'invitare i nuovi utenti ad esprimere voti su un set di item predefiniti in modo da utilizzare tali voti per costruire una base di conoscenza iniziale delle preferenze degli utenti. Altre tecniche prevedono l'utilizzo di approcci ibridi che sfruttano algoritmi di tipo Content Based per colmare le lacune presenti nel sistema in mancanza di un insieme di voti consistenti; in particolare gli approcci ibridi cercano di combinare i voti con i metadati associati agli item votati in modo da identificare in maniera affidabile le preferenze dell'utente.

New Item Problem: i sistemi di tipo Collaborative Filtering si basano su algoritmi che selezionano l'insieme di item da raccomandare tra quelli votati dall'insieme di utenti con similarità maggiore rispetto

all'utente attivo; gli item con un numero di voti elevato, quindi, hanno maggiore probabilità di essere inseriti nelle liste di raccomandazione. I sistemi reali, tuttavia, sono sistemi dinamici in cui l'insieme degli item cresce all'aumentare del tempo; i nuovi item inseriti nel sistema, anche se di interesse per un set di utenti, vengono esclusi dalle liste di raccomandazione e penalizzati fino a che non raggiungono un numero sufficiente di voti.

Sparsity: l'elevata dimensione dell'insieme degli utenti e degli item dei Sistemi di Raccomandazione rispetto al numero di voti a disposizione determina la costruzione di matrici utenti/item altamente sparse [35]. Tale sparsità può causare diverse problematiche all'interno del sistema:

- **Reduced coverage:** l'insieme dei voti ridotto rispetto al numero di item presenti nel sistema causa l'impossibilità di generare predizioni per un set esteso di item, riducendo la copertura dell'algoritmo di raccomandazione.
- **Neighbor transitivity:** nei database sparsi può risultare difficile identificare relazioni tra utenti realmente simili se questi non hanno votato un sottoinsieme comune di item.

Gli approcci proposti per superare tali problematiche sono legati principalmente a tecniche di riduzione della dimensionalità delle matrici, che tentano di eliminare dal modello di sistema utenti o item non significativi in modo da applicare gli algoritmi di predizione su matrici di rango inferiore.

Scalabilità: i Sistemi di Raccomandazione Collaborative Filtering possono soffrire di evidenti problemi di scalabilità quando l'insieme degli item e

degli utenti cresce esponenzialmente [36]. Le tecniche di fattorizzazione delle matrici vengono applicate in tali situazioni allo scopo di ridurre il costo computazionale della fase di analisi della matrice di similarità e tipicamente vengono eseguite in forma incrementale in modo da ridurre i costi di ricostruzione della matrice ad ogni nuovo ingresso nel sistema di utenti o item. Anche i modelli di sistema basati su clusterizzazione degli utenti o degli item riescono a limitare il costo computazionale degli algoritmi limitando lo spazio di ricerca all'interno di insiemi di utenti caratterizzati da dimensioni ridotte ed alto livello di similarità interna.

Sinonymy: nei sistemi reali alcuni item possono essere associati a nomi differenti e quindi gestiti come entità separate dagli algoritmi di raccomandazione, riducendone le performance. Una soluzione a tale problematica consiste nell'utilizzare tecniche di espansione dei termini che permettono di aggiungere un insieme di keyword significative agli item in modo da migliorarne il riconoscimento automatico; diversamente, le tecniche di tipo SVD costruiscono uno spazio semantico in cui posizionare in maniera attigua item altamente correlati.

Gray Sheep: nei Sistemi di Raccomandazione esistono gruppi di utenti le cui opinioni non possono essere considerate altamente simili o altamente dissimili dalle opinioni di un qualsiasi altro set di utenti del sistema. In queste situazioni gli algoritmi di Collaborative Filtering non riescono a produrre raccomandazioni affidabili e perdono di utilità. Una delle soluzioni più adottate consiste nell'implementare un approccio di raccomandazione ibrido che combina i risultati ottenuti da algoritmi di

tipo Collaborative Filtering con i risultati ottenuti da algoritmi di tipo Content Based. In particolare viene determinato per ciascun utente un peso da applicare ai differenti risultati di predizione in modo da generare una media pesata.

Shilling Attacks: uno dei problemi più evidenti nei sistemi di e-commerce che utilizzano algoritmi di raccomandazione è il tentativo da parte di utenti maliziosi di forzare il sistema a raccomandare uno specifico set di item rispetto ai restanti item del sistema [2]. Tale attacco viene eseguito principalmente da utenti che hanno interesse a far crescere la popolarità dei propri prodotti, inserendo un insieme di voti ed opinioni positive fittizie nel sistema che aumentano la loro probabilità di essere selezionati dagli algoritmi di raccomandazione.

1.3 Modelli Ibridi

Negli ultimi anni numerosi sono stati i tentativi di migliorare le performance delle differenti tipologie di Sistema di Raccomandazione; una delle strategie più interessanti e di maggior successo è rappresentata dalla costruzione di sistemi di tipo *ibrido* che cercano di combinare gli algoritmi di tipo Collaborative Filtering con gli algoritmi di tipo Content Based, in modo da limitare gli svantaggi caratteristici di ciascuna delle due tipologie.

In generale è possibile identificare quattro differenti strategie di ibridazione :

Utilizzo separato di algoritmi di tipo Collaborative Filtering e Content Based e combinazione dei risultati: gli algoritmi di tipo Collaborative Filtering e di tipo Content Based vengono applicati al medesimo sistema separatamente; le predizioni generate

dalle due tipologie, quindi, vengono combinate assieme per generare una predizione finale. Tipicamente è possibile combinare i risultati dei diversi algoritmi attraverso una combinazione lineare o una media pesata delle predizioni generate oppure costruendo una lista di raccomandazione ottimale ottenuta selezionando i migliori risultati dei due approcci [37]. Più nello specifico è possibile classificare le differenti strategie adottate per combinare i vari Sistemi di Raccomandazione [38]:

- Ibridazione pesata: le tecniche di raccomandazione disponibili vengono applicate al modello di sistema; il risultato finale è una media pesata dei singoli risultati ottenuti.
- Ibridazione switching: vengono utilizzati alcuni criteri dedotti dal sistema per scegliere la tipologia di Sistema di Raccomandazione da adottare; tale scelta non è definitiva ma può cambiare a seguito dell'evoluzione del sistema stesso.
- Ibridazione mista: la raccomandazione finale è una collezione dei migliori risultati ottenuti dai vari algoritmi di raccomandazione.
- Ibridazione a combinazione di feature: il modello di sistema viene integrato mediante l'applicazione di dati ottenuti attraverso algoritmi di tipo Collaborative Filtering; in seguito viene applicato un approccio di tipo Content Based per generare la raccomandazione.
- Ibridazione Cascade: gli algoritmi di raccomandazione vengono applicati in sequenza; i risultati ottenuti mediante un algoritmo vengono rifiniti dall'algoritmo successivo.

- Ibridazione feature augmentation: uno degli algoritmi di raccomandazione viene utilizzato per generare una serie di feature aggiuntive che vengono utilizzate all'interno dei restanti algoritmi di raccomandazione insieme ai dati originali.
- Ibridazione Meta-Level: uno degli algoritmi di raccomandazione viene utilizzato per costruire un modello di sistema che operi come base di conoscenza per i restanti algoritmi di raccomandazione.

Utilizzo di caratteristiche Content Based negli algoritmi di tipo Collaborative Filtering: gli algoritmi di tipo Collaborative Filtering basano il proprio funzionamento sull'individuazione di relazioni di similarità tra utenti e tra item; in molti casi la sparsità dei voti presenti nel sistema influenza tale processo e non permette di cogliere relazioni esistenti tra specifici insiemi di utenti. Diversamente gli algoritmi di tipo Content Based riescono a stabilire relazioni di similarità mediante la semplice analisi dei profili degli utenti e degli item. Basandosi su queste osservazioni sono stati sviluppati sistemi ibridi di tipo Collaborative Filtering che utilizzano come coefficienti di similarità i valori ottenuti attraverso il confronto tra i profili degli utenti e degli item mediante l'applicazione di tecniche di tipo Content Based. Le similarità così calcolate vengono utilizzate all'interno degli algoritmi Collaborative Filtering per generare l'insieme di raccomandazioni finali [39]. Una differente strategia di ibridazione prevede l'utilizzo di *filterbot* [5], agenti automatici che simulano le azioni di un utente del sistema e i cui voti vengono generati mediante l'applicazione di algoritmi di tipo Content Based; gli utenti reali del sistema associati ad un valore di similarità rilevante con i filterbot riescono a ricevere un insieme di raccomandazioni con elevato grado di affidabilità.

I sistemi di Collaborative Filtering di tipo *Content Boosted* [5], invece, applicano classificatori di tipo Bayesiano per costruire una pseudo matrice di voti; tale matrice contiene i voti reali del sistema ed integra le entry vuote con un insieme di voti fittizi calcolati applicando il classificatore bayesiano. Le predizioni vengono, quindi, generate attraverso l'applicazione di algoritmi basati sul calcolo del coefficiente di correlazione di Pearson.

Utilizzo di caratteristiche Collaborative Filtering negli algoritmi di tipo Content Based: gli approcci più comuni prevedono l'applicazione di tecniche di riduzione della dimensionalità della matrice dei voti, come l'utilizzo di Latent Semantic Indexing, per creare una vista di tipo collaborativa dei profili degli utenti, rappresentati mediante vettori di termini.

Costruzione di un modello unificato che incorpora caratteristiche Collaborative Filtering e Content Based: tale approccio prevede la costruzione di un modello bayesiano di preferenze che integra in modo statistico differenti fonti di informazione, quali i voti, i profili costruiti su utenti e item attraverso l'analisi di keyword, le opinioni espresse sull'insieme di item. I dati ottenuti vengono utilizzati all'interno di Catene di Markov per fare inferenza sul sistema e generare le predizioni. Tali approcci sono limitati dal costo computazionale richiesto per la generazione della base di conoscenza su cui effettuare inferenza; inoltre non riescono ad essere applicati a sistemi nei quali è impossibile rappresentare la base di conoscenza mediante strutture analizzabili da agenti automatici.

In alcuni casi le tecniche di ibridazione prevedono la combinazione di

differenti caratteristiche associate alla stessa tipologia di Sistema di Raccomandazione, come per esempio la creazione di sistemi che combinano approcci di tipo memory based con approcci di tipo model based [40] [18]; in particolare è possibile identificare due tipologie specifiche di ibridazione:

Probabilistic Memory Based Collaborative Filtering: tale strategia di ibridazione si basa sulla costruzione di un modello di sistema mediante l'utilizzo delle informazioni associate ai voti presenti nella matrice utenti/item. In particolare ciascun utente viene associato ad un profilo definito dall'insieme dei voti espressi, che viene utilizzato successivamente all'interno di metodi di tipo probabilistico per la generazione dell'insieme di raccomandazioni. In mancanza di sufficienti dati per la creazione dei profili l'algoritmo utilizza tecniche di interrogazione attiva per ottenere un insieme di informazioni addizionali. Per poter ridurre in maniera efficiente il costo computazionale mantenendo elevato il livello di affidabilità del Sistema di Raccomandazione, l'algoritmo probabilistico utilizza come base del modello un sottoinsieme ridotto dei profili degli utenti del sistema.

Personality Diagnosis: i sistemi di questo tipo cercano di determinare la "personalità" di ciascun utente attraverso l'analisi dell'insieme dei voti espressi [41] [5]. L'idea alla base di questi sistemi è che ogni voto espresso dagli utenti sia affetto da una forma di rumore gaussiano indotto da diversi fattori come per esempio il contesto degli item votati nella stessa sessione; l'obiettivo principale dell'algoritmo, perciò, consiste nel rimuovere tale fonte di rumore e basare l'insieme di predizioni sul voto "reale" degli item. L'algoritmo di predizione dei voti cerca di stabilire inizialmente la probabilità per gli utenti di condividere la stessa personalità, definita come l'insieme dei voti reali espressi; in

seguito, i valori di probabilità ottenuti vengono utilizzati per stabilire una distribuzione di probabilità di voto per l'utente attivo.

Capitolo 2

Time-Aware Recommender Systems

La maggior parte dei sistemi di raccomandazione sviluppati negli anni sono sistemi caratterizzati da algoritmi di predizione di tipo *statico* applicati ad un modello di sistema altamente *dinamico*: gli insiemi di utenti ed item, infatti, mutano al variare del tempo, arricchendosi continuamente di nuovi elementi e perdendone di vecchi.

La dinamicità di tali sistemi, tuttavia, non si limita alle variazioni di composizione degli insiemi del modello di computazione ma può essere rappresentata da diversi fattori, come l'aggiornamento nel tempo dei profili associati agli item e agli utenti attraverso l'inserimento di nuove informazioni, l'aggiunta continua, da parte degli utenti, di nuovi voti, il cambiamento progressivo degli interessi degli utenti, la minore rilevanza di alcuni oggetti al passare del tempo.

In particolare l'aspetto che maggiormente caratterizza Sistemi di Raccomandazione di tipo dinamico e che risulta più difficile da modellare è la variazione continua degli interessi e delle preferenze dell'utente nel tempo.

Tale aspetto, infatti, è legato a molteplici fattori, tutti egualmente importanti nella costruzione di un modello di comportamento per l'utente. Tipicamente l'insieme dei voti generati in un preciso istante da un utente dipende dal particolare *momento temporale* in cui ci si trova: oggetti visualizzati e votati nel periodo estivo, per esempio, differiscono da quelli visualizzati durante il periodo invernale e così gli item acquistati per una specifica festività possono non essere di interesse durante il resto dell'anno. Anche informazioni strettamente personali come la composizione del nucleo familiare di un utente possono incidere sull'insieme di item votati: utenti con bambini, infatti, effettuano acquisti differenti da utenti senza figli e la variazione del nucleo familiare nel tempo spesso si riflette in una variazione del pattern di acquisti effettuati. L'aggiornamento continuo del catalogo di item presenti nel sistema attraverso l'introduzione di nuovi elementi, inoltre, contribuisce al progressivo cambiamento di una parte degli interessi e delle preferenze dell'utente, che tipicamente riflettono i trend del momento.

Gli algoritmi di raccomandazione nella loro formulazione classica associano a ciascuna preferenza espressa dagli utenti il medesimo peso, indipendentemente dal tempo o dal contesto in cui è stata generata; tale comportamento di fatto elimina la possibilità di individuare la dinamicità che caratterizza l'interazione di ciascun utente con il sistema nel tempo, equiparando ogni scelta passata con le possibili scelte future.

Per questo motivo negli ultimi anni l'attenzione dei ricercatori nel campo dei Recommender System si è concentrata sullo sviluppo di algoritmi in grado di modellare con precisione l'evoluzione dei gusti degli utenti attraverso l'analisi dei dati a disposizione, come voti e profili. L'aspetto innovativo di tale tipologia di algoritmi risiede nell'introduzione del *fattore temporale* negli algoritmi di calcolo delle similarità e di predizione ; ciascuna preferenza

espressa, infatti, non è più considerata come un' entità immutabile ma diventa un dato che dipende dall'istante temporale in cui è stato generato e che diminuisce il proprio peso in maniera proporzionale allo scorrere del tempo [42].

Il concetto di *tempo* introdotto nei Sistemi di Raccomandazione classici è un concetto complesso che può essere modellato in differenti forme dipendentemente dal tipo di caratteristica dei dati che si vuole evidenziare [43] [44]. In particolare le preferenze degli utenti possono essere raggruppate in due *macro-insiemi*, uno rappresentante l'insieme di *interessi a lungo termine* legati alla personalità stessa dell'utente e l'altro rappresentante l'insieme di *interessi a breve termine* legati ad eventi o circostanze temporanee; gli algoritmi di raccomandazione, perciò, devono essere in grado di bilanciare efficacemente il contributo di ciascun dato temporale, in modo da scartare gli elementi di scarso impatto sulle scelte future ed identificare il trend di comportamento corrente per l'utente.

La maggior parte dei Sistemi di Raccomandazione *time-aware* basa i propri algoritmi sull'utilizzo di una nozione di *tempo globale* condivisa tra tutti gli utenti; in quest'ottica, attraverso l'analisi dell'insieme dei voti presenti nel sistema, è possibile costruire un modello di evoluzione temporale universale, in cui le preferenze espresse dagli utenti nel medesimo istante per item differenti possono essere interpretate come segni di una correlazione esistente tra i vari utenti. Nei sistemi reali, tuttavia, le scelte di voto sono dinamiche locali, legate in maniera specifica al singolo individuo che le ha effettuate e agli eventi esterni che le hanno generate; alcuni Sistemi di Raccomandazione, perciò, modellano il tempo come una *dimensione locale*, in modo da identificare le correlazioni esistenti tra le scelte effettuate dal singolo utente in periodi differenti.

La nozione di tempo utilizzata nei sistemi di tipo time-aware può rappresentare differenti caratteristiche degli item del sistema [45]:

Tempo di lancio: rappresenta l'istante temporale in cui l'item viene inserito nel catalogo del sistema e ne indica l'età. Tale tipologia di tempo viene utilizzata negli algoritmi di peso per diminuire l'importanza di item vecchi.

Tempo di acquisto/voto: rappresenta l'istante temporale in cui l'utente vota/acquista lo specifico item e indica l'età della preferenza dell'utente. L'insieme di preferenze più recenti tipicamente ha peso maggiore nella modellazione delle preferenze future.

Differenza temporale tra tempo di acquisto e tempo di lancio: rappresenta l'intervallo temporale entro il quale uno specifico item può essere di interesse per l'utente. Una grande differenza temporale rappresenta item vecchi ma potenzialmente di interesse per gli utenti per un lungo periodo futuro.

I Sistemi di Raccomandazione time-aware, così come i sistemi time-free, possono utilizzare algoritmi di tipo *memory-based* o algoritmi di tipo *model-based* per generare l'insieme di predizioni; nello specifico è possibile organizzare l'insieme di algoritmi sviluppati in sotto-categorie, ciascuna relativa alla modalità con cui il fattore temporale viene modellato ed utilizzato nel sistema.

2.1 Memory Based

2.1.1 Algoritmi con schemi di peso

Una delle tecniche più semplici per introdurre la nozione di tempo in algoritmi di raccomandazione preesistenti consiste nell'associare a ciascuna preferenza espressa dall'utente un *fattore di peso* correlato all'istante temporale di generazione del voto.

Tali algoritmi si basano principalmente sull'idea che l'insieme di voti recenti di un utente sia un modello più preciso dei suoi attuali interessi rispetto all'insieme di voti espressi in istanti temporali lontani; viene, perciò, utilizzata una funzione di peso il cui valore decresce al passare del tempo, penalizzando di fatto l'insieme di voti *"passati"* di ciascun utente.

Ogni algoritmo può adottare una differente funzione di peso che determina la velocità con cui l'importanza dei singoli voti decade; una strategia tipica utilizzata per ridurre il fattore di rumore introdotto negli algoritmi di predizione da voti associati a pesi trascurabili consiste nel fissare una *soglia di peso* al di sotto della quale le preferenze del singolo utente vengono scartate dalla computazione della predizione.

Nel 2005 Ding e Li [46] propongo l'estensione di un algoritmo di Collaborative Filtering di tipo Item-Based attraverso l'introduzione di un fattore di peso decrescente. In particolare il contributo dei due ricercatori si focalizza sulla fase di predizione dei voti, che viene modificata in modo da integrare una funzione temporale che associ a ciascun item un peso relativo al *timestamp* di generazione della predizione:

$$r_{xj} = \frac{\sum_{c=1}^{k} r_{xc} \cdot sim\,(j,c) \cdot f\,(t_{xc})}{\sum_{c=1}^{k} sim\,(j,c) \cdot f\,(t_{xc})} \tag{2.1}$$

dove r_{xj} è il voto previsto per l'utente x sull'item j, c è l'insieme di item *nearest neighbors* dell'item j e t_{xc} è il timestamp del voto assegnato dall'utente x sull'item c.

La funzione temporale di peso rappresenta l'elemento chiave del nuovo algoritmo, in quanto determina il modo in cui il tempo influisce sul contributo di ciascun voto degli utenti alla predizione finale.

Una delle funzioni tipicamente scelte negli algoritmi di tipo time-aware ed adottata in questo particolare caso è la *funzione di decadimento di tipo esponenziale*, che riduce in modo uniforme il peso di ogni preferenza al passare del tempo t.

$$f(t) = e^{-\lambda \cdot t} \tag{2.2}$$

La funzione di decadimento è caratterizzata da un parametro T_0 che concettualmente rappresenta la velocità di decadimento dei pesi degli item; formalmente, tale parametro può essere interpretato come l'intervallo temporale entro cui un qualsiasi peso viene ridotto di 1/2.

$$F(T_0) = \left(\frac{1}{2}\right) f(0) \tag{2.3}$$

Il rate di decadimento viene quindi definito come:

$$\lambda = \frac{1}{T_0} \tag{2.4}$$

L'utilizzo di un parametro per la definizione del rate di decadimento degli item permette di modulare in maniera dinamica il comportamento dell'algoritmo di predizione rispetto al modello di sistema cui è applicato: *valori bassi* di T_0 , infatti, si riflettono in una velocità di decadimento elevata, sinonimo di rapidi cambiamenti nei gusti degli utenti, mentre *valori alti* di T_0 vengono utilizzati per rappresentare sistemi caratterizzati da interessi stabili nel tempo.

Nel 2006 Ding e Li [47] sviluppano un nuovo algoritmo di Collaborative Filtering *recency based*, basato sul concetto di *"vicinanza temporale"* tra voti. L'idea alla base dell'algoritmo è quella di utilizzare, nella fase di predizione, una funzione di peso che assegni a ciascun voto un fattore inversamente proporzionale all'errore di predizione sulle preferenze future dell'utente, in modo da catturare la variabilità degli interessi dell'utente nel tempo.

$$r_{xj} = \frac{\sum_{c=1}^{k} r_{xc} \cdot sim\,(j,c) \cdot W_c}{\sum_{c=1}^{k} sim\,(j,c) \cdot W_c} \tag{2.5}$$

L'errore di predizione per un item target è teoricamente definito come la differenza tra il voto previsto per l'item e il voto reale assegnato dall'utente; tale valore non può essere utilizzato nella sua formulazione standard nella fase di predizione, non essendo noto il voto reale che l'utente assegnerà in futuro all'item stesso. Per poter superare tale limitazione l'algoritmo approssima l'errore di predizione utilizzando la *deviazione standard* tra il voto assegnato ad un item (r_i) e il voto più recente presente nel neighborhood dell'item stesso (r_r):

$$W_i = \left(1 - \frac{|r_i - r_r|}{|R|}\right)^{\alpha} \tag{2.6}$$

La funzione di peso introdotta nell'algoritmo non utilizza esplicitamente la nozione di tempo ma ne simula il comportamento ipotizzando che l'insieme delle preferenze più recenti presenti nel neighborhood di un item siano un' accurata rappresentazione dei trend e degli interessi futuri degli utenti.

Nel 2008 Lee e Park [45] [48] sviluppano un algoritmo Collaborative Filtering basato sull'utilizzo delle preferenze espresse implicitamente dagli utenti; in particolare i due ricercatori elaborano un algoritmo per sistemi di tipo *e-commerce*, nei quali gli utenti tipicamente interagiscono effettuando acquisti piuttosto che votando gli item visualizzati. Per poter modellare correttamente l'insieme di interessi di ciascun utente l'algoritmo costruisce

una *matrice di pseudo-voti*, in cui l'acquisto da parte di un utente di un item viene mappato nell'inserimento di un valore di peso nella specifica entry. I valori da inserire nella matrice vengono ottenuti attraverso l'analisi di due differenti *contributi temporali*, il tempo di inserimento di un item nel sistema e il tempo di acquisto dell'item.

$$w\left(p_i, l_j\right) = \text{peso di un item con tempo di lancio } l_j \text{ acquistato al tempo } p_i \tag{2.7}$$

Basandosi sull'osservazione che gli acquisti più recenti riflettono meglio i trend attuali dell'utente e che gli item inseriti da poco nel sistema risultano più interessanti, la funzione di peso associa valori maggiori agli item con tempo di acquisto e di lancio più recenti. In modo da semplificare l'assegnazione di peso a ciascun item, l'algoritmo non utilizza direttamente il valore del timestamp associato al tempo di acquisto e al tempo di lancio ma categorizza queste due componenti temporali in tre insiemi, associati a valori vecchi, valori intermedi e valori recenti.

	old purchase group	**middle purchase group**	**recent purchase group**
old launch group	0.7	1.7	2.7
middle launch group	1	2	3
recent launch group	1.3	2.3	3.3

Nel 2013 Li et al [49] propongono un algoritmo di Collaborative Filtering di tipo Item-based che utilizza una funzione di peso modellata sulla curva di *forgetting* della memoria umana. L'analisi dei processi di apprendimento nell'uomo, infatti, mostra come il ricordo di un insieme di informazioni decada in maniera graduale nel tempo a meno che gli stessi contenuti non vengano ripetuti in modo da rafforzarne l'impronta nella memoria.

Basandosi su queste osservazioni, i ricercatori hanno introdotto in un algoritmo classico di Collaborative Filtering una funzione di peso per gli item che simula il comportamento della memoria umana. In particolare l'insieme dei voti espressi da un utente viene suddiviso in diversi *cluster*, ciascuno associato ad un differente *finestra temporale* T_i; il peso iniziale degli item presenti nel primo cluster viene impostato ad un valore standard w_0.

La funzione di peso utilizzata all'interno dell'algoritmo cerca di identificare le variazioni di interessi degli utenti applicando due differenti funzioni di decadimento ai voti:

- una funzione di decadimento lineare, che simula interessi degli utenti stabili nel tempo, viene scelta quando la similarità tra due cluster di item contigui nel tempo è superiore ad un certo valore di soglia.

- una funzione di decadimento esponenziale, che simula il cambiamento degli interessi degli utenti rispetto alle preferenze precedenti, viene scelta quando la similarità tra due cluster di item contigui è minore del valore di soglia.

$$w_{xj,n+1} = \begin{cases} w_{xj,n} \cdot e^{-\lambda \cdot \Delta t} & \text{se } max\left(sim\left(I_{x,n}, I_{x,n+1}\right)\right) < \delta \\ -kt + (w_{xj,n} + \mu) & \text{altimenti} \end{cases} \tag{2.8}$$

dove $w_{xj,n}$ è il coefficiente di peso per l'item j nella collezione dell'utente x al tempo n, $I_{x,n}$ è l'insieme di item votati dall'utente x al tempo n e λ,μ e k sono costanti caratteristiche.

L'utilizzo di un valore di threshold per il peso degli item consente l'eliminazione dal database del sistema degli oggetti il cui peso non è più rappresentativo delle attuali scelte dell'utente.

Nel 2010 Zheng e Li [50]propongono la costruzione di un Sistema di Raccomandazione di tipo Collaborative Filtering che integra in un unico modello di predizione le informazioni associate ai tag degli item e le informazioni associate agli istanti temporali di generazione dei tag. In particolare l'algoritmo adotta una classica funzione di decadimento esponenziale per diminuire il peso dei singoli item nel tempo:

$$w_{time}\left(u,r\right) = e^{-\frac{\ln 2 \cdot time(u,r)}{hl_u}} \tag{2.9}$$

dove $time\left(u,r\right)$ è il timestamp associato al voto r espresso dall'utente u.

La novità introdotta dall'algoritmo consiste nel costruire per ciascun utente e per ciascuna risorsa un insieme di voti modificati, ottenuti mediante la combinazione lineare dei tag e delle informazioni temporali associate a ciascun item.

$$M_{u,r} = \lambda w_{tag}\left(u,r\right) + \left(1-\lambda\right) w_{time}\left(u,r\right) \tag{2.10}$$

I classici metodi di calcolo delle similarità tra utenti e di generazione delle predizioni vengono, quindi, modificati in modo da utilizzare il nuovo insieme di voti costruito dall'algoritmo.

Nel 2010 Liu et al. [51] propongono un algoritmo di Collaborative Filtering di tipo Item-Based basato sulla nozione di *rilevanza temporale* di un item.

Ciascuna preferenza espressa dagli utenti del Sistema di Raccomandazione viene associata ad un valore di peso che rappresenta la rilevanza dello specifico voto nell'ambito dell'algoritmo di raccomandazione; il peso dei voti, così come in altri algoritmi analoghi, viene determinato attraverso l'applicazione di una funzione di decadimento esponenziale, la cui velocità dipende da diversi fattori come l'istante di generazione della predizione t, l'istante di generazione del voto t_{ui} e un parametro modulabile α:

$$f_{ui}^{\alpha}(t) = e^{-\alpha(t-t_{ui})} \tag{2.11}$$

L'algoritmo descritto in tale paper introduce due importanti novità nell'ambito degli algoritmi di raccomandazione basati su *funzioni di peso*:

- la *rilevanza temporale* dei voti viene utilizzata anche nella fase di calcolo delle similarità tra item, modellando il fatto che le similarità tra gli item dipendono maggiormente dai trend di voto attuali rispetto ai trend passati.

- ad ogni istante temporale il calcolo del neighborhood di un item e il calcolo dei valori di similarità viene effettuato in maniera incrementale utilizzando i valori ottenuti nell'iterazione precedente; in questo modo l'algoritmo non è costretto a ricostruire il modello di sistema ad ogni esecuzione ma può limitarsi ad aggiornare i valori calcolati in precedenza, gestendo quindi uno sforzo computazionale minore rispetto ad un algoritmo non incrementale.

2.1.2 Algoritmi con knn dipendente dal tempo

L'introduzione del fattore temporale nei Sistemi di Raccomandazione permette di costruire modelli di tipo dinamico, in cui i dati e le relazioni

sottostanti variano a causa dell'introduzione di nuovi item, utenti o voti nel tempo. Tale dinamicità si manifesta nei sistemi di tipo *neighborhood based* nell' aggiornamento progressivo dei profili associati ad utenti e item e di conseguenza nel cambiamento nel tempo della struttura del neighborhood stesso.

Nel 2008 Lathia et al [52] analizzano in che modo le similarità tra coppie di utenti di un Sistema di Raccomandazione evolvono con l'introduzione di nuovi item e voti nel sistema. In particolare nella prima parte del loro lavoro si focalizzano sull'analisi delle variazioni del coefficiente di similarità tra coppie di utenti nel tempo. Gli esperimenti condotti attraverso l'applicazione di differenti misure di similarità evidenziano una naturale tendenza del coefficiente associato ad ogni singolo utente a convergere ad un valore stabile, dopo una serie di oscillazioni iniziali dovute alla dimensione ridotta dei dati utilizzati come input per l'algoritmo. Tale fenomeno dipende principalmente dal raffinamento progressivo dei profili degli item e degli utenti che si ottiene attraverso l'introduzione continua di nuovi dati di interesse nel sistema.

Nella seconda parte del loro lavoro i ricercatori analizzano le variazioni nel tempo della struttura del neighborhood degli utenti attraverso una serie di esperimenti basati sull'esecuzione di aggiornamenti del sistema in specifici intervalli temporali; ad ogni aggiornamento viene, quindi, analizzata la composizione del neighborhood di ciascun utente in modo da individuare il set di utenti presenti per la prima volta e il set di utenti presenti già nelle iterazioni precedenti. I dati ottenuti mostrano come la variabilità della struttura del neighborhood diminuisca al passare del tempo, facendo convergere il set di utenti *vicini* ad un insieme temporalmente stabile.

Nel 2009 Lathia et al. [53] elaborano un sistema di tipo Collaborative

Filtering in cui l'algoritmo di predizione sottostante cambia al variare delle informazioni presenti nel sistema, in modo da garantire sempre le prestazioni migliori. L'approccio utilizzato prevede l'adozione di due differenti metodi adattivi, uno basato sulla variazione dell'algoritmo di Collaborative Filtering implementato nel sistema e l'altro basato sulla variazione dei parametri all'interno di un classico algoritmo di tipo *neighborhood based*.

Nella prima metodologia viene introdotto nel sistema un insieme P di differenti algoritmi di raccomandazione, tra cui algoritmi di tipo k *nearest neighbors* e algoritmi *bias-based*. Ad ogni intervallo temporale, corrispondente ad un aggiornamento dei dati presenti nel sistema, viene valutato per ciascun utente l'algoritmo che predice meglio le sue preferenze, basandosi sui valori di *RMSE* ottenuti e sul valore di errore e_i commesso:

$$\forall u : L_{u,t+1} = \underset{L_i \in P}{max} \left(e_i - RMSE_{t,P_i}\right) \tag{2.12}$$

Tale strategia di adattamento temporale permette di generare un insieme di predizioni caratterizzate da un elevato livello di accuratezza; il modello di sistema che ne risulta è un modello fortemente personalizzato, in cui l'algoritmo scelto non ottimizza la predizioni per un insieme esteso di utenti ma si adatta al profilo di ogni singolo individuo. Il costo computazionale di una scelta architetturale di questo tipo è elevato, dal momento ogni aggiornamento dei dati presenti nel sistema determina l'aggiornamento dei parametri di tutti gli algoritmi a disposizione.

Una possibile soluzione a tale problematica viene analizzata nella seconda metodologia di adattamento temporale proposta, in cui viene utilizzato un singolo algoritmo di predizione di tipo k *nearest neighbors* i cui parametri di lavoro variano al variare dell'insieme dei dati del sistema. L'algoritmo di predizione viene associato ad un set P di valori differenti per il parametro k, che rappresenta la dimensione del neighborhood di ciascun utente; gli

utenti che entrano nel sistema in uno specifico intervallo t vengono associati ad un elemento predefinito dell'insieme P. Ad ogni intervallo temporale l'algoritmo seleziona, per ciascun utente, il valore di k che minimizza l'errore di predizione all'interno dell'insieme P:

$$\forall u : k_{u,t+1} = \underset{k_i \in P}{max} \left(e_i - RMSE_{t,P_i}\right) \tag{2.13}$$

L'utilizzo di una strategia adattiva permette di evidenziare alcune caratteristiche dei sistemi di tipo neighborhood based che gli algoritmi classici non riescono a cogliere. In particolare è possibile notare come le differenze tra i profili degli utenti dello stesso Sistema di Raccomandazione non consentano all'algoritmo di convergere su un valore del parametro k univoco; i dati a disposizione, infatti, rilevano come gli utenti in realtà siano suddivisi in gruppi di similarità, ciascuno associato ad un valore di k ottimale differente. L'analisi dei valori k associati a ciascun gruppo di utenti nel tempo, inoltre, mostra la tendenza a convergere ad un valore finale stabile, sebbene una percentuale di utenti mantenga nel tempo un andamento oscillatorio di tale valore.

2.2 Model Based

2.2.1 Algoritmi di fattorizzazione temporali

I sistemi classici di tipo model based si basano su modelli di sistema statici, in cui la nozione di evoluzione temporale non è presente; negli ultimi anni, perciò, c'è stata la tendenza a modificare gli algoritmi preesistenti in modo da integrare il fattore tempo ed evidenziare le dinamiche temporali dei dati. Parte della ricerca in questo ambito si è concentrata nell'individuazione dell'insieme di effetti temporali che possono migliorare l'accuratezza delle

predizioni generate dagli algoritmi classici; tali effetti vengono, quindi, introdotti nei processi di modellazione dei fattori latenti, in modo da creare un modello del sistema completo.

Nel 2008 Bell e Koren [54] [55] [56] nell'ambito della competizione *Netflix* propongono l'estensione di un classico algoritmo *SVD* attraverso l'introduzione di specifici parametri temporali.

In particolare i due ricercatori identificano una serie di *effetti temporali* che influenzano da un lato le caratteristiche di utenti e item nel tempo e dall'altro la loro correlazione. Il primo effetto descritto è la variabilità temporale della popolarità di ciascun item, legata spesso ad eventi esterni che possono influenzare l'aumento o la diminuzione del suo valore. Il secondo effetto, invece, è rappresentato dalla tendenza di ciascun utente ad assegnare voti differenti allo stesso item in diversi istanti temporali; tale comportamento è legato alla variazione nel tempo di numerosi fattori, come la scala di voto adottata dagli utenti o l'insieme di item covotati nel medesimo istante. Un ulteriore effetto temporale presente nei sistemi di tipo time-aware è rappresentato dalla naturale evoluzione degli interessi degli utenti nel tempo, legata principalmente al cambiamento di percezione rispetto a determinate caratteristiche degli item votati.

Le modalità con cui i diversi effetti temporali vengono modellati rispetto al fattore tempo deve tener conto della velocità di variazione che li caratterizza: i cambiamenti legati agli item, per esempio, si manifestano in un lungo intervallo temporale mentre gli effetti legati agli utenti possono verificarsi su base giornaliera.

L'algoritmo di tipo SVD time-aware proposto integra ciascun effetto temporale nella funzione di predizione finale, in modo da catturare le

variazioni temporali di utenti e item:

$$\hat{r_{ui}} = \mu + b_u(t) + b_i(t) + q_i^T \left(p_u(t) + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j \right) \tag{2.14}$$

dove $b_i(t), b_u(t)$ e $p_u(t)$ sono rispettivamente il drift degli item, il drift degli utenti e il drift delle preferenze.

Nel 2009 Xiang e Yang [57]estendono l'algoritmo time-aware SVD proposto da Bell e Koren identificando una serie di effetti temporali aggiuntivi non presenti nell'algoritmo originale.

L'algoritmo proposto dai due ricercatori incorpora nei modelli di fattorizzazione delle matrici i quattro *effetti temporali* descritti nel lavoro di Bell e Koren:

Time Bias: variazione nel tempo delle abitudini e degli interessi della società.

User Bias Shifting: variazione nel tempo delle abitudini del singolo individuo.

Item Bias Shifting: variazione nel tempo della popolarità di un item.

User Preference Shifting: variazione nel tempo delle preferenze dell'utente.

Questi elementi vengono affiancati nell'algoritmo di modellazione del sistema da una serie di *effetti temporali secondari*:

User Loyalty: tale parametro misura il tempo di attività di un utente all'interno di un sistema e permette di discriminare tra utenti vecchi, le cui abitudini di voto sono tipicamente stabili nel tempo, e utenti nuovi, le cui preferenze sono difficili da predire.

User Activity: tale metrica misura l'attività di un utente all'interno del Sistema di Raccomandazione interpretata come il numero di voti espresso dall'utente su base giornaliera.

Item Popularity: tale metrica rappresenta la popolarità di ciascun item interpretata come il numero di voti ricevuto su base giornaliera.

Nel 2011 Karatzoglou [58] propone un'estensione per algoritmi di tipo SVD che include nei processi di fattorizzazione l'insieme degli ultimi N item visualizzati dall'utente. In particolare viene modificato lo spazio dei *fattori latenti* rappresentati le relazioni tra utenti ed item attraverso l'introduzione dei fattori associati agli ultimi item visualizzati; i modelli ottenuti con questo approccio possono affidarsi a due differenti strategie di integrazione dei nuovi parametri, la prima basata su un metodo di tipo moltiplicativo

$$F_{ij} = \left\langle U_{i*}, M^{t}_{j*}, M^{\prime t-1}_{k*}, ..., M^{\prime\prime t-N}_{l*} \right\rangle \tag{2.15}$$

e la seconda basata su un metodo di tipo additivo

$$F_{ij} = \left\langle U_{i*}, M^{t}_{j*} \right\rangle + \left\langle U_{i*}, M^{\prime t-1}_{k*} \right\rangle + ... + \left\langle U_{i*}, M^{\prime\prime t-N}_{l*} \right\rangle \tag{2.16}$$

dove i e j sono gli item per i quali si vuole una predizione, k è l'ultimo item visualizzato dall'utente e l è l'item *n-esimo* visualizzato; F_{ij} approssima il voto Y_{ij} , M^{t}_{j*} sono i fattori associati all'item da predire mentre $M^{\prime t}_{k*}, ..., M^{\prime\prime t-N}_{l*}$ sono i fattori associati agli ultimi N item visualizzati.

Il modello dei fattori latenti viene, quindi, costruito attraverso la minimizzazione di una funzione di perdita tra i valori osservati e i valori predetti.

$$L\left(F, Y\right) := \sum_{i,j} D_{ij} l\left(F_{ij}, Y_{ij}\right) \tag{2.17}$$

2.2.2 Algoritmi di clustering

L'analisi dei fattori temporali legati ai voti espressi dagli utenti permette in molti casi di identificare schemi di comportamento ripetitivi o insiemi di interessi costanti in specifici intervalli temporali; tale strutturazione dei Sistemi di Raccomandazione consente l'applicazione di strategie complesse di clustering, nelle quali il fattore temporale diventa la discriminante principale per l'aggregazione dei dati a disposizione.

Nel 2005 Min e Han [59] propongono un algoritmo di tipo Collaborative Filtering in grado di identificare all'interno di un sistema le variazioni nell'insieme di interessi degli utenti. L'algoritmo utilizza una fase di preprocessamento dei dati il cui scopo è quello di permettere il confronto tra serie di voti appartenenti a intervalli temporali diversi: l'insieme degli item del sistema viene inserito, quindi, in una struttura gerarchica ad albero le cui foglie rappresentano gli item stessi e i nodi non foglia rappresentano le categorie di appartenenza degli item.

Il grado di variabilità degli utenti nel tempo viene misurato attraverso l'applicazione di due differenti metodologie:

Cambiamento dei cluster: i voti associati alle categorie degli item vengono utilizzati per determinare in ogni istante temporale il cluster di appartenenza dell'utente; il cambiamento continuo di cluster nel tempo indica un elevato grado di variabilità negli interessi dell'utente mentre un insieme di cluster costante rappresenta utenti con interessi stabili nel tempo.

Autosimilarità: per ogni utente viene calcolato un valore di autosimilarità, che rappresenta l'andamento nel tempo dei suoi interessi, attraverso il confronto delle categorie di item votati in coppie di intervalli distinti.

Il coefficiente di autosimilarità può assumere un valore pari a 1 se gli interessi in due intervalli consecutivi sono rimasti costanti o un valore pari a -1 se gli interessi divergono completamente.

Il grado di variabilità viene utilizzato nella successiva fase dell'algoritmo per discriminare tra l'applicazione di una funzione di similarità tra utenti statica, in caso di variabilità degli interessi nulla, o dinamica, negli altri casi.

Nel 2009 Baltrunas e Amatriain [60] propongono una strategia di rappresentazione degli interessi degli utenti attraverso l'utilizzo di un metodo di clusterizzazione delle preferenze espresse. In particolare il profilo di ogni singolo utente viene suddiviso in una serie di *microprofili*, ciascuno contenente i voti espressi dall'utente in un particolare intervallo temporale.

$$u = \{u_1, u_2, ..., u_n\} \tag{2.18}$$

La scelta della dimensione degli intervalli temporali da utilizzare nell'algoritmo determina l'accuratezza del modello di comportamento degli utenti rappresentato dai microprofili; tipicamente gli intervalli temporali vengono modellati in modo da evidenziare in ciascun microprofilo una serie di interessi simili o ripetitivi nell'intervallo considerato.

2.2.3 Modelli Graph Based

Il fattore temporale può essere introdotto nei metodi di modellazione dei Sistemi di Raccomandazione per individuare la presenza di schemi di voto variabili nel tempo; in tale scenario i *timestamps* degli item votati dagli utenti vengono utilizzati per definire gli intervalli temporali associati a specifici interessi.

La definizione di un modello di evoluzione temporale delle abitudini di voto degli utenti permette di applicare strategie di riduzione dei dati da

processare e di adattare gli algoritmi di raccomandazione ai diversi pattern individuati.

Nel 2009 Cao et al. [61] elaborano una strategia per rappresentare mediante *grafi* l'andamento degli interessi degli utenti all'interno di un Sistema di Raccomandazione.

I ricercatori , attraverso l'analisi delle serie di item votati dagli utenti nel tempo, individuano quattro possibili modelli di evoluzione temporale degli interessi:

Single Interest Pattern (SIP): gli item contenuti nella serie di voti appartengono alla medesima categoria, indicando la presenza di un interesse X univoco.

Multiple Interests Pattern (MIP): gli item contenuti nella serie di voti appartengono a categorie differenti, il cui interesse dura per tutto l'intervallo temporale.

Interests Drift Pattern (IDP): gli item contenuti nella serie di voti appartengono a categorie differenti, il cui interesse è stabile per intervalli estesi ma non per la maggior parte della durata della serie stessa.

Casual Noise Pattern (CNP): gli utem contenuti nella serie di voti appartengono a categorie differenti, il cui interesse dura per intervalli di tempo estremamente ridotti.

Formalmente è possibile rappresentare la serie di voti espressi da un utente attraverso due differenti strutture:

Grafo di voti: i nodi del grafo rappresentano l'insieme degli item associati alla serie di voti dell'utente; due nodi del grafo vengono collegati da un

arco solo se i corrispondenti item sono legati da un valore di similarità superiore ad una certa soglia.

Catena di voti: i nodi della catena rappresentano gli item associati alla serie di voti dell'utente, ordinati in base al tempo di voto crescente; due nodi consecutivi vengono collegati da un link solo se la loro similarità supera un certo valore di soglia.

I valori di *densità del grafo* e di *continuità della catena* di voti permettono di individuare con estrema facilità i modelli di interesse di tipo *SIP* e di tipo *CNP*. Il *Single Interest Pattern*, infatti, è associato alla costruzione di un grafo e di una catena di voti caratterizzati rispettivamente da densità e continuità elevate, dal momento che la serie di voti rappresenta un interesse univoco.

Il *Casual Noise Pattern*, invece, è associato alla costruzione di un grafo e di una catena di voti caratterizzati rispettivamente da densità e continuità ridotte, dal momento che la serie di voti rappresenta un insieme variabile di interessi.

Diversamente i modelli di tipo *MIP* e di tipo *IDP* sono difficilmente individuabili attraverso l'analisi dei grafi e delle catene di voti; viene, quindi, introdotta la nozione di *drift point*, che rappresenta il punto di transizione tra interessi differenti e che determina un'analisi più complessa delle serie di item a disposizione.

Il framework di raccomandazione proposto prevede l'identificazione del modello di interessi dello specifico utente e l'applicazione, quindi, di una differente strategia di raccomandazione in base al pattern rilevato:

SIP e MIP: la raccomandazione viene generata utilizzando l'intera serie di voti dell'utente.

IDP: la raccomandazione viene generata utilizzando i voti successivi al drift point.

CNP: i voti dell'utente non vengono utilizzati per generare la raccomandazione e l'algoritmo si limita a proporre gli item popolari.

Nel 2010 Xiang et al. [62] propongono la costruzione di un *Session Temporal Graph* che permette di rappresentare gli interessi a lungo e a breve termine degli utenti.

I ricercatori basano il proprio lavoro sull'idea che il tempo sia una *dimensione locale* dei Sistemi di Raccomandazione e che come tale vada utilizzato per identificare le relazioni esistenti tra gli item votati dal singolo utente nel tempo; tale strategia prevede, quindi, la suddivisione del tempo in *sessioni* separate, ciascuna rappresentate l'attività di un utente nello specifico intervallo temporale.

L'insieme di sessioni, utenti e item viene utilizzato per costruire un *grafo bipartito diretto*, caratterizzato da associazioni tra utenti e item e associazioni tra item e sessioni; in particolare il link esistente tra un nodo utente ed un set di nodi item rappresenta l'insieme di *preferenze a lungo termine* dell'utente mentre il link esistente tra un nodo sessione ed un set di nodi item rappresenta l'insieme di *preferenze a breve termine* dell'utente cui la sessione si riferisce.

Nella modellazione dei grafi STG le preferenze dell'utente vengono iniettate a partire dai nodi utente e dai nodi sessione, in modo da costituire rispettivamente preferenze a lungo termine e preferenze a breve termine; il grafo risultante viene, quindi, esplorato mediante un algoritmo di *random walk di tipo temporale.*

2.2.4 Sistemi di tipo Context aware

I Sistemi di Raccomandazione classici utilizzano uno spazio di raccomandazione tipicamente costituito dall'insieme di utenti e di item; in realtà i dati che ciascun sistema può raccogliere e catalogare sono eterogenei e permettono in molti casi di sviluppare strategie di profilazione degli utenti, di aggregazione in gruppi omogenei e di predizione complesse, basate non solo sul set di preferenze associate agli item ma anche su informazioni indirette quali il timestamp di voto di un item, la sua categoria di appartenenza o anche la sua provenienza. Le dimensioni rappresentate da queste informazioni possono, quindi, essere integrare negli algoritmi classici di raccomandazione allo scopo di generare risultati adatti al contesto di predizione utilizzato.

Nel 2001 Adomavicius e Tuzhilin [63] [64] [65] propongono la costruzione di un modello di raccomandazione *multidimensionale*; in particolare il modello classico bidimensionale applicato ai Sistemi di Raccomandazione e costituito dall'insieme degli utenti e degli item viene esteso attraverso l'introduzione di nuove dimensioni, quali il fattore temporale o il fattore spaziale.

$$S = User \times Item \times Time \times ... \times Space \tag{2.19}$$

La definizione di un nuovo insieme di contesti associati ai dati del Sistema di Raccomandazione si riflette nel miglioramento della fase di profilazione degli utenti e degli item, che può trarre vantaggio da informazioni sia di tipo testuale, come keyword e descrizioni, sia di tipo numerico, come età e numero di familiari, sia di tipo categorico, come stagioni o genere. L'estensione della dimensionalità del sistema, inoltre, permette di generare predizioni di tipo complesso, i cui contenuti possono essere personalizzati in base alla combinazione delle dimensioni di interesse scelte.

Una delle particolarità del sistema multidimensionale proposto è rappresentata dalla possibilità di organizzare ciascuna dimensione in diversi *livelli gerarchici*: gli utenti, quindi, possono esprimere voti per uno specifico livello della gerarchia ed ottenere come predizioni voti aggregati relativi alla categoria degli item.

Gli algoritmi classici di raccomandazione vengono applicati al contesto multidimensionale mediante una tecnica di *riduzione della dimensionalità*: la funzione di predizione viene associata ad un sistema bidimensionale classico costituito dagli utenti e dagli item del sistema multidimensionale mentre l'insieme di voti utilizzato come input della funzione è costituito dal set di preferenze associate al valore di interesse delle restanti dimensioni. Le dimensioni eliminate dall'algoritmo di riduzione vengono definite *dimensioni contestuali*, in quanto rappresentano lo specifico contesto per il quale la predizione è generata; i voti associati a tali dimensioni possono essere ottenuti dall'aggregazione delle preferenze associate a diversi livelli della gerarchia, in modo da limitare la sparsità dei dati a disposizione.

Le informazioni di contesto associate ai sistemi multidimensionali possono essere utilizzate in diverse fasi del processo di predizione; in particolare è possibile identificare tre strategie di utilizzo:

Contextual prefiltering: le informazioni associate allo specifico contesto vengono utilizzate per selezionare il set di dati rilevanti per la raccomandazione.

Contextual postfiltering: la raccomandazione viene generata utilizzando l'intero set di dati in un modello di tipo bidimensionale e viene successivamente adattata a ciascun utente utilizzando le informazioni di contesto.

Contextual modeling: le informazioni di contesto vengono introdotte nella fase di modellazione come parte del processo di stima dei voti.

Nel 2012 Stefanidis et al. [66] sviluppano un framework per sistemi di tipo time-aware basato sul modello multidimensionale introdotto da Adomavicius e Tuzhilin, utilizzando come contesto di predizione il solo fattore temporale. Nel sistema proposto vengono definiti due nuovi valori di peso per gli item, quali la *relevance*, che rappresenta l'insieme di voti espressi da utenti simili a quello attivo nell'ambito dello stesso contesto della query, e il *supporto*, che rappresenta il numero di utenti che hanno espresso preferenze per l'item nell'ambito dello stesso contesto della query. Per poter generare una raccomandazione relativa ad una specifica query l'algoritmo identifica inizialmente l'insieme di utenti significativi per la query stessa e successivamente calcola i valori di relevance e di supporto per gli item; l'insieme delle raccomandazioni, quindi, viene fornito secondo una strategia *top-k* basata sui valori ottenuti.

Capitolo 3

Twitter time-aware recommendation system

Twitter [67] nasce nel Luglio del 2006 come servizio gratuito di microblogging basato sulla pubblicazione di brevi messaggi di testo (140 caratteri) nella pagina personale di ciascun utente.

Diversamente dagli altri social network presenti nella rete, Twitter non utilizza una struttura sociale basata sulla costruzione di relazioni reciproche tra gli utenti ma consente a ciascuno di stabilire un collegamento univoco con le persone di interesse, senza che queste siano obbligate a stabilire a loro volta il collegamento opposto.

La definizione di un grafo sociale così flessibile, che non si basa sulla necessità di avere una fiducia reciproca sottostante per stabilire una connessione, permette a ciascun utente di selezionare l'insieme di utenti di interesse non solo sulla base di motivazioni sociali quali la conoscenza diretta ma anche sulla base di motivazioni secondarie, quali l'insieme di informazioni di interesse che si desiderano ricevere o i trend correnti; ogni utente, perciò, risulta tipicamente connesso ad utenti globalmente popolari, utenti conosciuti

personalmente, aziende di interesse lavorativo e società di gestione delle news.

La struttura di Twitter può essere ricondotta a quattro elementi fondamentali:

Tweet: rappresenta il messaggio di 140 caratteri che ciascun utente può postare sulla propria pagina personale; i messaggi possono contenere immagini o link e in questo caso la loro lunghezza viene ridotta a 120 caratteri.

HashTag: ciascun messaggio può includere nel testo uno o più tag il cui scopo è quello di categorizzare il contenuto e creare dei link a messaggi associati al medesimo tag; ogni hashtag è rappresentato dal simbolo # seguito da una o più parole concatenate.

Follower: il termine follower identifica all'interno di Twitter l'insieme di persone delle quali si vogliono ricevere i tweet; i messaggi pubblicati da tali utenti appaiono nella bacheca dell'utente che li segue.

Followee: il termine followee identifica all'interno di Twitter l'insieme di persone che vogliono ricevere i tweet di uno specifico utente; i messaggi pubblicati da quest'ultimo, quindi, appaiono sulle bacheche di ogni utente appartenente all'insieme dei suoi followee.

Ad oggi Twitter può contare su circa un miliardo di utenti registrati e 200 milioni di utenti attivi mensilmente [68], con un flusso giornaliero di circa 500 milioni di tweet; l'insieme di messaggi immessi giornalmente nella sua rete, quindi, genera un flusso di informazioni così elevato da non consentire agli utenti di raggiungere tutte le informazioni di interesse. Per questo motivo Twitter, così come gli altri social network esistenti, rappresenta il contesto ideale nel quale sviluppare ed applicare algoritmi più o meno complessi di

raccomandazione, molti dei quali orientati all'individuazione dei corretti url o hashtag da associare ai messaggi.

3.1 Scelte Progettuali

3.1.1 Tipologia di Sistema

Tipicamente l'obiettivo degli algoritmi di raccomandazione esistenti nei social network è quello di individuare l'insieme di utenti i cui contenuti pubblicati possono essere di interesse o che possono trarre vantaggio dai nostri stessi contenuti [69] [70].

Tali strategie di raccomandazione lavorano ad alto livello, fornendo agli utenti un insieme di risultati globali associati al medesimo grado di importanza; in tale scenario, quindi, non esiste differenziazione tra i vari contenuti pubblicati da un utente raccomandato, che possono essere più o meno di interesse. Nei sistemi reali, infatti, gli utenti hanno la tendenza a pubblicare un insieme di dati eterogeneo, legati a differenti contesti di tipo sociale, spaziale o temporale e spesso influenzati dai trend correnti; ciascuna informazione prodotta, quindi, è legata ad un valore di interesse univoco che può renderla adatta o meno ad essere inoltrata ad uno specifico utente.

L'elevata variabilità dei contenuti introdotti nei social network richiede nella maggior parte dei casi l'applicazione di algoritmi di raccomandazione in grado di fornire risultati con un elevato grado di personalizzazione; in Twitter tale obiettivo può essere raggiunto mediante l'utilizzo di algoritmi di tipo *Content Based* che, grazie all'analisi degli hashtag collegati ai messaggi degli utenti delle rete, sono in grado di fornire come predizione messaggi associati a categorie simili a quelle presenti nei messaggi postati dall'utente [71] [72].

Il limite evidente di questo approccio è rappresentato dal fatto che alcuni contenuti postati su Twitter non sono necessariamente associati ad uno o più hashtag e quindi non rientrano nel processo di predizione; in molti casi, inoltre, la presenza di un hashtag di interesse non determina di conseguenza l'esistenza di informazioni di interesse nel contenuto del messaggio. Il nostro lavoro, quindi, cerca di superare questa limitazione sviluppando un algoritmo di raccomandazione di tipo *Collaborative Filtering*, in cui l'insieme di predizioni dipenda dal set di voti espressi direttamente dagli utenti sui tweet visualizzati.

La scelta di adottare una strategia di predizione *"collaborativa"* dipende principalmente dal tentativo di costruire un sistema di raccomandazione altamente personalizzato in cui ciascun utente possa esprimere opinioni sui contenuti dei singoli messaggi in maniera indipendente rispetto all'utente che li ha postati o ai tag presenti; in tale modo ogni voto espresso viene interpretato esclusivamente come una valutazione personale del significato del messaggio ricevuto. In realtà i dati raccolti dal nostro algoritmo e collegati ai tweet votati dagli utenti permettono anche di applicare in un secondo momento strategie per l'individuazione di *pattern di voto* complessi dipendenti da molteplici fattori; l'analisi dell'insieme delle preferenze espresse dagli utenti effettuata a posteriori, infatti, permette di identificare insiemi di voti il cui valore dipende non solo dal contenuto del messaggio ma anche da fattori espliciti, come l'istante temporale del voto, i tag contenuti o l'autore del tweet, e da fattori impliciti come lo stato d'animo dell'utente nel momento del voto od eventi esterni che ne hanno modificato l'opinione.

3.1.2 Schema di voto

Il sistema di raccomandazione sviluppato su Twitter utilizza uno *schema di voto binario*, in cui ciascun utente ha la possibilità di associare ai tweet letti il voto "mi piace", mappato nell'algoritmo sul valore numerico 1, e il voto "non mi piace", mappato nell'algoritmo sul valore numerico -1. L'utilizzo di un insieme di voti ridotto permette di identificare in maniera precisa i gusti degli utenti, senza la necessità di dover interpretare insieme di voti intermedi il cui significato può variare da utente ad utente; in questo modo viene eliminato, di fatto, il fattore di drift, presente in molti sistemi di raccomandazione basati su voti espliciti, legato al differente voto medio utilizzato da ciasun utente.

Il nostro sistema di raccomandazione si differenzia, in particolare, dai sistemi esistenti per la scelta di utilizzare sia nella fase di costruzione delle similarità tra gli utenti che nella fase di predizione anche *l'insieme dei voti negativi* espressi dagli utenti. La decisione di adottare un approccio di questo tipo al problema della raccomandazione dipende da diversi fattori, come il tentativo di comprendere in che modo l'insieme di preferenze negative possa migliorare i valori di similarità calcolati e di conseguenza le predizioni generate o come l'esigenza di ridurre la sparsità della matrice di similarità utilizzando tutti i dati a disposizione.

3.1.3 Componenti software di supporto

Nella fase di progettazione del sistema di raccomandazione da introdurre all'interno del social network Twitter sono state affrontate diverse problematiche, sia di natura architetturale che di natura strettamente implementativa. Il primo problema legato alla natura stessa del social network scelto per il lavoro di tesi è stato quello della mancanza di una funzionalità interna che permettesse di associare ai differenti tweet un voto;

l'impossibilità di agire direttamente sul codice proprietario di Twitter e di modificarne l'interfaccia ha richiesto una soluzione alternativa, basata sulla progettazione di un *plugin esterno* con il compito di associare una funzione di voto a ciascun tweet. La scelta del web browser in cui sviluppare il plugin è stata dettata principalmente da motivazioni legate alla semplicità di implementazione e di pubblicazione dell'estensione; tra i differenti browser a disposizione, perciò, è stato scelto Google Chrome, che consente di sviluppare estensioni attraverso l'utilizzo di Javascript ,del formato JSON e di una serie di API proprietarie.

3.1.4 Modello architetturale

Una seconda problematica incontrata nella fase di progettazione del sistema è stata quella relativa alla scelta della tipologia di architettura da implementare; in generale è possibile identificare due differenti tipologie, una basata su un *modello di tipo centralizzato* nel quale i dati associati ai voti vengono memorizzati in un database centrale unico, e l'altra basata su un *modello di tipo distribuito*, in cui ciascun nodo della rete memorizza una porzione del database dei voti. L'utilizzo dell'una o dell'altra tipologia di sistema dipende dalle caratteristiche che si vogliono ottenere dal sistema finale: la soluzione centralizzata permette di applicare algoritmi di raccomandazione estremamente semplici ma presenta un costo di memorizzazione dei dati che aumenta al crescere del sistema; la soluzione distribuita, invece, assicura alta scalabilità del sistema e basso carico di lavoro su ciascun nodo, sebbene introduca un forte overhead dovuto agli algoritmi di comunicazione tra i nodi necessari per lo scambio dei dati contenuti nelle differenti porzioni del database dei voti.

Per poter beneficiare dei vantaggi offerti da una soluzione distribuita

senza la necessità di sviluppare algoritmi complessi di raccomandazione e di comunicazione tra i nodi presenti nella rete abbiamo adottato una *soluzione di tipo ibrido*, costituita da una fase di calcolo distribuito e da una fase di calcolo centralizzato.

La fase di calcolo distribuito viene strutturata come una fase di pre-processamento dei dati associati ai voti espressi dagli utenti, svolta interamente dai plugin installati sui web browser, il cui obiettivo principale è quello di ridurre il costo computazionale degli algoritmi di calcolo delle similarità e degli algoritmi di predizione.

La funzionalità principale introdotta all'interno di questa fase è un *algoritmo di clusterizzazione* degli utenti basato sull'insieme dei voti espressi: i cluster calcolati singolarmente da ogni nodo del sistema vengono, quindi, utilizzati nella successiva fase di calcolo centralizzato per generare i valori da inserire nella matrice di similarità degli utenti. La scelta di utilizzare come input dell'algoritmo di calcolo delle similarità l'insieme dei cluster generati in questa fase piuttosto che l'insieme di tutti gli item votati dagli utenti, come negli approcci classici, è dettata da motivazioni legate alle performance; il numero di cluster generati in ogni iterazione, infatti, è costante e limitato mentre il numero di item votati da ogni singolo utente è un valore non prevedibile che può assumere una dimensione elevata e quindi determinare un aumento significativo dei tempi di calcolo dei valori di similarità.

La fase di calcolo centralizzata utilizza i dati raccolti ed elaborati nella fase di calcolo distribuita; le informazioni associate agli utenti, agli item, ai cluster e ai voti vengono, infatti, inviate al database centrale, che le memorizza e le rende disponibili all'utilizzo negli algoritmi di predizione.

3.1.5 Algoritmo di raccomandazione

Nella fase di progettazione dell'algoritmo di raccomandazione è stata posta particolare attenzione allo sviluppo di strategie per la riduzione del costo computazionale e del costo di memorizzazione delle informazioni necessarie al calcolo delle similarità. Una delle scelte effettuate in tale contesto è l'utilizzo di un algoritmo di raccomandazione Collaborative Filtering di tipo *User Based*, il cui funzionamento si basa sulla costruzione di una matrice di similarità tra utenti, la cui dimensione è notevolmente inferiore rispetto alla matrice di similarità tra item richiesta in un approccio di tipo item based. La scelta della tipologia di algoritmo da adottare è stata dettata principalmente dall'esigenza di costruire ed utilizzare strutture dati la cui dimensione fosse più contenuta possibile e riflette le dinamiche di evoluzione del social network nel tempo.

Twitter è un sistema caratterizzato da insiemi di utenti e di item la cui dimensione aumenta al crescere del tempo; le velocità di crescita dei due insiemi, tuttavia, sono notevolmente differenti ed è possibile osservare come giornalmente il numero di nuovi utenti presenti nel sistema sia pari a circa 500.000 unità, a fronte di circa 180 milioni di tweet postati. La matrice utilizzata per rappresentare le relazioni tra utenti, quindi, ha una dimensione e un fattore di crescita nettamente inferiore rispetto a quella associata agli item e permette di ridurre sensibilmente il numero di aggiornamenti eseguiti ad ogni iterazione dell'algoritmo.

Il costo computazionale dei singoli algoritmi sviluppati viene ulteriormente ridotto attraverso l'adozione di un sistema di raccomandazione di tipo *time-aware*: i processi di creazione e aggiornamento della matrice di similarità e di predizione vengono eseguiti in specifici intervalli temporali ed associano ad ogni voto utilizzato in input un timestamp relativo all'istante

temporale in cui è stato generato. L'introduzione del fattore temporale nel sistema di raccomandazione da noi proposto permette di definire in maniera chiara l'età di ogni singola preferenza e di distinguere, quindi, l'insieme dei voti recenti e l'insieme dei voti meno recenti; tale differenziazione porta alla realizzazione di uno schema di peso per i tweet i cui valori sono basati su una funzione di tipo esponenziale legata ai timestamp dei voti. L'adozione di una funzione di peso temporale non solo consente di individuare per ciascun utente la variazione nel tempo degli interessi e dei trend seguiti ma contribuisce in maniera efficace alla riduzione del costo computazionale dei singoli metodi applicati, attraverso l'esclusione dalla fase di processamento dei tweet non significativi associati a valori di peso inferiori rispetto ad una soglia prefissata.

Gli algoritmi di calcolo delle similarità e di predizione vengono eseguiti all'interno del sistema di raccomandazione in intervalli prefissati e ad ogni iterazione utilizzano i nuovi dati inseriti nel sistema per aggiornare la matrice di similarità; tale struttura consente la progettazione di un *algoritmo di tipo incrementale*, in cui gli aggiornamenti della matrice vengono effettuati attraverso il calcolo del solo fattore di variazione rispetto ai valori calcolati nelle iterazioni precedenti.

Un approccio di questo tipo permette di limitare il costo computazionale della fase di calcolo delle similarità rispetto ad un approccio non incrementale classico, riducendo di conseguenza anche il costo complessivo dell'algoritmo di raccomandazione.

3.2 Architettura del Sistema

Il sistema di raccomandazione time-aware sviluppato per il social network Twitter è stato concepito come un sistema di tipo ibrido, nel quale è possibile individuare una *componente distribuita* presente nei nodi della rete e una *componente centralizzata* ospitata su un server remoto.

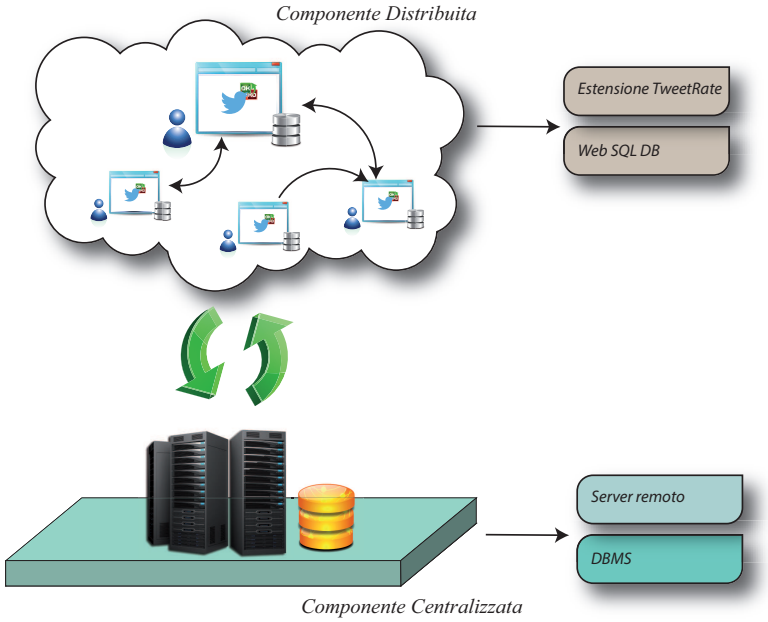


Figura 3.1: Architettura ibrida del sistema

L'architettura scelta per la realizzazione di tale sistema è una architettura modulare di tipo *Three-Tier*, costituita da un livello di presentazione, da un livello di logica di business e da un livello dati.

Il livello di presentazione ha il compito di gestire la comunicazione tra l'utente finale e il sistema e di organizzare in maniera chiara i dati forniti in output. Nella nostra architettura viene posto all'interno delle estensioni Chrome installate nei nodi client ed è associato ad una serie di funzionalità il cui scopo è la modifica dell'interfaccia standard di Twitter; in particolare

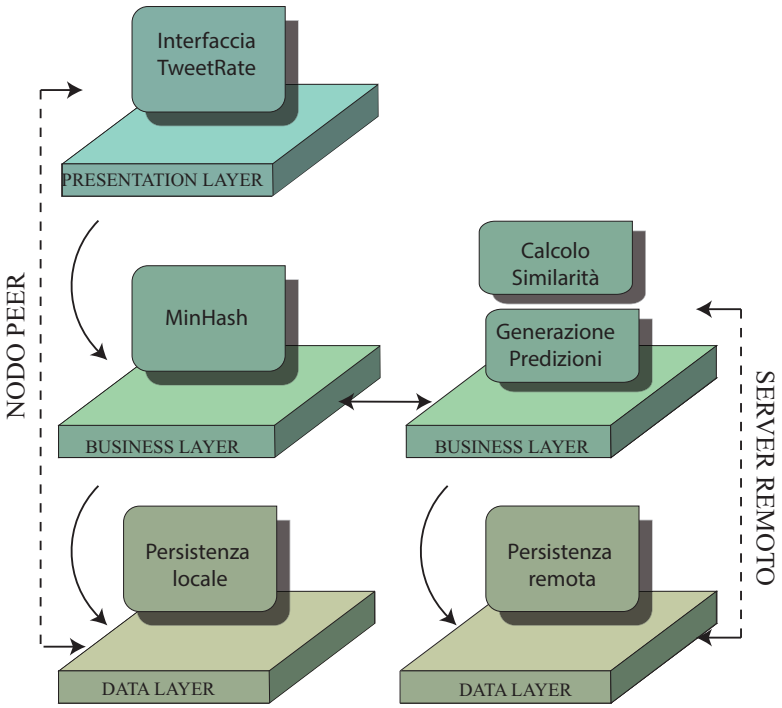


Figura 3.2: Three Tier Architecture

la logica presente in tale livello permette l'inserimento di speciali pulsanti per esprimere preferenze sui tweet, la marcatura dei messaggi già votati e la presentazione in output delle raccomandazioni generate dal sistema.

Il livello di logica del business è il cuore dell'architettura ed è costituito dall'insieme di algoritmi progettati per il processamento dei dati; posto tra il livello di presentazione e il livello dati, si occupa anche di gestire il passaggio dei dati tra i due livelli. Nella nostra architettura parte della logica applicativa viene inserita nelle estensioni Chrome presenti sui nodi client mentre la restante parte è presente nel server remoto.

Il livello di business realizzato all'interno delle estensioni si occupa di gestire una fase iniziale di processamento dei dati associati ai voti espressi dagli utenti; in particolare l'insieme delle preferenze di ciascun utente viene

utilizzato come input di un metodo di clusterizzazione basato su hash, il cui scopo è quello di generare il set di insiemi di appartenenza utile alla successiva fase di calcolo delle similarità.

La parte del livello di business implementata nel server remoto, invece, è costituita dagli algoritmi di calcolo delle similarità e di predizione necessari alla generazione delle raccomandazioni; per implementare tali funzionalità il livello di business si interfaccia direttamente con il livello sottostante dei dati.

Il livello dati, infatti, ha il compito di gestire la persistenza dei dati forniti in input dagli utenti utili agli algoritmi di raccomandazione; nella nostra architettura è realizzato mediante un DBMS posto su un server remoto nel quale vengono memorizzate le informazioni associate ad utenti, tweet e voti. Parte del livello di persistenza dei dati viene anche realizzato all'interno delle estensioni Chrome con il compito di memorizzare temporaneamente le informazioni prodotte dall'utente in caso di inattività del server remoto; tali informazioni vengono, quindi, inviate dal database locale al database remoto non appena il server torna attivo.

L'architettura realizzata, quindi, può essere sinteticamente descritta attraverso tre componenti principali:

Estensione Chrome: è un plugin sviluppato per il web browser Google Chrome; le sue funzioni principali sono la modifica dell'interfaccia standard di Twitter per l'introduzione della funzionalità di voto e la gestione della raccolta e dell'invio dei dati associati alle preferenze espresse dagli utenti.

Database centralizzato: è un DBMS ospitato su un server remoto; memorizza le informazioni inviate dal plugin di ciascun utente e fornisce i dati necessari come input per gli algoritmi di raccomandazione.

Motore di raccomandazione: è il componente principale del sistema di raccomandazione ospitato su un server remoto; è costituito dall'insieme di algoritmi necessari per generare le raccomandazioni finali, come gli algoritmi di calcolo delle similarità tra utenti e gli algoritmi di predizione dei voti.

3.3 Estensione Google Chrome

TweetRate è l'estensione realizzata per il web browser Chrome con l'obiettivo di modificare l'interfaccia standard di Twitter ed inserire la funzionalità di voto per i tweet. L'estensione è stata realizzata in Javascript ed utilizza il formato JSON per la comunicazione con le API fornite da Twitter e con il DBMS remoto.

Una delle caratteristiche dell'estensione è l'utilizzo di un database SQL locale per la memorizzazione delle chiavi di utilizzo delle API di Twitter e dei dati associati ai voti degli utenti generati nell'ultima connessione con il social network; a tale scopo viene utilizzato il *Web SQL Database*, che fa parte di una suite di specifiche elaborate a supporto di HTML5.

L'utilizzo di un livello di persistenza locale permette di ottimizzare la procedura di marcatura dei tweet già votati dall'utente presenti nella sua pagina principale; i dati associati ai voti dei tweet espressi nell'ultima connessione con il social network vengono, infatti, memorizzati e prelevati localmente, senza la necessità di stabilire una connessione con il server remoto. Tale strategia presenta due vantaggi: da un lato vengono ridotti i tempi di latenza per la visualizzazione dei voti già espressi dagli utenti e dall'altro lato viene ridotto il carico di lavoro del server remoto, che non deve prelevare dal DBMS i dati associati agli ultimi tweet votati ad ogni

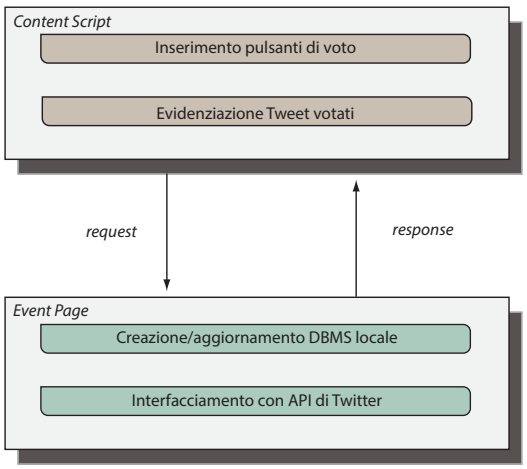


Figura 3.3: Struttura TweetRate

connessione.

Il livello di persistenza locale viene utilizzato anche per gestire situazioni nelle quali è impossibile inviare i dati prodotti dagli utenti al server remoto a causa di problemi di connessione o di inattività del server: l'estensione, infatti, memorizza nel database SQL locale i dati non inviati al DBMS remoto e li cancella solo dopo averli inviati con successo.

Strutturalmente l'estensione è costituita da due elementi fondamentali, il *Content Script* e l'*Event Page*.

3.3.1 Content Script

Il Content Script è lo script utilizzato per interagire con l'interfaccia di Twitter e modificarne l'aspetto; tale script si attiva solo nel contesto della pagina web di Twitter ed utilizza la libreria *JQuery* per accedere agli elementi della pagina HTML associati ai tweet ed inserire i pulsanti di voto.

Lo script è stato progettato in modo da realizzare differenti funzionalità, alcune relative al livello di presentazione ed altre relative al livello di business

logic:

- Lo script verifica, per ogni utente che accede alla pagina di Twitter, l'esistenza del database locale e lo stato dei dati memorizzati; nel caso in cui il database non esista o i dati non siano aggiornati invoca una procedura per la creazione del database e per il reperimento dei dati sul server remoto.

- Quando la procedura di verifica è terminata, lo script si occupa di modificare l'interfaccia standard di Twitter, inserendo due pulsanti di voto al di sotto di ciascun tweet visibile nella pagina dell'utente.

- Se uno o più tweet sono stati votati precedentemente dall'utente, lo script si occupa di evidenziarli ed inserire il numero di voti totali ricevuti dagli utenti.

- Quando un utente vota uno specifico tweet, lo script si occupa di cambiarne il colore in modo da evidenziare il voto espresso (rosso per i voti negativi e verde per i voti positivi); successivamente rimuove i pulsanti di voto al di sotto del tweet così che un utente non possa esprimere più di una preferenza per lo stesso item.

Le funzionalità di business logic realizzate all'interno del Content Script, come il reperimento dei dati relativi agli ultimi tweet votati o la funzione di voto, sono in realtà delle semplici chiamate a funzioni presenti all'interno dell'Event Page.

Il paradigma di comunicazione utilizzato per il passaggio dei dati tra i due script è un paradigma basato sullo scambio di messaggi: l'invocazione di una funzione implementata all'interno dell'Event Page avviene attraverso l'invio di un messaggio contenente i dati di input della funzione stessa; al termine

dell'esecuzione della funzione l'Event Page invia un messaggio di risposta contenente i risultati.

3.3.2 Event Page

L'Event Page è lo script utilizzato per realizzare l'intera logica di business all'interno dell'estensione; lo script è modellato secondo un paradigma ad eventi, che ne permette l'attivazione solo in risposta a specifici avvenimenti, come la ricezione di un messaggio dal Content Script o l'installazione dell'estensione stessa. Tale caratteristica consente un notevole risparmio delle risorse del sistema, che vengono rilasciate quando l'Event Page diventa inattiva.

La gestione degli eventi di attivazione dell'Event Page e quindi dei messaggi scambiati con il Content Script avviene attraverso la definizione di specifici listener collegati agli avvenimenti di interesse:

Installazione dell'estensione: nella fase di installazione dell'estensione l'Event Page deve svolgere differenti compiti, alcuni legati al settaggio delle strutture dati locali per il mantenimento delle informazioni di interesse e altri legati al reperimento dei dati associati agli utenti sui server privati di Twitter. Il primo passo compiuto dallo script è la creazione del database SQL locale nel quale i dati necessari al funzionamento dell'estensione vengono memorizzati. Successivamente l'Event Page si interfaccia con i database di Twitter attraverso l'invocazione di specifiche API in modo da ottenere i dati necessari alla creazione del profilo dell'utente, come l'insieme dei follower, l'insieme dei followee e il numero di tweet pubblicati; tali informazioni vengono, quindi, inviate al DBMS remoto e memorizzate in apposite tabelle.

Accesso di un utente alla pagina di Twitter: quanto un utente accede alla pagina di Twitter l'Event Page si occupa di effettuare una serie di verifiche sui dati memorizzati nel database SQL locale; in particolare viene verificato lo stato di aggiornamento degli ultimi voti espressi dall'utente e memorizzati localmente e la presenza di voti non ancora inviati al DBMS remoto. La procedura di aggiornamento dei voti memorizzati localmente viene eseguita ad intervalli di almeno 24 ore, in modo da limitare il carico di lavoro richiesto al server per il reperimento delle informazioni; viene, quindi, utilizzato un timestamp per memorizzare l'istante di esecuzione dell'ultima procedura di aggiornamento effettuata e determinare quando effettuarne una nuova. Il database locale può mantenere in memoria i voti generati nell'iterazione precedente dell'utente con il sistema e non ancora inviati al DBMS remoto; l'Event Page in questo caso si occupa di inviare l'insieme di tali dati al server remoto.

Voto di un tweet: quando un utente esprime una preferenza su uno specifico tweet, l'Event Page preleva attraverso le API di Twitter una serie di dati, quali l'id del tweet, il timestamp, i tag associati, l'autore, il numero di retweet e il testo. Successivamente si occupa di inviare tali dati al DBMS remoto o di memorizzarli localmente in caso di inattività del server.

L'accesso da parte dell'Event Page alle API fornite da Twitter avviene attraverso l'utilizzo del protocollo di autenticazione open *OAuth* [73].

OAuth nasce nel 2006 con l'obiettivo di consentire ad un applicazione esterna l' accesso sicuro ai dati sensibili degli utenti senza la condivisione delle loro credenziali. Nei modelli classici Client-Server l'accesso ai dati riservati presenti su un server remoto avviene attraverso l'autenticazione

dell'utente, che si occupa di fornire al server le proprie credenziali; l'utilizzo di un'applicazione di terze parti per l'accesso alle stesse risorse, quindi, richiede la condivisione di tali credenziali, che vengono memorizzate all'interno dell'applicazione per usi futuri.

Tale schema genera diverse falle di sicurezza nelle procedure di autenticazione:

- Le applicazioni di terze parti ottengono un accesso completo alle informazioni dell'utente; la compromissione di tali applicazioni, quindi, consente ad utenti maliziosi l'accesso alle informazioni sensibili.
- L'utente non può limitare la durata delle credenziali fornite alle applicazioni di terze parti o definire un sottoinsieme di informazioni riservate e non accessibili.
- L'utente può revocare l'accesso ad un'applicazione solo modificando le proprie credenziali; in tale modo vengono bloccate tutte le applicazioni di terze parti che le utilizzano.

OAuth viene introdotto per superare tali limitazioni, introducendo un livello di autorizzazione che separa il *client*, l'applicazione di terze parti, dal *resource-owner*, l'utente. In questo schema il client ottiene una serie di credenziali per l'accesso alle risorse differenti da quelle possedute dal resource-owner e caratterizzate da durata di accesso, ambito di applicazione e altri attributi che ne determinano il contesto di utilizzo.

All'interno del flusso di autenticazione implementato da OAuth è possibile distinguere quattro differenti ruoli:

Resource Owner: è l'entità che consente l'accesso alle risorse protette e tipicamente è rappresentato dall'utente finale.

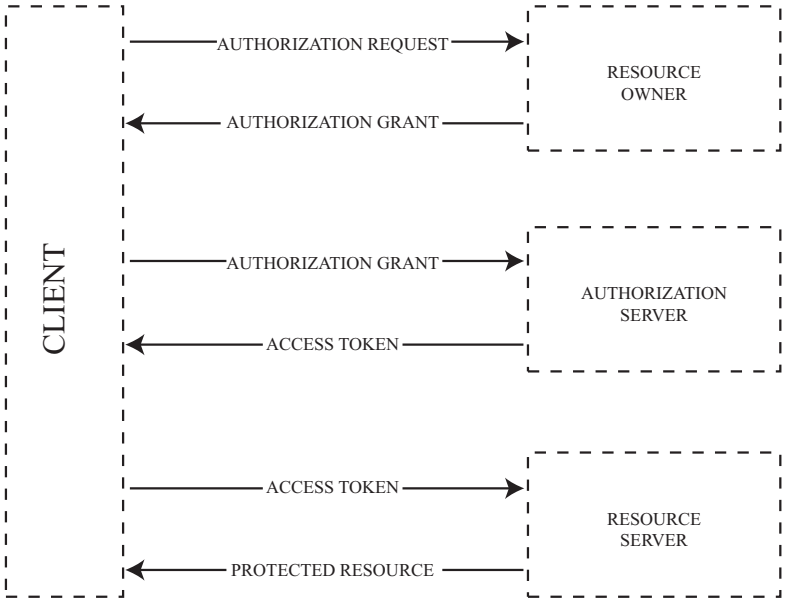


Figura 3.4: Flusso OAuth

Resource Server: è il server che mantiene l'insieme delle risorse protette; nel nostro scenario è rappresentato dai server di Twitter.

Client: è l'applicazione di terze parti che vuole accedere ai dati sensibili; nel nostro scenario è rappresentato dall'estensione Chrome.

Authorization Server: è il server che fornisce al client il token di accesso dopo l'autenticazione; nel nostro scenario è rappresentato dai server di autenticazione di Twitter.

L'interazione dell'estensione con i server di Twitter richiede l'invio di una *Consumer Key* e di un *Consumer Secret*, rilasciati da Twitter al momento della registrazione dell'applicazione; tali chiavi sono collegate a specifici livelli di accesso assegnati alle applicazioni e permettono di determinare l'origine di ciascuna richiesta effettuata tramite API. Il flusso di autenticazione necessario al rilascio dei token di accesso avviene in tre diverse macro-fasi,

nelle quali il Client si interfaccia con il Resource Owner, con l'Authorization Server e con il Resource Server:

Client ↔ Authorization Server

1. Il Client invia una richiesta all'Authorization Server per ottenere un Request Token; la richiesta è segnata e contiene la Consumer Key.
2. L'Authorization Server verifica la firma della richiesta e genera un Request Token e un Request Secret; il Request Token è una chiave di accesso temporanea utilizzata per richiedere all'utente l'accesso alle informazioni sensibili.

Client ↔ Resource Owner

3. Il Client ridireziona il Resource Owner sul sito di Twitter per ottenere il consenso all'accesso ai dati.
4. Il Resource Owner invia al client un Authorization Grant, la credenziale che rappresenta l'autorizzazione ricevuta dall'utente per l'accesso ai dati.

Client ↔ Resource Server

5. Il Client richiede al Resource Server un Access Token per l'accesso ai dati sui server di Twitter; la richiesta contiene il Consumer Key per l'autenticazione del client e il Request Token per l'autorizzazione ricevuta.

6. Il Resource Server verifica le credenziali della richiesta ricevuta e genera un Access Token e un Token Secret; tali credenziali verranno utilizzate successivamente all'interno di ogni richiesta effettuata alle API di Twitter.

L'Access Token ottenuto alla fine del flusso di autenticazione viene inserito nelle richieste effettuate dall'estensione ai server di Twitter ed è associato ad una durata limitata e ad un contesto di utilizzo; allo scadere del token viene avviato un nuovo flusso OAuth per ottenere un set di credenziali da utilizzare nelle successive interazioni con i server di Twitter.

3.4 DMBS

Il livello di persistenza dei dati all'interno del nostro sistema di raccomandazione viene realizzato mediante un *DBMS* remoto installato su una macchina Linux. Il database ha il compito di memorizzare l'insieme dei dati associati agli utenti e ai tweet che vengono inviati attraverso l'estensione Chrome e di renderli disponibili in input al motore di raccomandazione del sistema.

L'interfacciamento con il DBMS avviene mediante una serie di script PHP installati sul server remoto; ciascuno script realizza una differente funzionalità invocabile dall'estensione mediante specifiche chiamate AJAX. Le istruzioni SQL utilizzate per gestire l'inserimento e la modifica dei dati e l'interrogazione del database sono state realizzate attraverso la tecnica dei *Prepared Statements*. Nella fase di preparazione il Database Engine effettua il parsing del template definito per i comandi SQL da eseguire e lo memorizza all'interno del database; successivamente nella fase di esecuzione

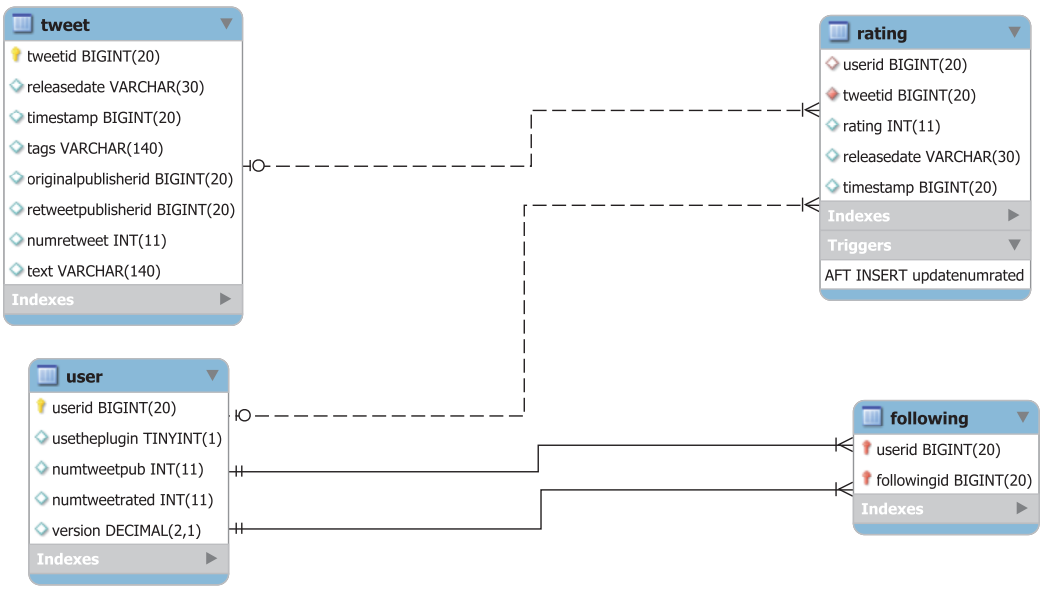


Figura 3.5: Diagramma ER del DBMS remoto

viene eseguito il bind tra i parametri presenti nel template dei comandi e i valori inviati agli script PHP.

La scelta di utilizzare tale tecnica progettuale è legata ai vantaggi che offre rispetto alle funzioni SQL classiche:

- L'esecuzione di comandi SQL ripetuti viene ottimizzata dal momento che il Database Engine esegue una sola volta le procedure di parsing e memorizzazione dei comandi.

- La separazione tra i dati e le dichiarazioni SQL elimina la possibilità di effettuare SQLInjection nel database; nella fase di bind, infatti, l'insieme dei valori passato alle Prepared Statement viene associato alla specifica tipologia di parametro che devono rappresentare. In questo modo eventuali comandi SQL inseriti forzatamente nel sistema vengono interpretati come semplici stringhe testuali, impedendone l'esecuzione.

Il database utilizza quattro differenti tabelle per la memorizzazione dei dati necessari all'algoritmo di raccomandazione:

Tabella Utenti: rappresenta il profilo degli utenti che hanno installato l'estensione Chrome; è costituita dai seguenti campi:

1. *Userid:* id univoco assegnato all'utente da Twitter.
2. *UseThePlugin:* valore booleano che segnala l'utilizzo dell'estensione.
3. *NumTweetPub:* numero dei tweet pubblicati dall'utente.
4. *NumTweetRated:* numero dei tweet votati dall'utente attraverso l'estensione.
5. *Version:* versione dell'estensione installata.

Tabella Following: rappresenta l'insieme degli utenti seguiti da uno specifico utente; è costituita dai seguenti campi:

1. *Userid:* id univoco assegnato all'utente da Twitter.
2. *Followingid:* id univoco assegnato da Twitter all'utente seguito.

Tabella Tweet: rappresenta l'insieme di dati associati ai tweet votati dagli utenti; è costituito dai seguenti campi:

1. *Tweetid:* id univoco assegnato da Twitter al tweet.
2. *ReleaseDate:* data di creazione del tweet in formato testuale.
3. *Timestamps:* timestamp associato all'istante di creazione del tweet in formato Unix.
4. *Tags:* insieme degli hashtag contenuti nel tweet.

5. *OriginalPublisherId:* id associato all'utente che ha pubblicato il tweet.
6. *RetweetPublisherId:* id associato all'utente che ha retwettato il tweet.
7. *NumRetweet:* numero di volte che il tweet è stato retwittato.
8. *Text:* testo del tweet.

Tabella Rating: rappresenta l'insieme di voti espressi dagli utenti; è costituito dai seguenti campi:

1. *Userid:* id univoco dell'utente che ha votato il tweet.
2. *Tweetid:* id univoco del tweet votato.
3. *Rating:* voto espresso dall'utente; può assumere valore 1 o -1.
4. *ReleaseDate:* data di voto del tweet in formato testuale.
5. *Timestamp:* timestamp associato all'istante di voto del tweet in formato Unix.

3.5 Motore di Raccomandazione

Il motore di raccomandazione, posto su di un server remoto, rappresenta l'elemento centrale dell'intero sistema di raccomandazione; come tale, si occupa dell'esecuzione degli algoritmi di similarità e di raccomandazione, che costituiscono gran parte del livello di business del sistema.

L'algoritmo progettato per realizzare la logica applicativa si basa su un *modello time-aware di tipo incrementale*; la particolarità di tale approccio risiede nella modalità di calcolo dei valori di similarità, che vengono aggiornati ad ogni intervallo temporale attraverso l'aggiunta di un valore

di variazione legato ai soli voti espressi all'interno dell'intervallo corrente. Tale strategia riduce, quindi, il costo computazionale dell'algoritmo di raccomandazione, limitando l'esecuzione del metodo di calcolo delle similarità solo ai voti nuovi entrati nel sistema nell'ultimo intervallo temporale e impedendo, di fatto, il calcolo sull'intero dataset presente nel database remoto.

L'approccio da noi scelto per la progettazione del motore di raccomandazione è la costruzione di un *sistema neighborhood-based* il cui funzionamento si appoggia sulla generazione di una matrice di similarità tra utenti; i valori contenuti all'interno della matrice permettono di definire il neighborhood associato a ciascun utente, utilizzato nella successiva fase di predizione per individuare tweet non ancora votati ma di interesse.

Il sistema progettato è legato a tre caratteristiche fondamentali:

Intervalli Temporali: l'algoritmo di calcolo delle similarità e l'algoritmo di raccomandazione vengono eseguiti periodicamente ad intervalli di tempo regolari; l'insieme dei dati utilizzati come input è costituito dai voti espressi all'interno dell'intervallo corrente.

L'utilizzo di intervalli temporali offre diversi vantaggi; il più evidente è la possibilità di modellare l'andamento degli interessi espressi dall'utente nel tempo ed adattare l'insieme di raccomandazioni ai trend di voto correnti. La scelta corretta della dimensione degli intervalli utilizzati all'interno degli algoritmi, principalmente legata alla natura del dataset utilizzato, permette di limitare la loro esecuzione in specifici istanti caratterizzati da un insieme di nuovi voti in input sufficiente a determinare le preferenze espresse dagli utenti. La definizione degli intervalli permette, inoltre, di strutturare gli algoritmi di raccomandazione secondo uno schema di calcolo incrementale; in

tale modo il costo computazionale dell'intera fase di processamento del dataset viene ridotto rispetto a quello di un approccio non incrementale, nel quale l'intera matrice di similarità deve essere calcolata successivamente all'ingresso del sistema di nuovi dati di input.

Schema di Clustering: gli utenti del sistema di raccomandazione vengono raggruppati in differenti cluster, generati attraverso l'applicazione di uno specifico algoritmo di clusterizzazione all'insieme di voti espressi all'interno dell'intervallo temporale corrente. Formalmente ciascun cluster contiene al suo interno utenti che hanno votato set di item per lo più identici e che sono, quindi, legati da valori di similarità elevati.

Negli algoritmi di raccomandazione classici l'insieme dei cluster viene utilizzato per implementare efficacemente i metodi di predizione di tipo neighborhood based: dopo aver costruito i cluster di appartenenza attraverso l'applicazione di una qualsiasi funzione di aggregazione, viene eseguita la funzione di predizione utilizzando i soli dati in input relativi agli utenti presenti nello stesso cluster dell'utente target.

Diversamente nel nostro approccio la funzione di clusterizzazione viene introdotta per implementare un metodo di calcolo delle similarità incrementale che sia in grado di cogliere l'andamento nel tempo delle similarità tra coppie di utenti senza la necessità di utilizzare nel calcolo l'intera storia pregressa dei voti. La fase di calcolo delle similarità da noi progettata, infatti, si basa sull'idea che il valore di similarità tra coppie di utenti possa essere determinato attraverso l'analisi del numero di cluster condivisi nel tempo; valori di similarità elevati, quindi, corrispondono ad utenti che hanno condiviso un numero consistente di cluster mentre valori di similarità ridotti vengono associati ad utenti

che nel tempo hanno partecipato a cluster differenti. I cluster generati dall'analisi delle preferenze espresse dagli utenti durante l'interazione con il sistema di raccomandazione vengono utilizzati come input della funzione di similarità basata sulla metrica della cosine similarity:

$$S_{i,j}(t) = \frac{\sum_{c \in C_i \cap C_j} f_{c_i}^{\alpha}(t) \cdot f_{c_j}^{\alpha}(t)}{\sqrt{\sum_{c \in C_i} (f_{c_i}^{\alpha}(t))^2 \sum_{c \in C_j} (f_{c_j}^{\alpha}(t))^2}} \tag{3.1}$$

dove $f_{c_i}^{\alpha}(t)$ è il peso per l'utente *i-esimo* del cluster di appartenenza c nell'intervallo temporale t.

I valori così ottenuti rappresentano un'approssimazione dei reali valori di similarità calcolati attraverso un approccio classico basato sulla storia dei voti di ciascun utente. Le funzioni di similarità tradizionali, infatti, confrontano l'intero set di voti espressi da ciascuna coppia di utenti del sistema e generano un valore che rappresenta il grado di vicinanza relativo all'istante nel quale viene effettuato il calcolo; tale approccio risulta generalmente affidabile ma è penalizzato da un costo computazionale elevato che dipende dal numero di preferenze espresse dall'utente all'interno del sistema di raccomandazione.

Nei sistemi caratterizzati da utenti attivi e da un flusso di informazioni costante e illimitato, quale Twitter, il costo dell'operazione di confronto dei voti associati ai tweet può diventare in breve tempo insostenibile, soprattutto per quegli utenti o per quei profili che generano quotidianamente un numero consistente di tweet. Il nostro approccio, invece, sacrifica parte dell'affidabilità dei risultati ottenuti ma permette di realizzare una funzione di similarità il cui costo per coppie di utenti è indipendente dal numero di voti espressi e limitato. Il numero di cluster generati all'interno di ciascun intervallo temporale, infatti, dipende esclusivamente dai parametri scelti per la funzione di

clusterizzazione ed è costante per ciascun utente. In questo modo è possibile realizzare un algoritmo di calcolo delle similarità i cui tempi di esecuzione rimangono costanti e limitati nel tempo nonostante la crescita continua del sistema di raccomandazione dovuta all'ingresso di nuovi utenti e item e alla generazione di nuovi voti.

Funzione di decadimento temporale: il fattore temporale viene introdotto nella nostra architettura non solo per suddividere l'esecuzione degli algoritmi di raccomandazione in intervalli ma anche per modellare l'andamento nel tempo degli interessi degli utenti attraverso l'applicazione di una funzione di peso.

Gli algoritmi classici basati su funzioni di peso applicate ai voti utilizzano come parametri di interesse la differenza temporale tra l'istante di generazione della preferenza e l'istante di generazione della predizione e un valore α che modula la velocità di decadimento; in tale modo ogni singola preferenza è associata ad un valore di peso univoco che permette di selezionare all'interno dell'intero dataset le preferenze degli utenti secondo un ordine decrescente dei pesi.

Diversamente il nostro approccio utilizza una tipologia di ordinamento tra le preferenze degli utenti legata all'intervallo di appartenenza piuttosto che al singolo istante di generazione; la funzione di peso elaborata, infatti, applica ai voti contenuti in uno specifico intervallo lo stesso valore di decadimento temporale, caratterizzato dal solo parametro di velocità α.

$$f^{\alpha}_{r_{u_i}}(t_0) = e^{-\alpha} \tag{3.2}$$

$$f^{\alpha}_{r_{u_i}}(t_k + 1) = e^{-\alpha} \cdot f^{\alpha}_{r_{u_i}}(t_k) \tag{3.3}$$

L'applicazione di tale funzione ai voti espressi negli intervalli precedenti fa si che al termine di ogni intervallo temporale il peso di ciascuna preferenza venga ridotto di un fattore costante $e^{-\alpha}$.

La stessa funzione di decadimento viene applicata anche ai cluster di appartenenza degli utenti, in modo da modellare l'evoluzione temporale degli interessi dei singoli individui nell'intero arco temporale associato al dataset utilizzato:

$$f_{c_i}^{\alpha}(t_0) = e^{-\alpha} \tag{3.4}$$

$$f_{c_i}^{\alpha}(t_k + 1) = e^{-\alpha} \cdot f_{c_i}^{\alpha}(t_k) \tag{3.5}$$

L'utilizzo di un decadimento costante legato agli intervalli temporali permette di creare, quindi, un modello di sistema nel quale gli interessi degli utenti evolvono a distanza di intervalli regolari, rimanendo costanti all'interno dello stesso intervallo temporale. Il corretto dimensionamento degli intervalli temporali permette di cogliere all'interno del sistema di raccomandazione l'andamento dei differenti trend di voto, legati ad eventi esterni che influenzano grandi insiemi di utenti o ad eventi particolari che colpiscono un singolo utente.

L'intero processo di raccomandazione può essere suddiviso in tre differenti fasi:

Fase di Collect: è la fase in cui vengono collezionate le informazioni associate agli item, agli utenti e ai voti presenti nel sistema di raccomandazione. Tale fase viene svolta interamente dalle estensioni Chrome installate sui browser degli utenti; completata la procedura di raccolta dei dati, le estensioni si fanno carico dell'esecuzione dell'algoritmo di clusterizzazione degli utenti.

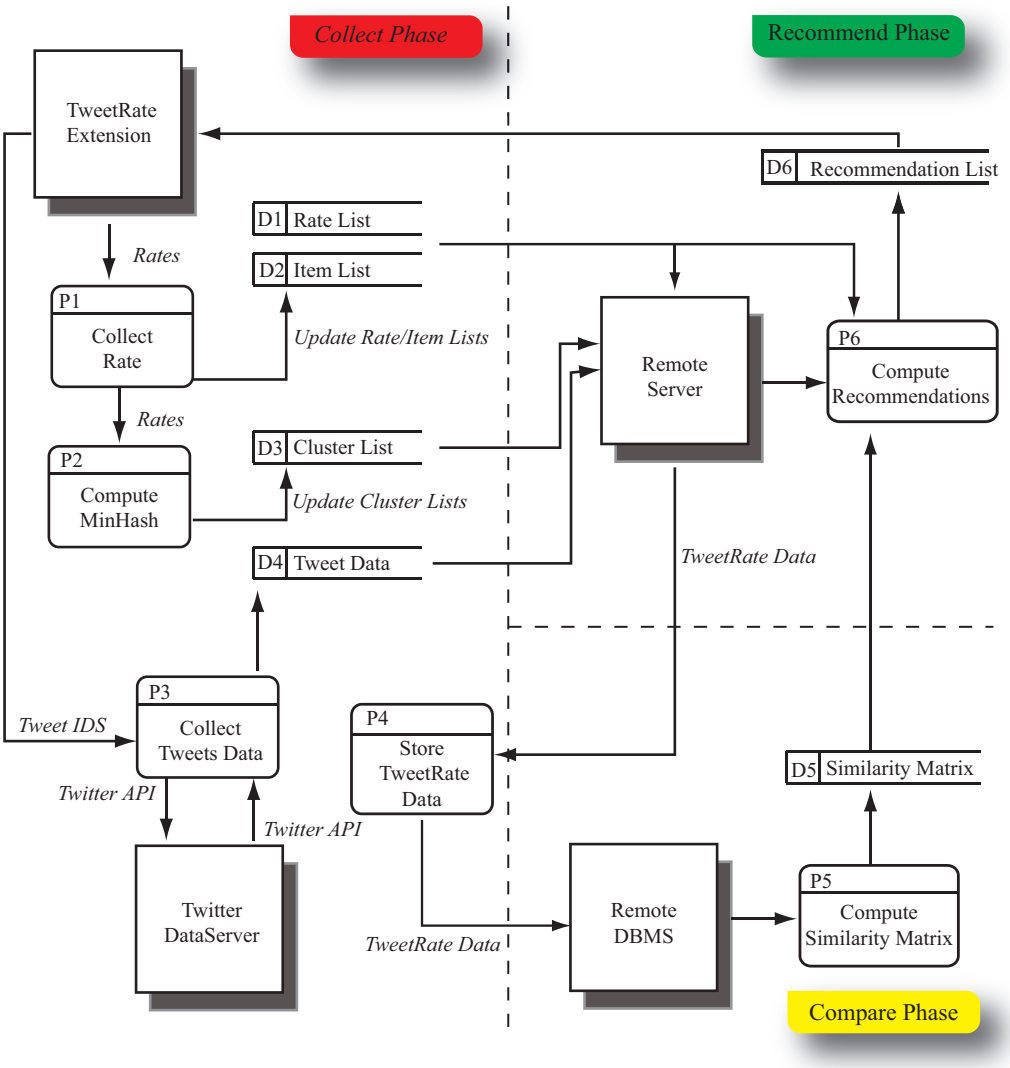


Figura 3.6: Fasi del processo di raccomandazione

Fase di Compare: è la fase nella quale vengono calcolati i valori di similarità tra gli utenti, svolta all'interno del server remoto. L'algoritmo di similarità implementato utilizza come input i cluster generati nella fase precedente e memorizza i valori ottenuti in una matrice di similarità. All'interno della fase di Compare viene anche applicata la funzione di peso per aggiornare i valori associati agli item votati negli intervalli precedenti.

Fase di Recommend: è la fase in cui viene generata la lista di item raccomandati per ciascun utente. L'input di tale fase è costituito dai valori di similarità calcolati nella fase precedente, che permettono di costruire il neighborhood associato a ciascun utente; l'insieme delle raccomandazioni generate dall'algoritmo viene organizzato in una lista top-k, nella quale sono presenti gli item associati al punteggio più alto.

3.5.1 Strutture Dati

Il sistema di raccomandazione da noi elaborato basa il proprio funzionamento su un insieme di strutture dati create a supporto degli algoritmi di similarità e di predizione. La struttura centrale dell'intero sistema è rappresentata da una matrice S $n \times n$ di numeri reali, dove n è il numero totale di utenti del sistema. L'elemento generico $s\,[i, j]$ della matrice contiene il valore di similarità esistente tra gli utenti u_i e u_j, calcolato attraverso l'applicazione del Coseno di Similarità.

Ogni utente u_i del sistema viene associato a dieci differenti strutture dati, utilizzate per memorizzare le informazioni relative agli item votati e ai cluster di appartenenza:

positive_item$_i$(t): la lista $positive\_item_i(t) = \{it_1^+, ..., it_k^+\}$ contiene gli item k associati a $r_{ik} = 1$ e votati all'interno dell'intervallo $t - esimo$.

positive_item$_i$: la lista $positive\_item_i = \{it_1^+, ..., it_k^+\}$ contiene gli item k associati a $r_{ik} = 1$ e votati nei $t - 1$ precedenti intervalli.

negative_item$_i$(t): la lista $negative\_item_i(t) = \{it_1^-, ..., it_k^-\}$ contiene gli item k associati a $r_{ik} = -1$ e votati all'interno dell'intervallo $t - esimo$.

negative_item$_i$: la lista $negative\_item_i = \{it_1^-, ..., it_k^-\}$ contiene gli item k associati a $r_{ik} = -1$ e votati nei $t - 1$ precedenti intervalli.

positive_cluster$_i$(t): la lista $positive\_cluster_i(t) = \{c_1^+, ..., c_k^+\}$ contiene i cluster k calcolati sull'insieme di item votati positivamente all'interno dell'intervallo $t - esimo$.

positive_cluster$_i$: la lista $positive\_cluster_i = \{c_1^+, ..., c_k^+\}$ contiene i cluster k calcolati sull'insieme di item votati positivamente nei $t-1$ precedenti intervalli.

negative_cluster$_i$(t): la lista $negative\_cluster_i(t) = \{c_1^-, ..., c_k^-\}$ contiene i cluster k calcolati sull'insieme di item votati negativamente all'interno dell'intervallo $t - esimo$.

negative_cluster$_i$: la lista $negative\_cluster_i = \{c_1^-, ..., c_k^-\}$ contiene i cluster k calcolati sull'insieme di item votati negativamente nei $t-1$ precedenti intervalli.

q$_i$(t): è una variabile reale che memorizza il valore q_i utilizzato all'interno dell'algoritmo di similarità e associato al precedente intervallo $t - 1$.

neighboorhood$_i$(t): la lista $neighboorhood_i = \{u_1, ..., u_k\}$ contiene i k utenti appartenenti al neghborhood dell'utente relativi all'intervallo $t - esimo$.

Un'altra struttura dati utilizzata all'interno dell'algoritmo è la variabile *threshold* che memorizza un valore reale rappresentante il peso minimo che un item o un cluster possono assumere prima di essere eliminati dalle rispettive liste di appartenenza.

3.5.2 Algoritmo di MinHash

L'algoritmo di clusterizzazione degli utenti basato sull'insieme delle preferenze espresse all'interno di un singolo intervallo temporale permette di ridurre la dimensione dei dati utilizzati in input nell'algoritmo di calcolo delle similarità e nel contempo consente la costruzione di un modello dinamico nel quale si tiene traccia della variazione degli interessi degli utenti attraverso l'aggiornamento nel tempo dei cluster di appartenenza.

La scelta della funzione di clustering dipende da differenti fattori, quali il dataset al quale viene applicata o la precisione dei risultati che si vogliono ottenere; in particolare nel nostro modello tale funzione influenza direttamente le prestazioni della fase di pre-processamento dei dati, che in sistemi caratterizzati da quantità di dati elevati come Twitter può diventare un collo di bottiglia dell'intero sistema. In tale ottica le funzioni di clusterizzazione basate su un approccio probabilistico rappresentano il giusto compromesso tra velocità di esecuzione e precisione dei risultati generati, fornendo un'approssimazione ammissibile degli insiemi di utenti simili.

Uno dei metodi utilizzati per eseguire la riduzione probabilistica della dimensionalità dei dati è rappresentato dal *Local Sensitive Hashing* (LSH), una famiglia di algoritmi basati sull'applicazione di una funzione di hash

agli item in input in modo da mappare con alta probabilità item simili negli stessi buckets. [74] La metodologia LSH viene principalmente utilizzata per risolvere il problema dell'*Approximate Near Neighbor*: dato un set di punti in uno spazio d-dimesionale $\mathbb{R}^d$, se esiste un punto p ad una distanza R dal punto di riferimento q (il punto p è il $R-nearest\ neighbor$), allora l'algoritmo restituisce ogni punto p' ad una distanza cR con probabilità $1-\delta$, dove c è il fattore di approssimazione e δ una costante limitata da 1.

Il funzionamento degli algoritmi LSH si basa sull'utilizzo di una famiglia di funzioni hash H definite *locality-sensitive*; l'implementazione scelta per tali funzioni, quindi, permette di realizzare differenti varianti dello stesso metodo applicabili in contesto specifici [75] [76].

Formalmente la famiglia di funzioni $(R, cR, P_1, P_2) - sensitive$ soddisfa per ogni coppia di punti $p, q \in \mathbb{R}^d$ due condizioni:

1. se $\|p-q\| \leq R$ allora $Pr_H\left[h(q) = h(p)\right] \geq P_1$
2. se $\|p-q\| \geq cR$ allora $Pr_H\left[h(q) = h(p)\right] \leq P_2$

L'algoritmo di generazione dei cluster eseguito all'interno delle estensioni Chrome da noi progettate si basa sull'algoritmo *MinHash*, una variante dell'algoritmo LSH ampiamente applicata nei sistemi di raccomandazione come metodologia per il calcolo della distanza di Jaccard tra coppie di insiemi caratterizzati da un'elevata dimensionalità.

L'approccio classico utilizzato per calcolare il coefficiente di Jaccard associato a due insiemi, infatti, è un approccio non scalabile, il cui costo computazionale dipende dalla numerosità dei dati in input e dall'efficienza delle operazioni di intersezione e di unione utilizzate.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.6}$$

Diversamente l'algoritmo MinHash calcola i coefficienti di Jaccard attraverso il confronto di valori hash associati ai singoli insiemi, realizzando così una funzione di similarità caratterizzata dalla riduzione dei dati in input e da costi computazionali indipendenti dalla dimensione degli insiemi.

L'algoritmo genera inizialmente una permutazione casuale dell'insieme degli item del sistema; successivamente associa ad ogni set di item un valore di hash calcolato come l'indice del primo elemento della permutazione che appartiene al set dell'utente.

$$h(A) = min \quad \pi(A) \tag{3.7}$$

L'efficacia dell'algoritmo di MinHash dipende dalla proprietà matematica che lo caratterizza, per la quale la probabilità di collisione degli hash di due set distinti è pari al coefficienti di Jaccard tra gli stessi; infatti la probabilità per tali insiemi di condividere lo stesso valore minhash è pari alla probabilità di selezionare all'interno della loro unione un item condiviso.

$$P(h(A) = h(B)) = P(min\pi(A) = min\pi(B)) = J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.8}$$

L'utilizzo di una sola funzione di permutazione degli elementi del sistema per generare il valore di hash di ogni set di item genera un insieme di cluster caratterizzati da un valore di precision ridotto; l'appartenenza al cluster, infatti, viene determinata solo in base alla condivisione di un singolo elemento del set di item, causando così l'aggregazione nello stesso insieme anche di set profondamente differenti.

L'algoritmo MinHash, perciò, utilizza p differenti funzioni di permutazione per aumentare l'accuratezza dei cluster; in tale modo l'hash associato al set di item di un utente viene definito attraverso la concatenazione di p differenti valori, aumentando la probabilità di

condivisione del medesimo valore di hash per insiemi simili.

$$P(h(A) = h(B)) = J(A, B)^p \tag{3.9}$$

La scelta del valore p per l'algoritmo MinHash influisce direttamente sulla dimensione dei cluster generati; valori alti, infatti, determinano la costruzione di cluster altamente precisi ma di dimensioni ridotte, costituiti da un numero limitato di set di item e quindi da valori di recall bassi.

Il miglioramento dei valori di recall associati ai cluster viene realizzato, quindi, attraverso l'esecuzione in parallelo di q algoritmi MinHash sugli stessi set di item; in tale modo ciascun set viene assegnato a q differenti cluster, definiti attraverso la concatenazione di p valori hash.

La generazione dell'insieme di permutazioni necessarie all'esecuzione dell'algoritmo MinHash e la loro memorizzazione non è realizzabile all'interno di sistemi caratterizzati da milioni di item, come Twitter. Nelle implementazioni classiche dell'algoritmo MinHash, perciò, vengono utilizzate p differenti funzioni hash per simulare l'applicazione di p funzioni di permutazione casuale degli item del sistema; il costo dell'implementazione e dell'esecuzione di tali funzioni, tuttavia, può risultare eccessivo e causare un rallentamento del tempo di esecuzione delle fasi di generazione dei coefficienti di similarità.

La soluzione adottata in alcuni sistemi, come il motore di raccomandazione implementato all'interno di Google News [77], prevede l'utilizzo di una singola funzione hash il cui comportamento viene variato attraverso l'applicazione di $p * q$ differenti valori di *seed*. Tale strategia permette di realizzare un'implementazione più efficiente dell'algoritmo, con il solo incremento di costo dovuto alla memorizzazione dei diversi valori di seed.

La funzione di hash utilizzata all'interno del nostro algoritmo è una versione del metodo di hash elaborato da Robert Jenkins nel 1997, che prevede una serie di operazioni di *shift* e di *add* eseguite su valore ottenuto tramite la concatenzazione tra l'id dell'item e il seed.

L'algoritmo MinHash implementato nella nostra architettura può essere descritto attraverso le seguenti fasi:

1. viene generato un set S di $p * q$ valori di seed casuali e uniformemente distribuiti
2. per ogni valore $s_j \in S$:
   (a) viene concatenato s_j con l'id di ogni item $i \in I_{u_i}$
   (b) viene applicata la funzione di hash ai valori concatenati $s_j \| id_i$
   (c) viene individuato il valore di hash minimo h_{min_j}
3. viene generato un insieme $H = \{h_{min_1}, ..., h_{min_j}, ...\}$ contenente i valori minimi di hash calcolati per ogni seed
4. i valori di hash $h_{min_i} \in H$ vengono raggruppati in q gruppi di p valori
5. i q gruppi di hash vengono utilizzati per identificare i cluster $cluster = \{c_1, ..., c_q\}$ di appartenenza dell'utente u_i

3.5.3 Algoritmo di similarità

L'algoritmo di similarità implementato all'interno di un Sistema di Raccomandazione ha il compito di generare i valori di similarità associati a coppie di utenti attraverso l'applicazione di una delle possibili metriche a disposizione.

Tra le varie metodologie esistenti il nostro algoritmo utilizza il coseno di similarità, che permette di definire in un sistema caratterizzato da voti positivi e negativi sia la similarità che la dissimilarità tra insiemi, attraverso il calcolo di un coefficiente compreso tra i valori -1 e 1. La funzione di similarità implementata è una funzione *user-based* che incorpora le informazioni temporali associate ai voti espressi dagli utenti attraverso l'utilizzo di una funzione di decadimento temporale.

I valori di similarità associati a coppie di utenti vengono calcolati attraverso il confronto nel tempo dei cluster di appartenenza, generati dall'algoritmo MinHash nella precedente fase di computazione.

$$S_{i,j}(t) = \frac{\sum_{c \in C_i \cap C_j} f_{c_j}^{\alpha}(t) \cdot f_{c_i}^{\alpha}(t)}{\sqrt{\sum_{c \in C_i} (f_{c_i}^{\alpha}(t))^2 \sum_{c \in C_j} (f_{c_j}^{\alpha}(t))^2}} \tag{3.10}$$

L'espressione utilizzata per definire il coseno di similarità può essere scomposta nei suoi elementi costitutivi, associati ad un differente insieme di voti e calcolabili singolarmente; è possibile, infatti, scomporre la matrice $S_{i,j}$ in una matrice $P_{i,j}$ contenente i valori calcolati sull'insieme di item covotati dagli utenti e nei valori Q_i, Q_j associati al set totale di item votati singolarmente da ogni utente.

$$S_{i,j}(t) = \frac{P_{i,j}(t)}{\sqrt{Q_i(t) \cdot Q_j(t)}} \tag{3.11}$$

L'utilizzo della funzione di decadimento esponenziale applicata ai cluster permette di costruire un valore di similarità dinamico che rappresenta l'evoluzione nel tempo degli interessi condivisi tra gli utenti. Il valore di similarità, infatti, è costituito da due differenti componenti: una componente principale, legata ai voti espressi all'interno dell'intervallo temporale più recente, che rappresenta l'insieme di interessi correnti e una componente secondaria, associata alla correlazione esistente tra gli stessi

utenti negli intervalli passati, che rappresenta l'insieme di interessi condivisi nel tempo; l'utilizzo di differenti valori di peso permette, quindi, di modulare l'importanza di ciascuna componente all'interno del valore di similarità finale.

La definizione di un algoritmo di tipo incrementale, basato sulla suddivisione del dataset in intervalli di tempo regolari e distinti, permette,così, di identificare in maniera chiara le componenti del coefficiente di similarità associate ai differenti intervalli, attraverso la decomposizione di ogni elemento della formula in diversi fattori.

$$\begin{aligned} P_{i,j}(t) = &\sum\nolimits_{x \in \Delta C_i^t \cap C_j^{t-1}} f_{x_j}^{\alpha}(t) \\ &+ \sum\nolimits_{y \in C_i^{t-1} \cap \Delta C_j^t} f_{y_i}^{\alpha}(t) \\ &+ \sum\nolimits_{z \in \Delta C_i^t \cap \Delta C_j^t} 1 \\ &+ \sum\nolimits_{k \in C_i^{t-1} \cap C_j^{t-1}} f_{k_i}^{\alpha}(t) f_{k_j}^{\alpha}(t) \end{aligned} \tag{3.12}$$

$$Q_i(t) = \sum_{x \in \Delta C_i^t} 1 + \sum_{x \in C_i^{t-1}} f_{x_i}^{\alpha}(t) \tag{3.13}$$

L'insieme ΔC_i^t rappresenta il set di cluster generati dai voti espressi all'interno dell'intervallo corrente mentre l'insieme C_i^{t-1} rappresenta i cluster generati negli intervalli precedenti; il valore di similarità associato a ciascun utente, quindi, sintetizza non solo la correlazione tra interessi all'interno dello stesso intervallo presente o passato ma anche la correlazione esistente tra intervalli distanti nel tempo.

La memorizzazione all'interno del sistema dei valori associati a $P_{i,j}(t-1)$, $Q_i(t-1)$ e $Q_j(t-1)$ relativi all'esecuzione dell'algoritmo di similarità nell'intervallo temporale precedente permette di generare efficacemente i valori di similarità relativi all'intervallo corrente, attraverso il calcolo dei soli fattori di variazione Δ relativi ai nuovi cluster generati nel sistema;

tali fattori, rappresentati nella formula precedente dalle prime tre somme, possono essere sintetizzati in un'unica espressione, in modo da evidenziare il contributo associato ai nuovi cluster e il contributo calcolato nella fase precedente di computazione.

$$P_{i,j}(t) = \Delta P_{i,j}(t) + e^{-2\alpha} \cdot P_{i,j}(t-1) \tag{3.14}$$

$$Q_i(t) = \Delta Q_i(t) + e^{-2\alpha} \cdot Q_i(t-1) \tag{3.15}$$

L'utilizzo di una strategia incrementale per la costruzione della matrice di similarità consente di ridurre la complessità temporale dell'algoritmo. In un classico approccio non incrementale, infatti, la matrice $P_{i,j}$ viene generata ad ogni intervallo temporale utilizzando l'insieme di tutti i cluster generati nel sistema fino all'istante corrente, con un costo computazionale pari a $O(\left|C_i^t \cap C_j^t\right|)$. Diversamente l'approccio incrementale si limita a calcolare in ogni intervallo il solo fattore di variazione Δ legato ai nuovi cluster, con un costo limitato da $O(|\Delta C_i^t| + \left|\Delta C_j^t\right|)$.

L'algoritmo di similarità implementato all'interno della nostra architettura viene strutturato in tre differenti fasi:

Fase di aggiornamento dei pesi dei cluster: in tale fase il peso dei cluster generati negli intervalli precedenti viene aggiornato attraverso l'applicazione del fattore di decadimento temporale $e^{-\alpha t}$; i cluster associati ad un peso inferiore al valore di threshold impostato per il sistema vengono rimossi dalle liste di appartenenza.

1. per ogni $x \in C_i^{t-1}$, si ha $w_{xi}(t) = w_{xi}(t-1) * e^{-\alpha}$
2. se $w_{xi}(t) \leq threshold$ allora $C_i^{t-1} = C_i^{t-1} \backslash \{x\}$

Fase di calcolo della similarità: in tale fase i set dei cluster di appartenenza degli utenti vengono confrontati a coppie per generare la matrice di similarità; nella sua formulazione più semplice tale fase si basa sul confronto dei cluster generati dai soli voti positivi espressi dagli utenti.

3. per ogni coppia i, j di utenti si prelevano i valori:
   (a) $\Delta C_i^t, C_i^{t-1}$: cluster dell'utente i associati all'intervallo corrente t e agli intervalli precedenti $t-1$
   (b) $\Delta C_j^t, C_j^{t-1}$: cluster dell'utente j associati all'intervallo corrente t e agli intervalli precedenti $t-1$
   (c) $P_{i,j}(t-1)$: valore di $P_{i,j}$ calcolato nell'intervallo precedente $t-1$
   (d) $Q_i(t-1)$: valore $Q_i(t)$ calcolato nell'intervallo precedente $t-1$
   (e) $Q_j(t-1)$: valore $Q_j(t)$ calcolato nell'intervallo precedente $t-1$
4. vengono calcolate le intersezioni tra gli insiemi di cluster correnti e gli insiemi di cluster precedenti:
   (a) $\Delta C_i^t \cap C_j^{t-1}$
   (b) $C_i^{t-1} \cap \Delta C_j^t$
   (c) $\Delta C_i^t \cap \Delta C_j^t$
5. vengono calcolati i contributi forniti al valore di similarità finale dai cluster presenti nelle tre differenti intersezioni:
   (a) per ogni $x \in \Delta C_i^t \cap C_j^{t-1}$, $a = \sum_x (w_{xi}(t) \cdot w_{xj}(t))$
   (b) per ogni $y \in C_i^{t-1} \cap \Delta C_j^t$, $b = \sum_y (w_{yi}(t) \cdot w_{yj}(t))$

(c) per ogni $z \in \Delta C_i^t \cap \Delta C_j^t$, $c = \sum_z (w_{zi}(t) \cdot w_{zj}(t))$

6. viene calcolato il valore $P_{i,j}(t)$:
   (a) $P_{i,j}(t) = a + b + c + e^{-2\alpha} P_{i,j}(t-1)$
7. per ogni coppia i, j di utenti vengono calcolati i nuovi valori $Q_i(t), Q_j(t)$:
   (a) $Q_i(t) = \sum_{x \in \Delta C_i^t} w_{xi}(t) + e^{-2\alpha} Q_i(t-1)$
   (b) $Q_j(t) = \sum_{x \in \Delta C_j^t} w_{xj}(t) + e^{-2\alpha} Q_j(t-1)$
8. viene calcolato il nuovo valore di similarità per la coppia di utenti i, j:
   (a) $S_{i,j}(t) = \frac{P_{i,j}(t)}{\sqrt{Q_i(t) \cdot Q_j(t)}}$

Fase di aggiornamento dei cluster: in tale fase i cluster associati all'intervallo corrente vengono inseriti all'interno dell'insieme contenente tutti i cluster di appartenenza dell'utente nel tempo; il peso iniziale associato ai nuovi cluster è pari a 1.

9. per ogni $i \in User$ e per ogni $c \in \Delta C_i^t$:
   (a) $w_{ci}(t) = 1$
   (b) $C_i^{t-1} = C_i^{t-1} \cup \{c\}$

3.5.4 Algoritmo di costruzione del neighborhood incrementale

L'algoritmo di costruzione del neighborhood degli utenti è basato su un approccio di tipo incrementale, che prevede l'aggiornamento dell'insieme generato nell'intervallo di esecuzione precedente attraverso l'utilizzo delle nuove similarità calcolate all'interno dell'intervallo corrente. La lista degli

utenti *k-nearest neighbors* associati ai valori di similarità più elevati rispetto all'utente target viene generata attraverso l'applicazione di un algoritmo di ordinamento ai valori contenuti nella matrice di similarità; per ciascun utente, perciò, vengono selezionati gli utenti contenuti nelle prime k posizioni della lista ordinata.

L'approccio incrementale si basa sull'osservazione che la variazione del *neighborhood* di un utente tra un intervallo temporale e l'altro è minima; risulta possibile, quindi, utilizzare come punto di partenza per la costruzione dell'insieme di utenti simili associato ad uno specifico intervallo l'insieme generato nell'iterazione precedente del'algoritmo, riducendo così parte del costo computazionale dell'intera operazione.

Gli algoritmi di ordinamento utilizzati all'interno del nostro sistema, perciò, vengono strutturati in modo da tener conto dell'esistenza dell'insieme precedente di k utenti simili, i cui valori di similarità vengono aggiornati attraverso l'analisi dei voti espressi nell'intervallo corrente.

Il primo passo dell'algoritmo di costruzione del neighborhood consiste nell'applicazione di un algoritmo di ordinamento ai k valori presenti nel neighborhood di ciascun utente in modo da garantire l'ordinamento rispetto ai nuovi valori di similarità calcolati nell'intervallo corrente. Successivamente l'algoritmo effettua una scansione dei restanti $n - k$ valori per individuare eventuali utenti caratterizzati da elevati valori di similarità da inserire all'interno del nuovo neighborhood; tale fase di ricerca viene eseguita attraverso il confronto degli $n - k$ valori con il valore minimo presente nel neighborhood, che rappresenta il fattore discriminante per l'inserimento di un elemento all'interno del neighborhood stesso.

L'algoritmo formalmente è costituito dalle seguenti fasi:

1. per ogni utente $i \in U$ viene costruito l'insieme $S_i = \bigcup_{j \in U} S(i, j)(t)$

2. se $neighborhood_i(t-1)$ non esiste:
   (a) viene calcolato $quickselect(S_i(t))$
   (b) viene costruito $neighborhood_i(t) = \bigcup_{j \in S_i(t)} u_j$, con $1 \leq j \leq k$
3. se $neighborhood_i(t-1)$ esiste:
   (a) viene eseguito $quicksort(neighborhood_i(t-1))$
   (b) per ogni utente $j \in S_i(t)$ con $k+1 \leq j \leq n$ viene eseguito $insertionsort(neighborhood_i(t-1), j)$ se e solo se $s(i,j) \geq \min(neighborhood_i(t-1))$
4. viene generato $neighborhood_i(t) = \bigcup_{j \in S_i(t)} u_j$, con $1 \leq j \leq k$

Rispetto agli algoritmi classici di costruzione del neighborhood degli utenti, che prevedono la scansione dell'intera matrice di similarità per la selezione degli utenti k più simili, l'approccio incrementale ha il vantaggio di essere associato ad un costo computazionale inferiore. La complessità temporale dell'operazione del *QuickSelect* eseguita sull'insieme di tutti gli utenti ha un costo, infatti, pari a $O(n + k\log(k))$, che in un approccio non incrementale viene pagato ad ogni esecuzione dell'algoritmo.

Diversamente l'approccio incrementale trae vantaggio dall'osservazione che il neighborhood di un utente non varia sensibilmente tra un'intervallo temporale e l'altro; le operazioni di ordinamento, quindi, operano su un set di valori che resta parzialmente ordinato, con una complessità temporale inferiore al costo asintotico medio.

L'operazione di *QuickSort*, infatti, viene eseguita sull'insieme dei k utenti generato nell'intervallo precedente per garantire l'ordinamento rispetto alle nuove similarità, con un costo generalmente inferiore a $O(k\log(k))$.

L'operazione di *InsertionSort*, invece, viene eseguita sui restanti $n - k$ utenti del sistema per individuare gli utenti associati ad un valore di similarità almeno superiore al valore minimo contenuto nel neighborhood; l'esecuzione dell'operazione di inserimento solo per valori superiori al valore minimo del neighborhood permette di ottenere un costo computazionale ridotto rispetto all'esecuzione nel caso peggiore.

3.5.5 Algoritmo di Raccomandazione

L'algoritmo di raccomandazione implementato all'interno della nostra architettura ha l'obiettivo di costruire una lista di item personalizzata per ciascun utente contente i tweet non ancora visualizzati ma associati ad un'alta probabilità di gradimento.

L'approccio utilizzato all'interno dell'algoritmo di raccomandazione si basa sulla ricerca degli item di interesse di uno specifico utente all'interno del set di item votati dagli utenti appartenenti al suo neighborhood; gli alti valori di similarità dell'utente target con i neighbors, infatti, assicurano l'esistenza di un sottoinsieme di preferenze condivise nel tempo e permettono di assegnare con alta precisione un livello di gradimento elevato per item considerati interessanti dalla maggior parte dei neighbors.

L'insieme degli item valutati per l'inserimento nella lista di raccomandazione, perciò, viene costruito selezionando i soli item votati positivamente dai neighbors dell'utente ma non ancora votati dallo stesso; tali item vengono organizzati in una lista ordinata, nella quale il fattore di ordinamento è rappresentato da un valore di score dipendente da diversi elementi, quali il numero di voti positivi ricevuti dai neighbors e il valore di

similarità con l'utente target:

$$\hat{r_{ix}}(t) = \sum_{j \in N_j^t} s_{ij}(t) \cdot w_{jx}(t) \cdot r_{jx} \tag{3.16}$$

La lista di raccomandazione, quindi, viene costruita selezionando l'insieme di k item associati ai valori di score maggiore.

L'algoritmo di raccomandazione viene formalmente descritto dalle seguenti fasi:

1. per ogni utente i vengono prelevati gli utenti $j \in neighborhood_i(t)$
2. per ogni utente $j \in neighborhood_i(t)$ viene costruita una lista $item_i = \bigcup_{x \in r_{jx}^+} x$ contenente gli item associati a voti positivi
3. per ogni item $x \in item_i$ viene calcolato $score(x) = \sum_{j \in neighborhood_i(t)} w_{jx}(t) r_{jx}$
4. viene costruita la lista $score_i = \bigcup_x score(x)$
5. viene eseguito $quickselect(score_i)$
6. viene costruita la lista di raccomandazione $list_i = \bigcup_{x \in score_i} x$ con $1 \leq x \leq k$

L'utilizzo della similarità tra utenti come fattore di peso per i voti associati agli item permette di assegnare a ciascun voto un livello di importanza differente legato alla correlazione esistente tra gli utenti; i voti espressi da utenti con similarità più alta, quindi, vengono associati ad un coefficiente maggiore, dal momento che hanno una probabilità elevata di rappresentare gli item appartenenti all'insieme di interessi condivisi con l'utente target. Il peso associato ad ogni item, invece, viene utilizzato per generare un insieme di raccomandazioni legate agli interessi correnti

dell'utente. In un sistema dinamico come Twitter, infatti, il tempo di vita di ciascuna informazione, inteso come durata dell'importanza della notizia associata, è estremamente breve e ogni utente è interessato a ricevere come raccomandazione un insieme di messaggi legati ad accadimenti recenti. L'assegnazione di un coefficiente di peso temporale ai voti, quindi, permette di costruire un valore di score per ciascun item che dipende maggiormente dall'insieme di voti ricevuti nel recente passato, limitando o eliminando il contributo fornito da voti ormai non più significativi.

Capitolo 4

Valutazione sperimentale

La progettazione e la realizzazione del nuovo sistema di raccomandazione per il social network Twitter rappresenta solo una delle fasi di lavoro svolte nell'ambito di questo lavoro di tesi. Particolare attenzione, infatti, è stata posta alla successiva fase di valutazione sperimentale, necessaria per identificare eventuali problemi nell'architettura e nel codice e per stabilire l'efficacia dell'algoritmo di raccomandazione implementato rispetto ad algoritmi realmente esistenti.

La prima parte della fase di valutazione del sistema ha riguardato la verifica delle funzionalità implementate all'interno dell'estensione di Chrome e la ricerca di eventuali bug del codice. Durante tale fase, quindi, l'estensione TweetRate è stata rilasciata ad un set ridotto di utenti di test, i quali hanno utilizzato le funzionalità di voto per un periodo di un mese.

I feedback ricevuti da tali utenti relativi alla loro User Experience sono stati utilizzati per individuare gli aspetti visuali dell'interfaccia da modificare e creare, così, un'applicazione di facile utilizzo ed intuitiva; l'analisi dei dati raccolti nel database remoto, invece, ha permesso di verificare la corretta acquisizione delle informazioni necessarie agli algoritmi di raccomandazione

e di individuare eventuali dati superflui la cui presenza aumenta i costi di memorizzazione senza apportare alcun beneficio.

La seconda parte della fase di valutazione del sistema ha riguardato l'analisi dell'efficacia dell'algoritmo di raccomandazione implementato, con particolare attenzione ai contributi forniti alla predizione finale dai singoli elementi caratteristici utilizzati all'interno dell'algoritmo. La precisione dei risultati ottenuti in tale processo di valutazione dipende strettamente dalla tipologia e dalla quantità di dati utilizzati come input del sistema di raccomandazione; l'utilizzo di un dataset esteso, infatti, permette di superare problematiche legate alla sparsità dei dati e limitare il fattore di rumore introdotto nelle predizioni da un dataset di dimensioni ridotte in cui non siano chiaramente individuabili le correlazioni tra i diversi utenti.

L'insieme dei dati raccolti durante la fase di test dell'estensione Chrome si è rivelato inadeguato all'esecuzione dell'algoritmo di raccomandazione: il numero ridotto di utenti del sistema e di voti espressi, infatti, non ha permesso la costruzione di una matrice di similarità che permettesse di stabilire relazioni significative tra gli utenti. É stato necessario, quindi, individuare un dataset tra quelli distribuiti liberamente che si adattasse agli scopi del nostro lavoro di tesi, caratterizzato da un insieme di voti facilmente riconducibili ad un insieme binario e associati ad un timestamp che ne permettesse l'ordinamento.

4.1 Dataset

Molti sistemi informativi esistenti rilasciano i propri dataset per l'utilizzo nei lavori di ricerca connessi all'information filtering o all'elaborazione di nuovi algoritmi di raccomandazione; ciascuno di questi dataset è associato

a differenti caratteristiche dei dati, che dipendono essenzialmente dalla tipologia di sistema nel quale sono stati raccolti.

Il dataset scelto per l'utilizzo all'interno del nostro sistema è il dataset rilasciato dal sistema di raccomandazione online Movielens, che rappresenta uno degli insiemi di dati più utilizzati per il test delle performance degli algoritmi di predizione.

Movielens è un sistema online di raccomandazione per film, nel quale ciascun utente può esprimere per i film visti un voto compreso tra 1 e 5 stelle. L'insieme delle raccomandazioni viene generato attraverso l'applicazione di un algoritmo di collaborative filtering che analizza i voti espressi dall'utente target e dagli utenti a lui simili, generando una predizione di voto per i film non ancora visualizzati.

I dati raccolti dal sistema Movielens si prestano all'applicazione dell'algoritmo di raccomandazione sviluppato per Twitter grazie alle analogie esistenti tra i due sistemi: entrambi, infatti, sono caratterizzati da un insieme di utenti la cui dimensione è inferiore rispetto all'insieme di item votabili, rendendo possibile l'applicazione di strategie di tipo user-based, nelle quali l'elemento centrale è il neighborhood di ciascun utente; inoltre i voti espressi all'interno di Movielens vengono associati ad un timestamp che ne permette l'ordinamento, così come accade in Twitter attraverso l'utilizzo dell'estensione TweetRate.

Tale dataset, perciò, richiede un minimo sforzo computazionale per adattarlo ai nostri scopi.

4.1.1 Movielens 100k: descrizione del dataset

Movielens 100k è un insieme di dati raccolti dal *GroupLens Research Project* dell'Università del Minnesota in un periodo di sette mesi compreso tra il 19

Settembre 1997 e il 22 Aprile 1998.

Il dataset è caratterizzato da un insieme di 100000 voti compresi tra 1 e 5 espressi da 943 utenti su 1682 differenti film; i dati sono stati inizialmente corretti dai ricercatori rimuovendo gli utenti con un numero di voti inferiore a 20 o associati ad informazioni demografiche incomplete.

L'intero dataset viene strutturato in sei differenti file:

- u.data: contiene l'insieme di 100000 voti ordinati in maniera casuale e organizzati in una tabella caratterizzata dai campi:

$$user\ id \mid item\ id \mid rating \mid timestamp$$

- u.item: contiene le informazioni associate ai film organizzate in una tabella caratterizzata dai campi:

$$movie\ id \mid movie\ title \mid release\ date \mid video\ release\ date \mid IMDb\ URL \mid unknown \mid Action \mid Adventure \mid Animation \mid Children's \mid Comedy \mid Crime \mid Documentary \mid Drama \mid Fantasy \mid Film-Noir \mid Horror \mid Musical \mid Mystery \mid Romance \mid Sci-Fi \mid Thriller \mid War \mid Western \mid$$

  I campi associati al genere dei film sono campi di tipo binario, nei quali il valore 1 indica il genere di appartenenza.

- u.user: contiene le informazioni demografiche associate agli utenti organizzate in una tabella caratterizzata dai campi:

$$user\ id \mid age \mid gender \mid occupation \mid zip\ code$$

- u.info: contiene informazioni sul numero di utenti, di item e di voti.

- u.genre : contiene una lista dei generi dei film.
- u.occupation: contiene una lista delle occupazioni degli utenti.

4.1.2 Operazioni preliminari sui dati

Nella fase preliminare di analisi del dataset sono state individuate alcune caratteristiche dei dati non compatibili con i requisiti di funzionamento del nostro algoritmo:

- Range dei voti: i voti esprimibili da ciascun utente sugli item del sistema Movielens vanno da un voto minimo pari a 1 ad un voto massimo pari a 5; il nostro algoritmo di raccomandazione, tuttavia, prevede l'utilizzo di un voto di tipo binario che mappi esclusivamente gli stati mi piace (1) e non mi piace (-1)
- Strutturazione del dataset: il dataset rilasciato dal GroupLens Research Project memorizza l'insieme dei voti secondo un ordine casuale, non collegato al timestamp delle preferenze; il nostro algoritmo di raccomandazione, tuttavia, lavora su un insieme di dati ordinato cronologicamente in modo da evidenziare l'andamento nel tempo delle correlazioni tra utenti

Il dataset, quindi, è stato sottoposto ad una fase di pre-processamento dei dati allo scopo di modificare tali caratteristiche ed adattarle agli algoritmi da noi sviluppati.

La prima operazione effettuata ha riguardato la correzione del range dei voti attraverso una mappatura del set di preferenze sull'insieme [-1,1]. L'algoritmo utilizzato per realizzare tale operazione opera secondo le seguenti fasi:

1. *Calcolo del voto medio di ciascun utente*: in questa fase viene determinato per ciascun utente il voto medio espresso sull'intero dataset delle preferenze; tale operazione viene effettuata per determinare il voto limite che separa le preferenze positive da quelle negative, diverso per ciascun utente a causa delle differenti abitudini di voto.

2. *Mappatura dei voti sull'insieme binario*: in questa fase il valore del voto medio di ciascun utente viene confrontato con il set di voti espressi all'interno del sistema, in modo da mappare ciascuna preferenza sul valore binario 1 se maggiore del voto medio o sul valore -1 in caso contrario.

La seconda operazione effettuata sul dataset di Movielens ha riguardato l'ordinamento dei dati rispetto al timestamp associato; i dati contenuti nel dataset originale, infatti, sono organizzati secondo un ordine casuale che non tiene conto dell'istante di generazione del voto.

La fase di ordinamento genera, quindi, un insieme di dati totalmente ordinato; la struttura interna del nuovo dataset permette, quindi, al nostro algoritmo di costruire una matrice di similarità i cui valori dipendono dall'evoluzione nel tempo delle correlazioni tra gli utenti.

Il dataset generato dalla fase di pre-processamento dei dati viene suddiviso in un set di file, ciascuno contenente l'insieme di voti associati ad uno specifico intervallo temporale. L'ampiezza di tali intervalli dipende sia dalla natura del dataset utilizzato che dalla tipologia di correlazione tra utenti che si vuole analizzare: dimensioni ridotte, infatti, permettono di evidenziare l'andamento a breve termine delle similarità tra utenti ma rappresentano un fattore limitante per quei dataset nei quali gli utenti hanno una frequenza di voto molto bassa.

Nel nostro scenario, quindi, si è scelto di utilizzare una dimensione degli intervalli pari a 30 giorni, in modo da avere a disposizione all'interno di ciascun intervallo un numero sufficiente di preferenze per modellare correttamente l'insieme di interessi degli utenti.

4.1.3 Analisi del dataset

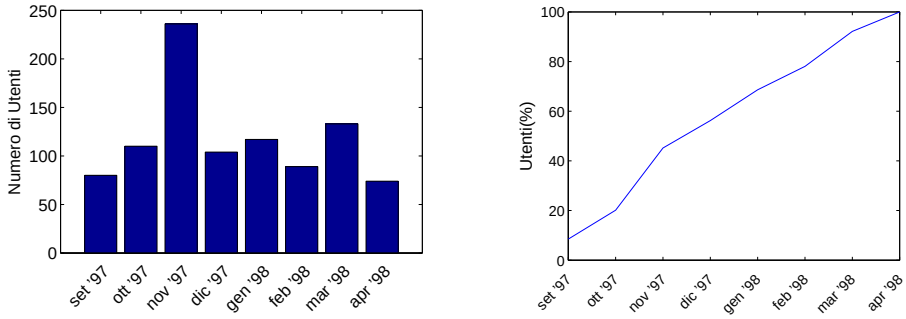
La fase di pre-processamento dei dati genera un dataset la cui struttura simula quella di un dataset reale ottenuto attraverso l'utilizzo dell'estensione TweetRate; l'analisi preliminare effettuata su tali dati permette di evidenziare alcune caratteristiche del sistema, come la distribuzione degli utenti e degli item *attivi* nel tempo o le abitudini di voto degli utenti.

I primi dati ottenuti sono stati utilizzati per determinare l'evoluzione temporale della struttura del sistema di raccomandazione utilizzato nella fase di test. In particolare è stato tracciato un grafico il cui scopo è quello di mostrare il cambiamento nel tempo della dimensione del sistema a seguito dell'ingresso di nuovi utenti o di nuovi item; in tale scenario, perciò, il timestamp associato al primo voto espresso da ciascun utente o al primo voto ricevuto da ciascun item è stato utilizzato per determinare l'istante di ingresso di nuovi elementi nel sistema.

Nella Figura 4.1a è possibile osservare come il numero di nuovi utenti che entrano nel sistema in ogni intervallo di osservazione sia tendenzialmente uniforme, ad eccezione del picco relativo al mese di Novembre; questa caratteristica si riflette in una crescita costante della dimensione del sistema di raccomandazione in termini di numero di utenti attivi, come evidenziato dal grafico in Figura 4.1b. Il dataset, perciò, riesce a rappresentare una porzione limitata di funzionamento di un sistema reale, nel quale l'insieme

degli utenti attivi non è una entità costante ma varia periodicamente a seguito dell'ingresso di nuovi individui.

La variabilità dell'insieme di utenti monitorati e la sua crescita nel tempo si riflettono nel nostro sistema di raccomandazione in una crescita costante della matrice di similarità tra gli utenti, la cui dimensione dipende direttamente dal numero di utenti che hanno espresso almeno un voto nell'istante di creazione della matrice.

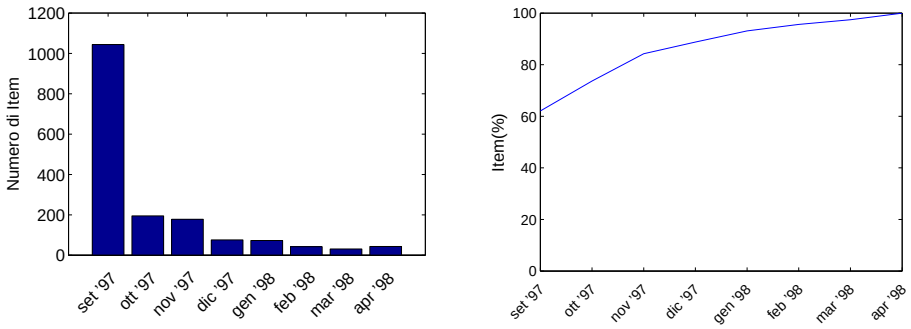


(a) Nuovi utenti per mese (b) Crescita utenti

Figura 4.1: Andamento del numero di utenti nel tempo

Diversamente la Figura 4.2a rappresenta l'andamento del numero di nuovi item nel tempo; in particolare il grafico evidenzia come circa il 60% degli item totali contenuti nel dataset sia disponibile e quindi votabile a partire dal primo mese di raccolta dei dati; questo aspetto permette di avere all'interno del dataset un sottoinsieme consistente di item covotati in ogni intervallo temporale considerato, andando a costruire una base di conoscenza ideale per l'individuazione delle correlazioni esistenti tra la maggior parte degli utenti attivi. La Figura 4.2b, inoltre, evidenzia l'andamento nel tempo della crescita dell'insieme degli item ; i dati a disposizione permettono di osservare come la dimensione del set di item del sistema sia caratterizzata da un rate

di crescita inferiore rispetto a quello del set di utenti, generando così minime variazioni nella dimensione delle strutture dati associate agli item.



(a) Nuovi item per mese (b) Crescità item

Figura 4.2: Andamento del numero di item nel tempo

I successivi dati ottenuti dall'analisi della struttura del dataset sono stati utilizzati per identificare le variazioni tra le abitudini di voto dei singoli utenti in termini di numero di preferenze espresse durante il periodo di osservazione del sistema.

Inizialmente l'intero set di utenti del sistema è stato suddiviso in classi di frequenza, ciascuna contenente il sottoinsieme di individui che hanno espresso un numero di voti compreso negli estremi che identificano la classe stessa; successivamente le frequenze ottenute sono state utilizzate per determinare le frequenze cumulate e le densità di frequenza di ciascuna classe di voto.

La Figura 4.3 evidenzia l'andamento delle frequenze cumulate rilevate sul dataset; l'analisi del grafico permette di osservare come circa il 50% del totale degli utenti abbia espresso un numero di voti non superiori a 40 e come circa il 75% degli utenti abbia espresso un numero di voti non superiore a 70, andando così a coprire solo il 5% degli item presenti nel sistema.

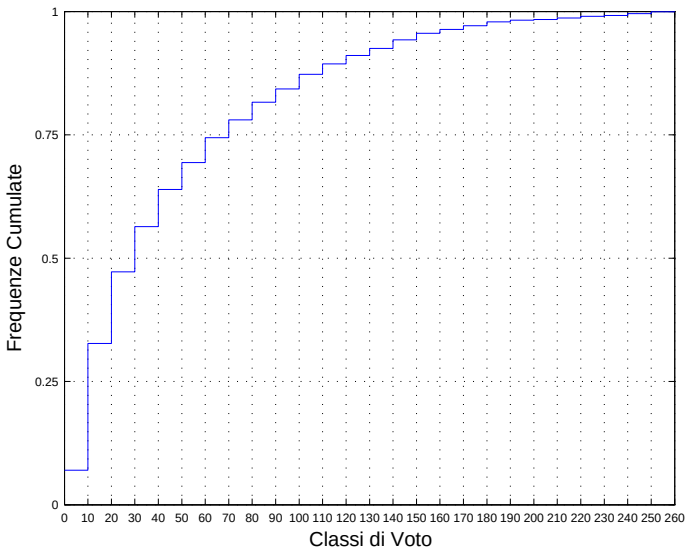
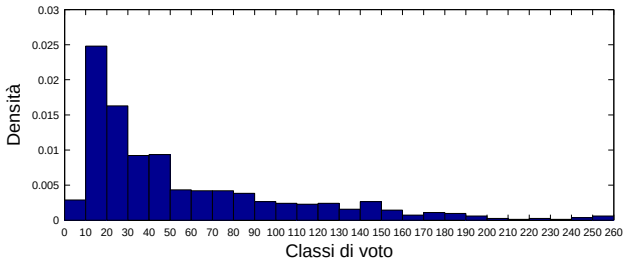


Figura 4.3: Distribuzione di frequenza totale dei voti

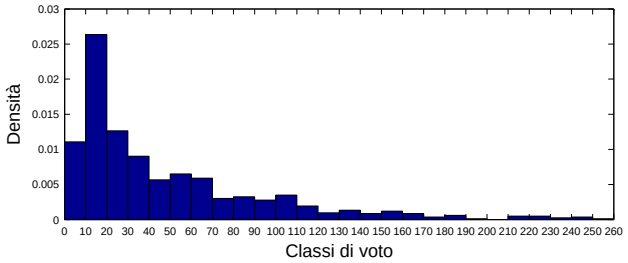
Le Figure 4.4a e 4.4b evidenziano,invece, le densità associate alle differenti classi di voto sia per l'insieme di preferenze positive che per l'insieme di preferenze negative; tali dati confermano la tendenza tipica degli utenti presenti nel dataset ad esprimere un numero di voti limitato e compreso tra le 10 e le 30 preferenze. Questo andamento permette di ipotizzare la presenza di un sottoinsieme consistente di utenti scarsamente attivo nel dataset, che si limita a visualizzare e votare una porzione ridotta degli item a disposizione.

L'analisi temporale effettuata sul dataset è stata orientata all'individuazione del corretto dimensionamento degli intervalli da utilizzare all'interno dell'agoritmo di raccomandazione; in particolare si è cercato di stabilire quale ampiezza degli intervalli garantisse la distribuzione più uniforme dell'insieme delle preferenze, evitando di generare periodi caratterizzati da alta variabilità.

Le tipologie di intervallo considerate nell'analisi del dataset sono quelle



(a) Voti Positivi



(b) Voti Negativi

Figura 4.4: Distribuzione di frequenza dei voti del dataset

legate a periodi di 1, 7 e 30 giorni; per ciascuna tipologia sono stati inizialmente calcolati il numero di voti complessivi (positivi e negativi) espressi dagli utenti attivi negli intervalli e successivamente è stata determinata la deviazione standard di tali valori.

Vista la differente natura delle distribuzioni e del valore medio associato è stato necessario calcolare il coefficiente di variazione, definito a partire dalla deviazione standard σ e dalla media μ come

$$CV = \left(\frac{\sigma}{|\mu|}\right) \times 100 \tag{4.1}$$

che permette di confrontare la dispersione di dati aventi range di variazione diversi.

La Tabella 4.1 evidenzia come la distribuzione associata ad intervalli di

30 giorni sia caratterizzata da una variazione minore, rendendo quindi tale ampiezza ideale per l'esecuzione della fase di test dell'algoritmo.

	1 giorno	7 giorni	30 giorni
Coefficiente di variazione	98.95%	58.73%	22.31%

Tabella 4.1: Coefficiente di variazione per i diversi intervalli temporali

4.2 Metriche di interesse

L'insieme dei test effettuati sul nostro sistema di raccomandazione è orientato al raggiungimento di due differenti obiettivi:

Valutazione delle raccomandazioni generate: il sistema di raccomandazione viene utilizzato per generare una lista di item non ancora visualizzati da raccomandare a ciascun utente; successivamente viene valutata la precisione delle raccomandazioni generate.

Valutazione delle predizioni di voto generate: il sistema di raccomandazione viene utilizzato per generare una predizione di voto per item già visualizzati e votati dagli utenti; successivamente viene valutata la precisione della predizione di voto.

Per poter valutare tali scenari applicativi è necessario stabilire l'insieme di misure di valutazione utilizzate, che variano a seconda della tipologia di utilizzo del sistema di raccomandazione e dell'utilizzo o meno di fattori temporali.

4.2.1 Sistemi time-free

I sistemi di raccomandazione time-free utilizzano l' insieme di metriche classiche per la valutazione dell'efficacia degli algoritmi applicati; tali metriche vengono suddivise in diversi macro-insiemi, ciascuno relativo ad uno specifico contesto di funzionamento del sistema [78] [79] [80] [9] [18] [5] [81].

Sistema di raccomandazione

Il sistema di raccomandazione ha come scopo principale la generazione di una lista personalizzata di item *top-k* da raccomandare a ciascun utente; tali item rappresentano entità inserite nel sistema non ancora visualizzate dall'utente attivo ma associate ad un'alta probbailità di gradimento. Le metriche di valutazione adottate in tale contesto, perciò, cercano di stimare l'accuratezza delle liste generate in termini di item di interesse presenti.

La prima metrica di valutazione utilizzata è la *Precision* che misura la frazione di item rilevanti individuati dall'algoritmo rispetto all'insieme totale di item restituiti.

La Precision viene definita come:

$$Precision = \frac{I_R(u) \cap I_K(u)}{|I_K(u)|} \tag{4.2}$$

dove $I_R(u)$ e $I_K(u)$ sono rispettivamente gli insiemi degli item rilevanti per l'utente u e degli item restituiti nella lista di raccomandazione.

La successiva metrica di valutazione utilizzata è la *Recall* che misura la frazione di item rilevanti individuati dall'algoritmo rispetto all'insieme totale di item rilevanti presenti nel sistema di raccomandazione.

La Recall viene definita come:

$$Recall = \frac{I_R(u) \cap I_K(u)}{|I_R(u)|} \tag{4.3}$$

dove $I_R(u)$ e $I_K(u)$ sono rispettivamente gli insiemi degli item rilevanti per l'utente u e degli item restituiti nella lista di raccomandazione.

La Precision e la Recall sono due misure di valutazione il cui comportamento su uno stesso sistema è inversamente correlato: l'aumento, infatti, di uno dei due valori attraverso uno specifico tuning dei parametri del sistema di raccomandazione si riflette nella corrispondente diminuzione dell'altro valore. L'utilizzo, perciò, dei valori di Precision e di Recall per

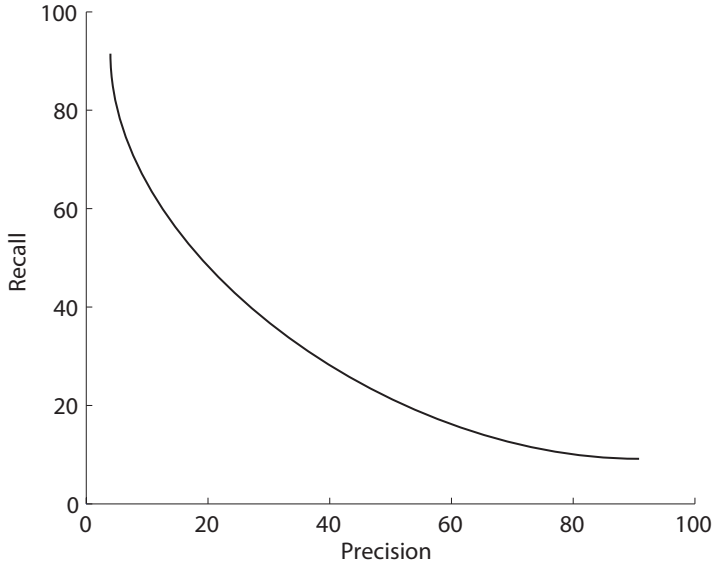


Figura 4.5: Grafico precision/recall

il confronto tra algoritmi di raccomandazione differenti genera una serie di risultati non univocamente interpretabili; gli algoritmi caratterizzati da valori di Precision più alti, infatti, risultano, rispetto agli altri algoritmi, penalizzati sul piano dei valori di Recall.

Una misura di valutazione adottata per evitare tale limitazione e fornire un valore significativo per il confronto tra gli algoritmi è la metrica *F1-score*,

che rappresenta una media armonica tra i valori di Precision e di Recall associati al medesimo sistema.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.4}$$

Sistema di predizione

Il sistema di predizione è un sistema che cerca di assegnare agli item non ancora visualizzati dagli utenti un voto più vicino possibile al voto reale espresso in futuro; i risultati ottenuti da tali sistemi, perciò, vengono valutati in relazione all'errore commesso nella procedura di predizione.

La metrica più popolare utilizzata per valutare l'accuratezza delle predizioni è il *Root Mean Squared Error*(RMSE), che stima l'errore commesso attraverso il confronto tra l'insieme dei voti previsti e l'insieme dei voti reali.

Il RMSE viene definito come:

$$RMSE = \sqrt{\frac{\sum_{i \in I} (\hat{r}_{ui} - r_{ui})^2}{|I|}} \tag{4.5}$$

dove I è l'insieme di item del sistema per i quali sono noti i voti espressi dagli utenti e $\hat{r}_{ui}$ e e r_{ui} sono rispettivamente il voto previsto e il voto reale dell'item i-esimo per l'utente u. La metrica RMSE è adatta all'utilizzo nei sistemi di predizione in quanto misura l'errore compiuto su tutti i voti del sistema; tuttavia tale metrica non riesce ad evidenziare la differenza tra gli errori commessi nei sistemi caratterizzati da uno spazio dei voti non uniforme, nei quali la previsione di un voto positivo per item con voto reale negativo genera un errore di peso maggiore rispetto alla previsione di un voto positivo per item realmente apprezzati dall'utente.

Un'altra metrica utilizzata per stimare l'errore commesso nel processo di predizione è il *Mean Absolute Error* (MAE) che, a differenza del RMSE, non penalizza maggiormente grandi errori.

Il MAE viene definito come:

$$MAE = \frac{\sum_{i \in I}(\hat{r}_{ui} - r_{ui})}{|I|} \tag{4.6}$$

Una metrica utilizzata per valutare l'efficacia del sistema di raccomandazione per gli utenti è la *Coverage*, che cerca di stimare la quantità di item per la quale il sistema è in grado di generare una predizione. Gli algoritmi di raccomandazione, infatti, non riescono a fornire una predizione di voto per ogni item presente nel sistema, a causa della sparsità dei dati della matrice dei voti; il valore di Coverage, perciò, fornisce una stima della quantità di item per i quali gli algoritmi di predizione possono essere applicati. In particolare alti valori di Coverage sono associati a sistemi in grado di aiutare gli utenti a selzionare item di interesse, mentre bassi valori di Coverage sono associati a sistemi nei quali gli utenti devono affidarsi ad altri criteri per la selezione degli item.

La Coverage viene definita come:

$$Coverage = \frac{\sum_{i=1}^{m} np_i}{\sum_{i=1}^{m} n_i} \tag{4.7}$$

dove n_i e np_i sono rispettivamente il numero di item per i quali l'utente u_i ha espresso un voto e il numero di questi item per i quali il sistema può fornire una predizione.

Sistema di classificazione binario

Un sistema di classificazione binario è una variante dei sistemi di raccomandazione classici; diversamente da questi ultimi, che cercano di

stabilire una gerarchia di livelli di gradimento per i differenti item, i sistemi di classificazione utilizzano una scala di voto binaria che permette di suddividere l'intero set di item in due insiemi distinti, rappresentanti gli item di gradimento e non per lo specifico utente [82].

Le metriche di valutazione utilizzate in tali sistemi si basano su valori dedotti a partire dalla *Matrice di Confusione*:

		Actual		
		p	n	Total
Predicted	p'	true positive (TP)	false positive (FP)	P
	n'	false negative (FN)	true negative (TN)	N
	total	P'	N'	

Tabella 4.2: Matrice di Confusione

La *Precision* del sistema di classificazione rappresenta la percentuale di voti previsti come positivi che realmente lo sono:

$$Precision = \frac{TP}{TP + FP} \tag{4.8}$$

La *Recall* del sistema di classificazione, chiamata anche *Sensitivity*, rappresenta la percentuale di voti positivi previsti come tali:

$$Recall \quad o \quad Sensitivity = \frac{TP}{TP + FN} \tag{4.9}$$

La *Specificity* del sistema di classificazione rappresenta la percentuale di voti negativi previsti come tali:

$$Specificity = \frac{TN}{TN + FP} \tag{4.10}$$

L'*Accuracy* del sistema di classificazione rappresenta la percentuale di classificazioni corrette rispetto al totale di classificazioni effettuate (corrette o meno):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.11}$$

L'*Error Rate* del sistema di classificazione rappresenta la percentuale di errori compiuti dall'algoritmo di classificazione:

$$ErrorRate = \frac{FP + FN}{TP + TN + FP + FN} \tag{4.12}$$

Una delle metriche grafiche utilizzate per valutare l'efficacia del classificatore binario è la *Curva ROC* che rappresenta la correlazione esistente tra i valori di *Specificity* e di *Sensitivity* del sistema.

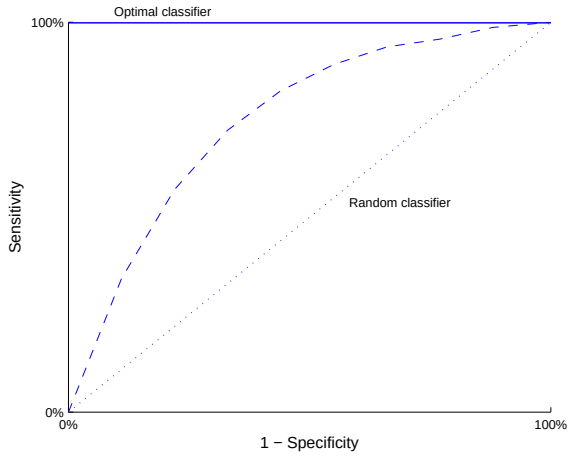


Figura 4.6: Receiver operating characteristic

Tali valori rappresentano nello specifico la probabiità dell'algoritmo di selezionare in maniera casuale item negativi votati come tali e la probabilità di selezionare item positivi votati come tali; i valori del grafico vengono ottenuti variando il valore di threshold utilizzato nel classificatore per

distinguere gli item da classificare come positivi dagli item da classificare come negativi.

L'area sottesa alla curva ROC rappresenta, quindi, la capacità dell'algoritmo di valutare correttamente il voto degli item; un algoritmo con funzionamento perfetto è associato ad un'area pari a 1.

4.2.2 Sistemi time-aware

I sistemi time-aware utilizzano generalmente le stesse metriche applicate ai sistemi time-free; in alcuni scenari, tuttavia, sono state sviluppate nuove metodologie di valutazione degli algoritmi di raccomandazione orientate ad evidenziare particolari comportamenti legati al fattore temporale [83] [53] [84].

Una metrica utilizzata per valutare l'evoluzione dell'accuratezza del sistema nel tempo è la *TA-RMSE* (Time Averaged Root Mean Squared Error); tale metrica si pone come obiettivo il calcolo del valore RMSE in uno specifico istante t attraverso l'analisi delle predizioni generate fino a tale istante.

Il TA-RMSE viene definito come:

$$TA - RMSE_t = \sqrt{\frac{\sum_{i \in P_t} (\hat{r}_{ui} - r_{ui})^2}{|P_t|}} \tag{4.13}$$

dove P_t è l'insieme di predizioni generate fino all'istante t.

Un'altra metrica che cerca di stabilire la capacità di adattamento nel tempo del sistema di raccomandazione ai cambiamenti nella matrice dei voti è la *Diversità*, che misura il grado di differenza esistente tra due liste di raccomandazione generate in intervalli differenti:

$$Diversità(L_1, L_2, N) = \frac{|L_2 \setminus L_1|}{N} \tag{4.14}$$

dove L_i è la lista di raccomandazione i-esima,$L_2 \setminus L_1 = \{x \in L_2 | x \notin L_1\}$ ed N è la lunghezza della lista di raccomandazione.

La metrica della *Novelty*, invece, viene utilizzata per valutare in che modo cambiano le liste di raccomandazione nel tempo, in termini di nuovi item inseriti nella lista; tale valutazione viene effettuata confrontando gli item inseriti nella lista i-esima con l'insieme di tutti gli item raccomandati fino all'istante t. La Novelty viene quindi definita come:

$$Novelty(L_1, N) = \frac{|L_1 \setminus A_t|}{N} \tag{4.15}$$

dove L_i è la lista di raccomandazione i-esima e A_t è l'insieme contenente tutti gli item raccomandati nelle liste fino all'istante t.

4.2.3 Metodologia di valutazione

L'insieme dei test eseguiti sul nostro sistema di raccomandazione per stimare l'efficacia dell'algoritmo implementato e determinare il valore di alcuni dei parametri di funzionamento utilizza due differenti strategie di validazione del modello generato:

k-Fold Cross Validation: il dataset originale viene suddiviso in k sottoinsiemi, ciascuno caratterizzato da un numero uguale (o quasi) di elementi. Viene, quindi, eseguita una serie di k test, utilizzando a rotazione uno dei k sottoinsiemi come *test set* e i restanti k-1 insiemi come *training set*.

L'utilizzo di tale strategia permette di evitare problemi associati all'errato campionamento del training set, come l'overfitting o il campionamento asimmetrico, presenti in situazioni nelle quali il dataset viene suddiviso in due sole parti (training set e validation set).

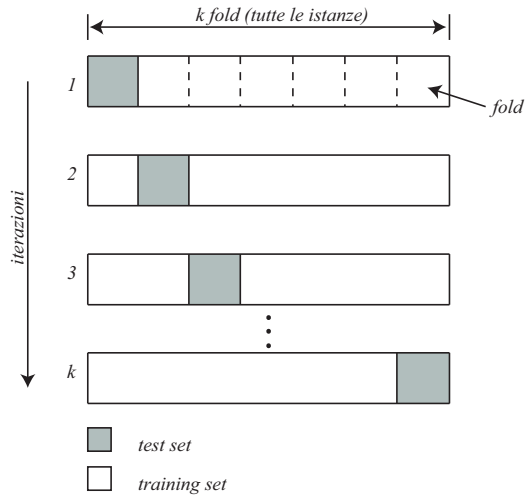


Figura 4.7: k fold Cross Validation

Interval Validation: il dataset originale viene ordinato temporalmente e suddiviso in sottoinsiemi associati a specifici intervalli temporali. Viene, quindi, eseguita una serie di test per ogni intervallo ottenuto, utilizzando come *test set* l'intervallo i-esimo e come *training test* i precedenti i-1 intervalli.

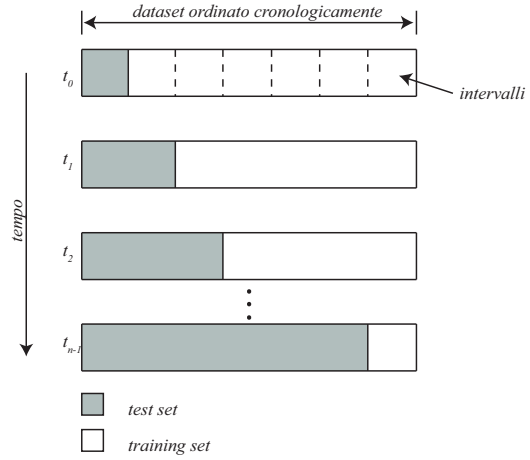


Figura 4.8: Interval Validation

4.3 Risultati

L'insieme dei test eseguiti sul sistema di raccomandazione ha avuto come obiettivo l'analisi dell'impatto delle singole caratteristiche dell'algoritmo sulle predizioni finali e sulle prestazioni ottenute; in particolare ciascun aspetto caratterizzante dell'algoritmo è stato valutato in maniera indipendente, in modo da isolare in maniera efficace i vantaggi o gli svantaggi legati al suo utilizzo.

I test sviluppati ed eseguiti sul sistema hanno analizzato in particolare il contributo di tre differenti elementi caratteristici utilizzati nell'algoritmo di raccomandazione:

Voti Negativi: l'algoritmo di raccomandazione, diversamente dagli algoritmi classici presenti in letteratura, utilizza nella fase di costruzione della matrice di similarità sia le preferenze positive espresse dagli utenti che quelle negative.

Cluster: l'algoritmo di raccomandazione utilizza una fase di clusterizzazione degli utenti basata sull'insieme delle preferenze espresse.

Decadimento Temporale: l'algoritmo utilizza un approccio di calcolo incrementale basato sulla suddivisione della fase di computazione in intervalli e sull'applicazione di un fattore di decadimento temporale agli item che ne decresce il peso con il tempo.

4.3.1 Valutazione dell'utilizzo dei voti negativi

I test eseguiti per valutare l'impatto dei voti negativi sulle predizioni generate dal sistema utilizzano la strategia di validazione *k-Fold Cross Validation*: il dataset viene suddiviso in 5 differenti fold, contenenti ciascuno un numero

equivalente o quasi di voti; a rotazione ciascun fold viene utilizzato come *test set* mentre i rimanenti fold costituiscono il *training set* sul quale l'algoritmo viene addestrato.

L'utilizzo dei voti negativi per generare i coefficienti della matrice di similarità rappresenta l'aspetto originale del nostro algoritmo di raccomandazione; gli algoritmi classici, infatti, utilizzano le sole preferenze positive per generare i coefficienti di similarità tra gli utenti, scartando totalmente l'insieme delle preferenze negative.

I coefficienti di similarità utilizzati all'interno di questa fase di test dell'algoritmo vengono calcolati attraverso l'applicazione del coefficiente di Jaccard a coppie di utenti:

$$sim(i,j) = J(i,j) = \frac{|I_i \cap I_j|}{|I_i \cup I_j|} \tag{4.16}$$

La valutazione dell'impatto dei voti negativi sulle predizioni del sistema viene effettuata utilizzando a rotazione all'interno dell'algoritmo una delle 5 possibili matrici di similarità, ottenute attraverso differenti combinazioni delle preferenze positive e delle preferenze negative degli utenti:

1. S^{++} : ogni riga s_{ij} della matrice contiene un valore di similarità ottenuto attraverso l'applicazione del coefficiente di Jaccard all'insieme delle preferenze positive espresse da ogni coppia di utenti i,j.

$$S^{++} = \left\{ s_{ij} : \forall i,j \in User, s_{ij} = \frac{\left|r_i^+ \cap r_j^+\right|}{\left|r_i^+ \cup r_j^+\right|} \right\} \tag{4.17}$$

2. S^{--} : ogni riga s_{ij} della matrice contiene un valore di similarità ottenuto attraverso l'applicazione del coefficiente di Jaccard all'insieme delle preferenze negative espresse da ogni coppia di utenti i,j.

$$S^{--} = \left\{ s_{ij} : \forall i,j \in User, s_{ij} = \frac{\left|r_i^- \cap r_j^-\right|}{\left|r_i^- \cup r_j^-\right|} \right\} \tag{4.18}$$

3. $S^{++}S^{--}$: ogni riga s_{ij} della matrice contiene un valore di similarità ottenuto come somma dei coefficienti di Jaccard associati alle preferenze positive e alle preferenze negative di ciascuna coppia di utenti i,j.

$$S^{++}S^{--} = \left\{ s_{ij} : \forall i,j \in User, s_{ij} = \frac{\left|r_i^+ \cap r_j^+\right|}{\left|r_i^+ \cup r_j^+\right|} + \frac{\left|r_i^- \cap r_j^-\right|}{\left|r_i^- \cup r_j^-\right|} \right\} \quad (4.19)$$

4. $S^{++}S^{+-}S^{-+}$: ogni riga s_{ij} della matrice contiene un valore di similarità ottenuto a partire dal coefficiente di Jaccard associato alle preferenze positive di ciascuna coppia di utenti i,j a cui viene sottratto il contributo fornito dal confronto tra gli item votati in maniera opposta dagli utenti stessi.

$$S^{++}S^{+-}S^{-+} = \left\{ s_{ij} : \forall i,j \in User, s_{ij} = \frac{\left|r_i^+ \cap r_j^+\right|}{\left|r_i^+ \cup r_j^+\right|} - \frac{\left|r_i^+ \cap r_j^-\right|}{\left|r_i^+ \cup r_j^-\right|} - \frac{\left|r_i^- \cap r_j^+\right|}{\left|r_i^- \cup r_j^+\right|} \right\} \quad (4.20)$$

5. $S^{++}S^{--}S^{+-}S^{-+}$: ogni riga s_{ij} della matrice contiene un valore di similarità ottenuto a partire dalla somma dei coefficiente di Jaccard associati alle preferenze positive e alle preferenze negative di ciascuna coppia di utenti i,j a cui viene sottratto il contributo fornito dal confronto tra gli item votati in maniera opposta dagli utenti stessi.

$$S^{++}S^{--}S^{+-}S^{-+} = \left\{ s_{ij} : \forall i,j \in User, s_{ij} = \frac{\left|r_i^+ \cap r_j^+\right|}{\left|r_i^+ \cup r_j^+\right|} + \frac{\left|r_i^- \cap r_j^-\right|}{\left|r_i^- \cup r_j^-\right|} - \frac{\left|r_i^+ \cap r_j^-\right|}{\left|r_i^+ \cup r_j^-\right|} - \frac{\left|r_i^- \cap r_j^+\right|}{\left|r_i^- \cup r_j^+\right|} \right\} \quad (4.21)$$

I test vengono eseguiti sul sistema di raccomandazione utilizzando una serie di variabili i cui valori ottimali vengono determinati attraverso i risultati ottenuti:

n: rappresenta la dimensione del neighborhood di ciascun utente i cui voti vengono utilizzati per produrre le raccomandazioni/predizioni.

In generale il valore ottimale del parametro n dipende dalla tipologia di dataset utilizzata nel sistema e viene determinato dopo l'esecuzione di una serie di test; tipicamente il valore selezionato deve essere un valore abbastanza grande da fornire risultati affidabili ma abbastanza piccolo da non introdurre disturbo nei dati. Una delle regole comunemente adottate per il settaggio di tale parametro è quella di selezionare il valore relativo alla radice quadrata del numero di istanze considerate (nel nostro caso la radice quadrata del numero di utenti del sistema).

La serie di test eseguita sul sistema di raccomandazione utilizza due valori estremi per il parametro n, in modo da rendere evidente i differenti comportamenti dell'algoritmo di raccomandazione al variare della dimensione del neighborhood: un valore $n = \sqrt{N} = 30$ che rappresenta il classico scenario di lavoro di un sistema di raccomandazione e un valore $n = N = 943$ che rappresenta uno scenario particolare in cui la dimensione del neighborhood non viene limitata.

k: rappresenta la dimensione della lista di raccomandazione fornita all'utente. Il numero di item inseriti in tale lista influenza direttamente le prestazioni dell'algoritmo di raccomandazione utilizzato: un valore maggiore, infatti, aumenta sensibilmente la probabilità di inserire nella lista un numero elevato di item di interesse per l'utente, con un conseguente aumento dei valori di recall del sistema (la recall

rappresenta la percentuale di item di interesse inseriti nella lista rispetto al totale di item di interesse presenti nel sistema); diversamente un valore ridotto aumenta la probabilità di inserire nella lista di raccomandazione solo item di interesse per l'utente, con un conseguente aumento dei valori di precision del sistema (la precision rappresenta la percentuale di item di interesse inseriti nella lista rispetto alla dimensione dalla lista stessa).

L'insieme di run dell'algoritmo eseguite in questa fase di valutazione del sistema ha avuto come obiettivo l'individuazione dei valori ottimali per i parametri n e k e l'analisi del comportamento dell'algoritmo al variare delle matrici di similarità utilizzate.

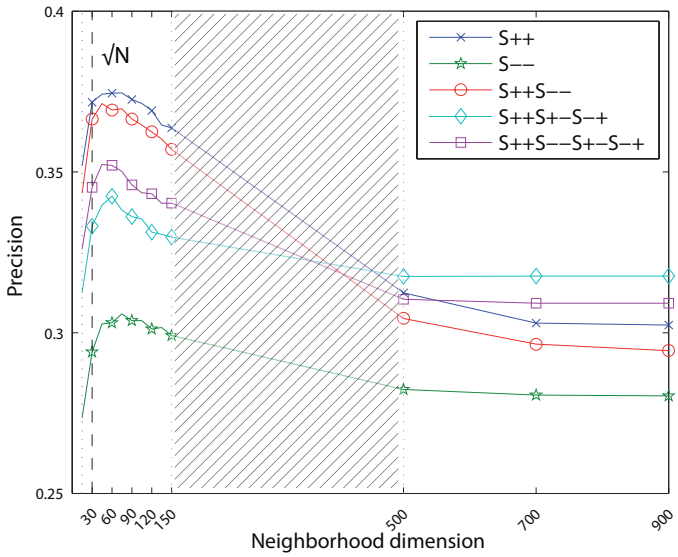


Figura 4.9: Valori di precision al variare della dimensione del neighborhood

Il grafico 4.9 mostra la variazione dei valori di precision dell'algoritmo al variare della dimensione del neighborhood di ciascun utente; tali test sono stati eseguiti utilizzando come valori del parametro n due differenti insiemi:

$n \in [15, 150]$ e $n \in [500, 700]$. I dati ottenuti dai test permettono di fare diverse considerazioni sul comportamento del sistema di raccomandazione in esame:

- *valore ottimale di n*: il grafico evidenzia come le prestazioni migliori dell'algoritmo in termini di precision si ottengano con un valore n pari circa a 60 utenti; l'aumento progressivo della dimensione del neighborhood a partire da tale valore, infatti, genera un graduale decadimento delle prestazioni totali del sistema.

  La causa di tale comportamento è da ricercarsi nell'aumento del disturbo presente nei dati che costituiscono il modello di predizione dell'algoritmo. Gli algoritmi di tipo *nearest-neighbors*, infatti, cercano di limitare la dimensione del neighborhood di ciascun utente inserendo i soli utenti associati a valori di similarità elevata, le cui preferenze, perciò, sono orientate ad item con alta probabilità di gradimento per l'utente finale. L'utilizzo, invece, di una lista di neighbors estesa causa l'introduzione di un numero consistente di utenti associati a valori di similarità ridotti e quindi con interessi non compatibili con quelli dell'utente finale; la presenza di tali utenti, perciò, determina l'ingresso nella lista di raccomandazione di un set di item associati ad una bassa probabilità di gradimento, avendo ricevuto un numero elevato di voti dai neighbors meno simili all'utente finale.

  Il grafico evidenzia, inoltre, come la scelta euristica del valore di n pari a $\sqrt{N}$ permetta di ottenere delle prestazioni per l'algoritmo non ottimali ma in ogni caso positive; tale valore, quindi, si dimostra una buona scelta in situazioni nelle quali si cerca di ridurre il tempo di computazione totale riducendo lo spazio di istanze analizzate dall'algoritmo per generare la raccomandazione finale.

- *comportamento delle diverse matrici di similarità*: l'analisi dell'andamento dei valori di precision associati alle differenti tipologie di matrici di similarità utilizzate nell'algoritmo permette di evidenziare due differenti comportamenti collegati ai diversi insiemi di valori assegnabili al parametro n.

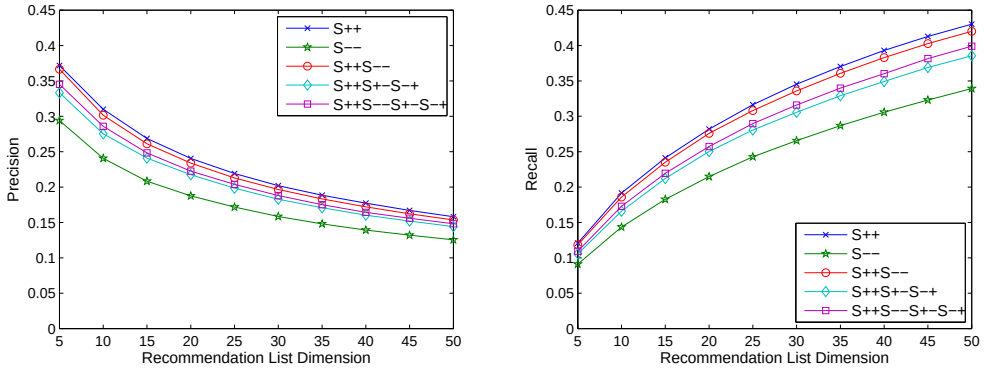
  I risultati associati ai test eseguiti con un valore di n appartenente all'insieme $[15, 150]$ mostrano la preminenza delle run che utilizzano nella costruzione della matrice di similarità le sole preferenze positive o i valori ottenuti dal confronto delle preferenze positive e delle preferenze negative degli utenti; diversamente le matrici i cui valori di similarità presentano delle componenti legate alla diversità di opinione sui medesimi item mostrano un comportamento peggiore, seguite dalla matrice S^{--} che ha in assoluto le prestazioni peggiori.

  I risultati associati ai test eseguiti con un valore di n appartenente all'insieme $[500, 900]$, invece, mostrano un comportamento opposto; in tale scenario, infatti, le matrici contenenti delle componenti di similarità associate alla diversità di opinione tra coppie di utenti presentano le prestazioni migliori.

  I dati relativi all'intervallo di valori $[150, 500]$ non sono stati rilevati ma dall'andamento globale del grafico è possibile ipotizzare la presenza di un valore specifico del parametro n a partire dal quale il comportamento delle matrici $S^{++}, S^{++}S^{--}, S^{++}S^{+-}S^{-+}$ e $S^{++}S^{--}S^{+-}S^{-+}$ si inverte.

I grafici 4.10 e 4.11 permettono di approfondire il comportamento del sistema di raccomandazione in due scenari di funzionamento specifici associati ai valori del parametro n pari a $\sqrt{N}$ e pari a N; in particolare i dati rappresentati in tali grafici mostrano l'andamento dei valori di precision

e di recall associati all'algoritmo di raccomandazione testato al variare della dimensione della lista di raccomandazione.

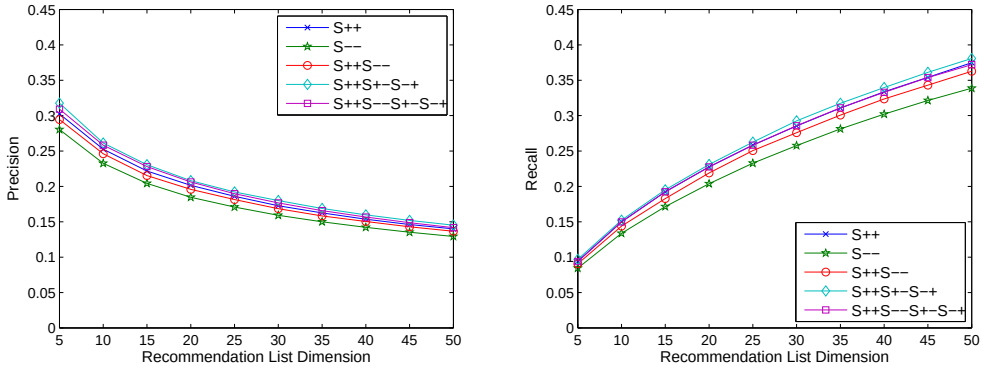


(a) Precision (b) Recall

Figura 4.10: Prestazioni del sistema con dimensione del neighborhood pari a 30 utenti

I grafici evidenziano come i valori di precision e di recall abbiano un comportamento inversamente proporzionale legato alla variazione della dimensione della lista di raccomandazione: una dimensione ridotta della lista, infatti, privilegia i valori di precision, dal momento che rappresentano la percentuale di raccomandazioni esatte rispetto alla dimensione della lista stessa; diversamente una dimensione estesa della lista privilegia i valori di recall, dal momento che rappresentano la percentuale di raccomandazioni esatte rispetto al totale degli item raccomandabili.

I risultati ottenuti, inoltre, confermano quanto rilevato dall'analisi del grafico 4.9: in uno scenario di funzionamento classico quale quello associato ad un valore ridotto del parametro n, le prestazioni migliori vengono ottenute attraverso l'utilizzo della matrice S^{++} mentre in uno scenario di funzionamento inusuale come quello in cui non viene limitato il numero di neighbors le prestazioni migliori vengono ottenute dalla matrice $S^{++}S^{+-}S^{-+}$.



(a) Precision (b) Recall

Figura 4.11: Prestazioni del sistema con dimensione del neighborhood illimitata

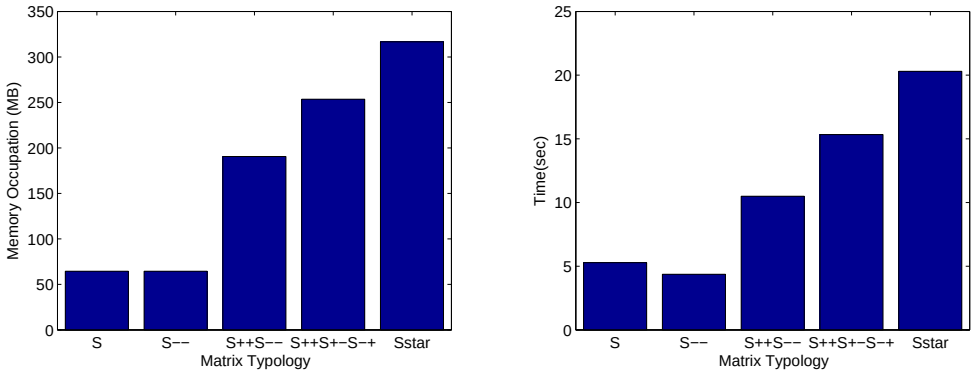
Tale comportamento dell'algoritmo è legato alla tipologia di item che vengono considerati per la predizione finale nei due scenari di funzionamento: nel primo scenario l'utilizzo di un neighborhood ridotto permette di selezionare gli item da inserire nella lista di raccomandazione solo all'interno degli item votati da utenti con valori di similarità particolarmente elevati, garantendo così un'alta probabilità di gradimento. Nel secondo scenario, invece, la lista di raccomandazione viene generata selezionando gli item con score più alto tra tutti quelli che hanno ricevuto almeno un voto da un utente; alcuni degli item che entrano nel processo di raccomandazione, perciò, possono essere associati ad uno score elevato ottenuto grazie al numero consistente di voti ricevuti da utenti con valori di similarità ridotti e quindi con interessi differenti da quelli dell'utente attuale. In tale situazione l'utilizzo nella fase di calcolo dei valori di similarità di un fattore che dipende dalla diversità tra utenti permette di adattare in maniera più realistica i valori di similarità agli interessi degli utenti, assegnando peso maggiore agli individui che condividono le stesse opinioni positive e negative

e penalizzando, invece, quelli che esprimono pareri opposti sugli stessi item.

L'introduzione, perciò, dei voti negativi all'interno dell'algoritmo di calcolo delle similarità tra utenti riesce ad eliminare i fattori di disturbo presenti quando la dimensione del neighborhood non viene limitata; tuttavia, l'analisi dei valori assoluti di precision e di recall raggiunti dalla matrice $S^{++}S^{+-}S^{-+}$ fa capire come le prestazioni totali del sistema di raccomandazione siano inferiori rispetto a quelle ottenute in uno scenario di funzionamento con neighborhood ridotto. La strategia comunemente adotta di scegliere un sottoinsieme di utenti ad alto valore di similarità per generare le raccomandazioni, infatti, si dimostra l'approccio migliore per implementare un algoritmo efficace: l'utilizzo dei soli voti positivi all'interno della matrice di similarità S^{++} in un contesto di funzionamento limitato a $n = \sqrt{N}$ assicura valori di precision e di recall superiori a quelli ottenuti con l'utilizzo di una qualsiasi delle altre matrici di similarità a disposizione nel contesto di funzionamento $n = N$.

Un'ulteriore prova del vantaggio esistente nell'utilizzo della strategia tipica di riduzione del neighborhood e dell'utilizzo dei soli voti positivi per le matrici di similarità è fornito dall'analisi dei grafici 4.12a e 4.12b che rappresentano rispettivamente l'occupazione di memoria e il tempo di calcolo associati alle run dell'algoritmo eseguite con neighborhood ridotto.

I dati a dispozione, infatti, mostrano come l'utilizzo dei soli voti positivi nella costruzione della matrice di similarità permetta di ridurre da un lato l'occupazione di memoria del sistema, che mantiene la sola matrice S^{++} rispetto alle quattro matrici $S^{++}, S^{--}, S^{+-}, S^{-+}$ necessarie agli altri scenari di funzionamento, e dall'altro i tempi di computazione totali dell'algoritmo, che si limita ad effettuare un confronto sui soli item votati positivamente rispetto all'insieme totale di item votati dagli utenti.



(a) Occupazione di memoria per tipologia di matrice

(b) Tempo di esecuzione per tipologia di matrice

Figura 4.12: Occupazione di memoria e tempo di esecuzione con n=30

Il contributo, perciò, fornito dall'elemento originale del nostro lavoro di tesi, l'introduzione delle preferenze negative nel calcolo delle similarità, ha un impatto negativo all'interno di un tipico scenario di funzionamento di un sistema di raccomandazione: l'intuizione, infatti, che le dissimilarità tra i voti espressi possano migliorare la procedura di profilazione degli interessi dei singoli utenti trova applicazione in un punto di funzionamento del sistema di raccomandazione non utilizzabile nella pratica nei sistemi reali a causa degli elevati costi legati al processamento di una quantità elevata di informazioni relative ai voti e agli item e a causa di prestazioni non ottimali.

4.3.2 Cluster

Nella seconda fase di test in cui viene valutato l'impatto dell'utilizzo della strategia di clusterizzazione degli utenti sulle prestazioni totali del sistema viene utilizzata la strategia di validazione *k-Fold Cross Validation* con il dataset suddiviso in 5 differenti fold.

L'introduzione di un algoritmo di clusterizzazione degli utenti ha come obiettivo la riduzione dei tempi di processamento dei dati del sistema e di conseguenza dei tempi di generazione delle raccomandazioni per gli utenti finali; la fase di *compare* dell'algoritmo, infatti, viene eseguita attraverso il confronto dei set dei cluster di appartenenza di ciascun utente, caratterizzati da una dimensione ridotta rispetto agli insiemi di item votati.

La tecnica impiegata per generare i cluster di appartenenza dei singoli utenti utilizza l'algoritmo di *MinHash*:

1. Vengono scelti $p*q$ valori di *seed* casuali
2. Per ogni valore di seed $s_i \in$ Seed:
   (a) Viene concatenato il valore s_i all'id id_j di ciascun item votato dall'utente
   (b) Viene applicata una funzione di hash ai valori concatenati $s_i||id_j$
   (c) Viene memorizzato il valore di hash minimo tra quelli ottenuti su tutti gli item dell'utente
3. Il set dei $p*q$ valori di hash minimi viene suddiviso in q insiemi di p valori ciascuno, rappresentanti gli id dei p cluster di appartenenza del singolo utente

Formalmente i valori assegnati ai parametri p e q definiscono le modalità di costruzione dei cluster assegnati all'utente, all'interno dei quali vengono inseriti utenti con interessi simili :

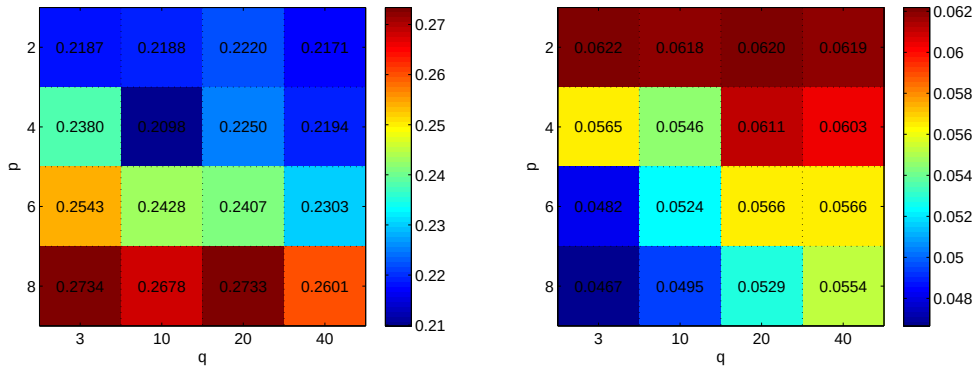
p: rappresenta il numero di valori di hash che definiscono un singolo cluster: valori elevati di p permettono di definire cluster più piccoli ma più precisi mentre valori ridotti di p permettono di definire cluster di grandi

dimensioni ma poco precisi. Il valore di p, perciò, permette di modulare la precision dell'insieme di neighbors di ciascun utente.

q: rappresenta il numero totale di cluster a cui un singolo utente appartiene: valori elevati di q permettono di assegnare ciascun utente ad un numero di cluster tale da rappresentare un insieme consistente di interessi mentre valori ridotti di q permettono di assegnare ciascun utente ad un set di cluster rappresentativi solo di un sottoinsieme ridotto dei suoi interessi. Il valore di q, perciò, permette di modulare la recall dell'insieme dei neighbors di ciascun utente.

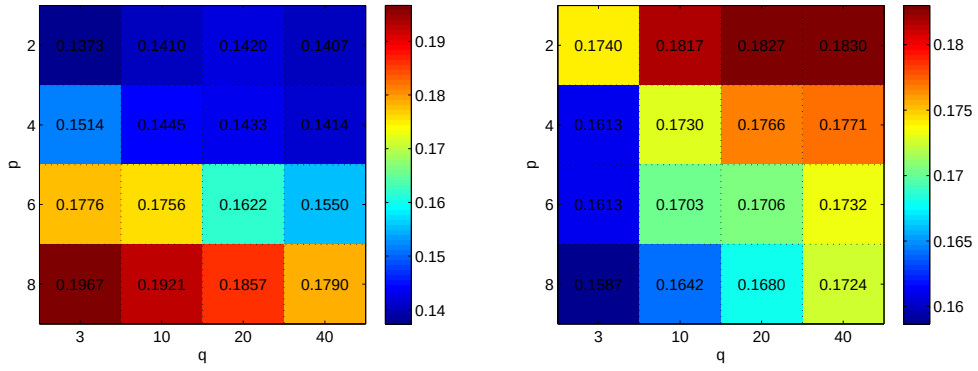
I primi test eseguiti in questa fase, perciò, hanno avuto come obiettivo l'individuazione dei valori ottimali dei parametri p e q del sistema in esame. Tipicamente i valori assegnabili a tali parametri sono contenuti negli insiemi $p \in [2, 4]$ e $q \in [10, 20]$; nei nostri test, tuttavia, è stato scelto per il parametro p l'insieme standard di valori mentre il parametro q è stato valutato nell' insieme di valori $[3, 40]$. I test sono stati eseguiti utilizzando tre differenti valori k per la dimensione della lista di raccomandazione, in modo da individuare un valore medio per i parametri p e q utilizzabile in diversi scenari: $k = 5$, $k = 25$ e $k = 50$.

I risultati ottenuti, visibili nelle figure 4.13, 4.14 e 4.15 sono stati organizzati in una serie di heatmap in modo da rendere facilmente visibile i valori associati alle prestazioni migliori. Una delle prime considerazioni che può essere effettuata dall'analisi dei dati è la presenza di un andamento inversamente proporzionale dei valori di precision e di recall rispetto ai diversi valori assegnati ai parametri p e q; tale comportamento rappresenta una caratteristica tipica dei sistemi di raccomandazione, nei quali l'ottimizzazione di uno dei due parametri si riflette nella penalizzazione dell'altro.



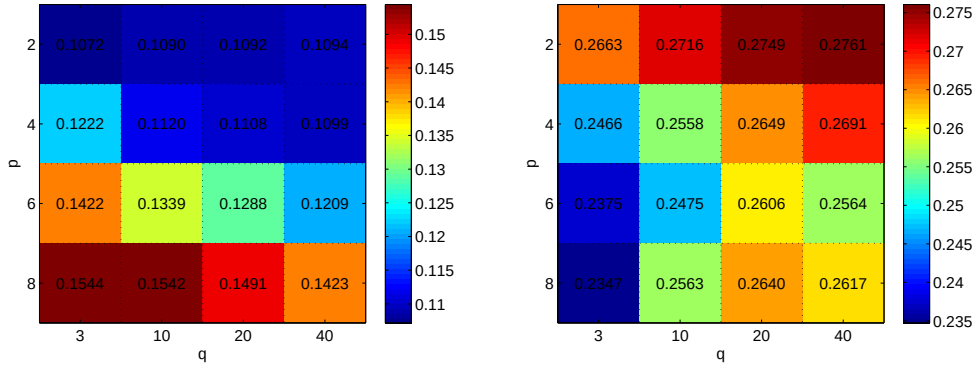
(a) Precision (b) Recall

Figura 4.13: HeatMap associata all'esecuzione dell'algoritmo con dimensione del neighborhood pari a 5



(a) Precision (b) Recall

Figura 4.14: HeatMap associata all'esecuzione dell'algoritmo con dimensione del neighborhood pari a 25



(a) Precision (b) Recall

Figura 4.15: HeatMap associata all'esecuzione dell'algoritmo con dimensione del neighborhood pari a 50

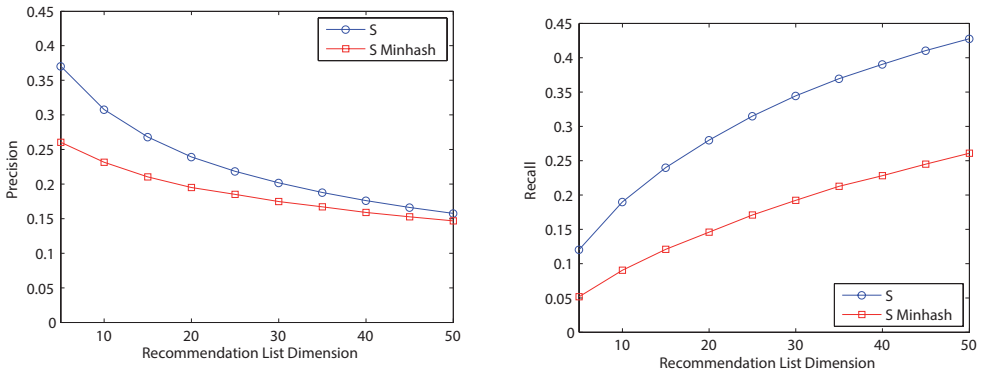
La scelta, perciò, dei valori ottimali per i parametri p e q deve essere fatta considerando un valore medio che permetta di ottenere delle prestazioni adeguate in termini di precison e recall per ciascuna dimensione della lista di raccomandazione considerata. I valori assegnati ai due parametri per l'esecuzione delle successive run dell'algoritmo sono rispettivamento $p = 8$ e $q = 20$, che permettono di ottenere valori alti di precision mantenendo la recall su un valore medio; in tale scenario la *history* dei voti permette di associare ciascun utente ad un insieme di 20 cluster identificati da un *id* costituito da 8 valori di hash concatenati.

I grafici 4.16a e 4.16b rappresentano il confronto tra l'esecuzione di due run dell'algoritmo:

Run con matrice S : la prima run non utilizza alcun algoritmo di clusterizzazione degli utenti; l'algoritmo di calcolo delle similarità si basa sulla costruzione di una matrice di similarità S ottenuta attraverso l'applicazione del *coefficiente di Jaccard* all'insieme delle preferenze

positive degli utenti.

Run con matrice SMinHash : la seconda run utilizza l'algoritmo di MinHash per la clusterizzazione degli utenti; l'algoritmo di calcolo delle similarità si basa sulla costruzione di una matrice di similarità *SMinHash* ottenuta attraverso l'applicazione del *coefficiente di Jaccard* all'insieme dei cluster generati attraverso l'analisi delle preferenze positive degli utenti.



(a) Precision (b) Recall

Figura 4.16: Comparazione tra S e SMinHash

I dati evidenziano come l'introduzione dei cluster nella definizione della similarità tra utenti determini un peggioramento dei valori assoluti di precision e di recall dell'algoritmo: l'assegnazione degli utenti ad un numero di cluster predefinito e limitato, infatti, genera un modello semplificato degli interessi dei singoli individui.

I cluster sono, in effetti, la rappresentazione di un sottoinsieme caratteristico di tutti gli interessi espressi dall'utente; tale tipologia di modello permette, quindi, di cogliere le correlazioni esistenti con gli utenti più significativi associate a particolari insiemi di item, perdendo nel processo

alcune informazioni che permettono, nei sistemi con matrici di similarità classiche, di raffinare i valori di precision e di recall ottenuti. I coefficienti di similarità definiti attraverso i cluster, infatti, sono valori che approssimano i valori reali di correlazione tra gli utenti e che quindi sono soggetti in alcuni casi ad errori e arrotondamenti rispetto ai valori ottenuti dal confronto dell'intera *history* delle preferenze.

Il vantaggio associato all'utilizzo dell'algoritmo di clusterizzazione degli utenti è rappresentato da una riduzione dei tempi di computazione della fase di compare, come evidenziato nella tabella 4.3; in particolare è possibile osservare come l'utilizzo dei cluster nel sistema di raccomandazione consenta di raggiungere una riduzione di circa il 75% dei tempi della fase di compare con un aumento del solo 4% della memoria occupata dovuto alla memorizzazione della lista di cluster di appartenza di ciascun utente.

	S	S MinHash
Tempo di computazione fase compare	6.91 sec	1.61 sec
Occupazione di Memoria totale	26.34 MB	27.44 MB

Tabella 4.3: Prestazioni del sistema di raccomandazione senza l'utilizzo di MinHash e con l'utilizzo di MinHash

L'analisi approfondita dei tempi di computazione relativi alla fase di compare evidenzia come un'ulteriore vantaggio associato all'utilizzo dell'algoritmo di clusterizzazione degli utenti sia la possibilità di parallelizzare tale fase di elaborazione senza la necessità di implementare particolari procedure di bilanciamento del carico.

Il grafico 4.17 rappresenta i tempi di esecuzione della fase di compare relativi ad un campione di 100 utenti del sistema di raccomandazione; ogni valore del grafico, nello specifico, rappresenta la durata della fase di confronto

degli item/cluster di un singolo utente con quelli dei restanti utenti del sistema di raccomandazione per il calcolo dei valori di similarità associati.

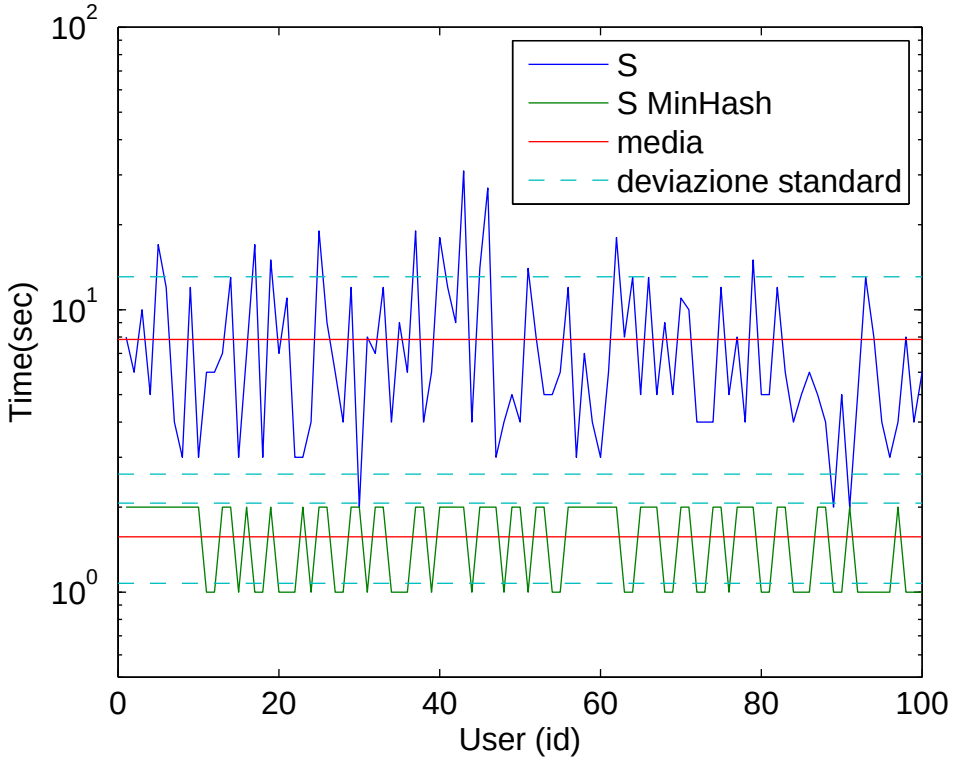


Figura 4.17: Comparazione dei tempi di esecuzione della fase di compare tra S e SMinHash

I tempi totali dipendono direttamente dai tempi di esecuzione del metodo Java *retainAll(Collection <?>C)*, che si occupa di calcolare l'intersezione tra gli item/cluster di coppie di utenti *i,j* attraverso la ricerca degli elementi contenuti nel set dell'utente i-esimo all'interno del set dell'utente j-esimo; tale approccio determina, quindi, costi più elevati quando $|Item_i| > |Item_j|$.

L'esecuzione dell'algoritmo senza l'utilizzo della fase di clustering degli utenti prevede il confronto tra insiemi di item di dimensioni differenti; i tempi riportati nel grafico, di conseguenza, mostrano una elevata variabilità

e sono maggiori per utenti con un numero elevato di preferenze. Diversamente l'esecuzione dell'algoritmo con l'utilizzo dei cluster di appartenenza degli utenti prevede il confronto tra insiemi di dimensioni costanti determinate dal valore del parametro q; in questo scenario, perciò, i tempi di esecuzione presentano una bassa variabilità, dovuta principalmente ai tempi di esecuzione delle operazioni accessorie utilizzate per implementare l'intersezione.

La fase di clusterizzazione degli utenti introdotta nel sistema di raccomandazione fornisce, quindi, un efficace metodo per migrare la computazione dell'algoritmo di raccomandazione da un sistema centralizzato ad un sistema di calcolo parallelo senza la necessità di elaborare un algoritmo per la distribuzione ottimale del carico.

4.3.3 Decadimento temporale

Nella terza fase di test viene valutato l'impatto del decadimento temporale del peso degli item votati dagli utenti sulle prestazioni del sistema di raccomandazione; in questo particolare scenario la tipologia di validazione adottata è la *Interval Validation*, che prevede la suddivisione del dataset ordinato cronologicamente in un insieme di 8 intervalli di 30 giorni ciascuno.

La tipologia di validazione utilizzata prevede che ad ogni intervallo *i-esimo* il *training set* sia costituito dai voti espressi negli intervalli$[t_0, t_i]$ mentre il *test set* sia costituito dai voti espressi negli intervalli $[t_{i+1}, t_n]$.

In tale scenario i valori di precision e di recall associati al sistema di raccomandazione vengono calcolati per ciascun intervallo *i-esimo* di osservazione.

Il valore di $Precision(t_i)$, che rappresenta la percentuale di item significativi inseriti nella lista di raccomandazione i-esima rispetto alla

dimensione della lista, viene definito come:

$$Precision(t_i) = \frac{I_{R_u}(\tilde{t_i}) \cap I_{K_u}(t_i)}{|I_{K_u}(t_i)|} \tag{4.22}$$

dove $\tilde{t_i}$, $I_{R_u}(\tilde{t_i})$ e $I_{K_u}(t_i)$ sono rispettivamente il test set $\tilde{t_i} = [t_{i+1}, t_n]$, l'insieme degli item rilevanti per l'utente u nell'intervallo i e l'insieme degli item restituiti nella lista di raccomandazione generata nell'intervallo i.

Il valore di $Recall(t_i)$, che rappresenta la percentuale di item significativi individuati dall'algoritmo nell'intervallo i-esimo rispetto al totale di item significativi, viene definito come:

$$Recall(t_i) = \frac{I_{R_u}(\tilde{t_i}) \cap I_{K_u}(t_i)}{\left|I_{R_u}(\tilde{t_i})\right|} \tag{4.23}$$

dove $\tilde{t_i}$, $I_{R_u}(\tilde{t_i})$ e $I_{K_u}(t_i)$ sono rispettivamente il test set $\tilde{t_i} = [t_{i+1}, t_n]$, l'insieme degli item rilevanti per l'utente u nell'intervallo i e l'insieme degli item restituiti nella lista di raccomandazione generata nell'intervallo i.

La distribuzione non omogenea dei voti lungo l'intero arco temporale associato al dataset causa la creazione di un insieme di intervalli contenenti un numero variabile di voti; molti utenti, inoltre, presentano delle abitudini di voto discontinue che possono causare situazioni nelle quali la maggior parte delle preferenze vengono assegnate ad un medesimo intervallo temporale, impedendo quindi al modello degli interessi degli utenti di evolvere nel tempo come in un qualsiasi sistema reale.

In questo particolare scenario di funzionamento l'algoritmo di raccomandazione utilizza come strategia per il calcolo delle similarità tra gli utenti l'applicazione del *Coseno di similarità* all'insieme di item votati positivamente:

$$sim(i,j) = \frac{\sum_{r \in I_i^+ \cap I_j^+} f^{\alpha}_{r_{u_i}}(t) \cdot f^{\alpha}_{r_{u_j}}(t)}{\sqrt{\sum_{r \in I_i^+} (f^{\alpha}_{r_{u_i}}(t))^2 \sum_{r \in I_j^+} (f^{\alpha}_{r_{u_j}}(t))^2}} \tag{4.24}$$

I dati ottenuti dai test eseguiti in questa fase di valutazione dipendono da due differenti parametri:

Fattore di decadimento α : il fattore di decadimento α determina la velocità di dimimuzione del peso degli item votati dagli utenti secondo la formula:

$$f^{\alpha}_{r_{u_i}}(t_k + 1) = e^{-\alpha} \cdot f^{\alpha}_{r_{u_i}}(t_k) \tag{4.25}$$

I valori assegnati a tale parametro durante i vari test appartengono all'insieme $A = \{0, 0.01, 0.05, 0.1, 0.5, 0.9\}$. L'introduzione del fattore temporale permette di modellare la variazione del tempo degli interessi degli utenti che in un sistema reale sono in continua evoluzione.

Valore di threshold t : il valore di threshold t rappresenta il peso minimo che un item può assumere nel sistema di raccomandazione prima di essere eliminato dalle strutture dati. I valori assegnati a tale parametro durante i vari test appartengono all'insieme $T = \{0.5, 0.8\}$. L'introduzione del valore di threshold t permette di limitare, in un sistema di raccomandazione in continua crescita come può essere un sistema reale, la dimensione delle strutture dati che mantengono le informazioni necessarie all'esecuzione degli algoritmi, eliminando di volta in volta i dati il cui contributo è diventato trascurabile ai fini della raccomandazione finale.

I primi test eseguiti sul sistema di raccomandazione hanno avuto come obiettivo l'analisi dell'andamento dei pesi degli item nel tempo al variare del parametro α.

Il grafico 4.18 mostra i risultati ottenuti applicando ad ogni intervallo t_k la funzione di decadimento temporale al peso dell'item, impostato ad un valore iniziale pari a 1.

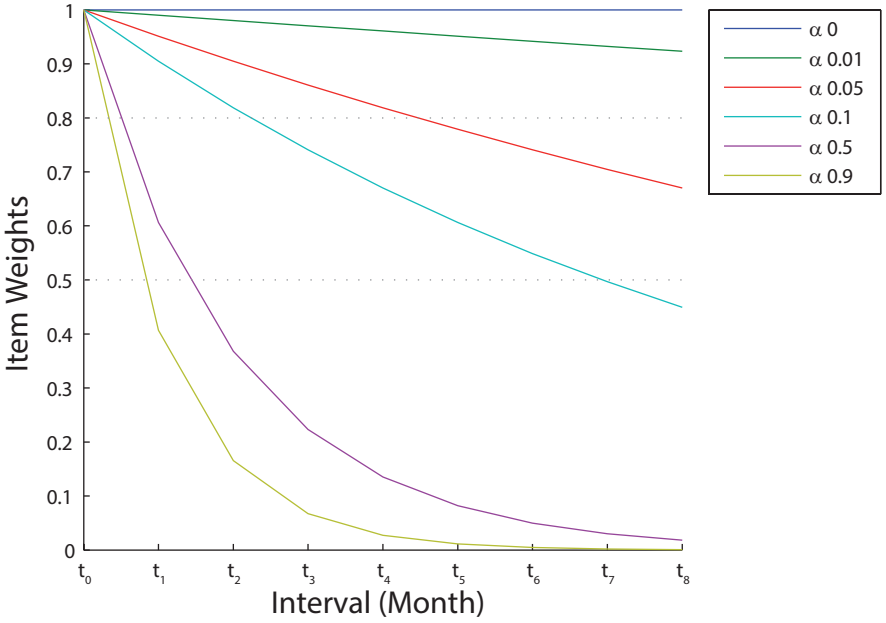


Figura 4.18: Andamento del peso degli item al variare del parametro α

L'utilizzo di un fattore α il cui valore è contenuto nell'insieme $A_1 = \{0, 0.01, 0.05, 0.1\}$ determina una tendenza di dimunizione del peso degli item i cui valori approssimano un comportamento di tipo lineare; diversamente, l'utilizzo di un fattore α il cui valore è contenuto nell'insieme $A_2 = \{0.5, 0.9\}$ permette di osservare all'interno degli 8 intervalli temporali di funzionamento del sistema l'andamento esponenziale associato alla funzione di diminuzione del peso degli item.

Il grafico 4.18 permette anche di osservare in che modo la scelta del corretto valore per il fattore di threshold t sia relazionata al valore utilizzato per il parametro α; in particolare i dati a disposizione evidenziano come run associate a valori di α ridotti, come quelli contenuti nell'insieme $A_1 = \{0.01, 0.05, 0.1\}$, necessitino di un valore di threshold particolarmente elevato per l'esecuzione della fase di eliminazione dei dati non significativi mentre

run associate a valori di α elevati, come quelli contenuti nell'insieme $A_1 = \{0.5, 0.9\}$, possano utilizzare valori di threshold più bassi per il peso degli item da eliminare.

Il valore specifico scelto per il fattore di threshold t, inoltre, determina l'intervallo temporale a partire dal quale l'insieme degli item viene ridimensionato; tale scelta, perciò, deve essere effettuata dopo un'attenta analisi condotta sul dataset, in modo da individuare la velocità ottimale di decadimento dei fattori di peso e il modello evolutivo dei dati che si vuole generare attraverso l'eliminazione degli elementi non significativi.

Il successivo insieme di test viene eseguito per valutare le prestazioni del sistema di raccomandazione in termini di precision e di recall al variare del fattore di decadimento utilizzato e del valore di threshold scelto.

In un sistema reale, caratterizzato da un flusso costante di voti generato da utenti con un elevato fattore di interazione con il sistema, i valori di precision e di recall ottenuti nell'intero arco temporale di osservazione dipendono da due caratteristiche in contrasto: da un lato l'utilizzo di un insieme finito di intervalli temporali causa la diminuizione progressiva della dimensione dell'insieme di item su cui vengono verificate le raccomandazioni generate e di conseguenza determina la diminuzione dei valori di precision e di recall ottenuti; dall'altro lato l'utilizzo di un algoritmo operante per intervalli determina la costruzione di una matrice di similarità la cui affidabilità migliora all'aumentare del tempo, a causa della disponibilità di un insieme di dati sempre più consistente su cui effettuare il training dell'algoritmo.

L'azione di queste due caratteristiche, quindi, genera teoricamente un grafico per i valori di precision e di recall suddivisibile in due differenti parti: la prima parte, relativa ai primi intervalli di osservazione del sistema, è associata ad un andamento crescente dei valori rilevati, causato dal

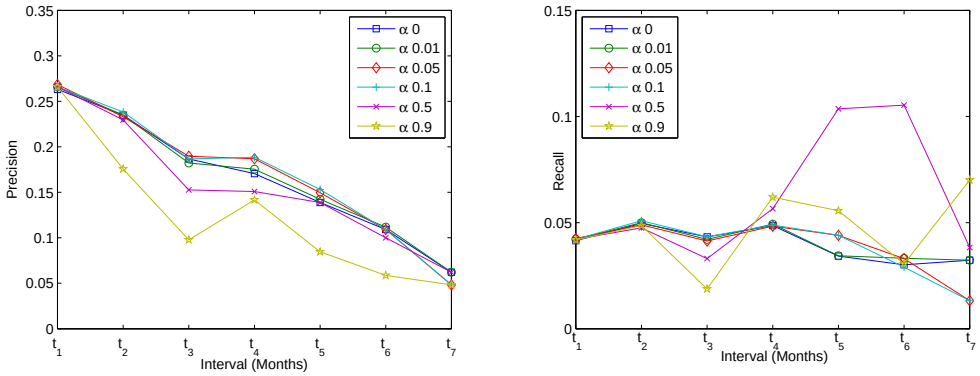
progressivo miglioramento del modello degli interessi degli utenti grazie all'utilizzo di un training set crescente; la seconda parte, relativa ai successivi intervalli di osservazione del sistema, è associata ad un andamento decrescente dei valori rilevati, causato dalla diminuzione costante del test set dell'algoritmo.

Il dataset associato al sistema in esame, tuttavia, è caratterizzato da un insieme di utenti che non interagiscono con regolarità con il sistema di raccomandazione, limitando, quindi, nei risultati ottenuti l'azione della fase di training dell'algoritmo; in molti casi, infatti, le preferenze espresse da ciascun utente sono contenute in un singolo intervallo temporale e il modello così generato permette di rappresentare solo un insieme istantaneo degli interessi espressi.

I risultati dei test condotti con i valori del fattore di threshold pari a 0.5 e a 0.8 sono rappresentati nei grafici 4.19 e 4.20, nei quali viene evidenziato l'andamento dei valori di precision e di recall al variare del parametro α.

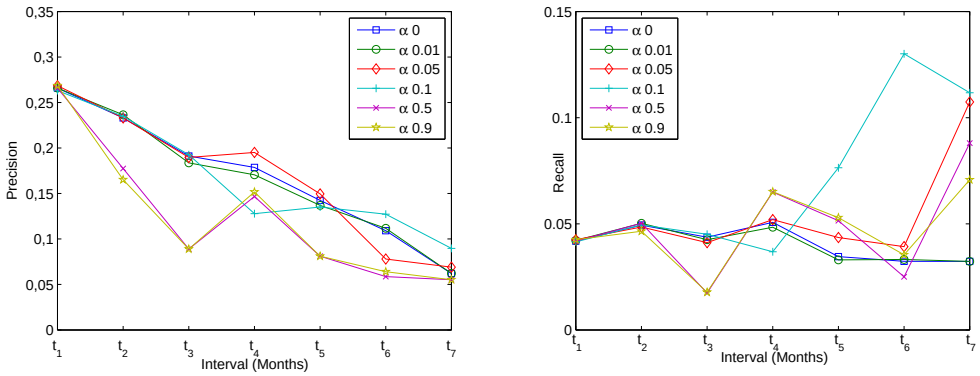
L'analisi dei grafici permette di osservare come le run associate ad un fattore di decadimento α tale da non ridurre il peso degli item al di sotto del valore di threshold ($\alpha = \{0, 0.01, 0.05, 0.1\}$ per $t = 0.5$ e $\alpha = \{0, 0.01, 0.05\}$ per $t = 0.8$) siano caratterizzate da un andamento decrescente dei valori di precision e da un andamento quasi costante dei valori di recall; diversamente le run associate ad un valore di decadimento α tale da ridurre il peso degli item al di sotto del valore di threshold ($\alpha = \{0.5, 0.9\}$ per $t = 0.5$ e $\alpha = \{0.1, 0.5, 0.9\}$ per $t = 0.8$) sono caratterizzate da un aumento temporaneo dei valori di precision e di recall negli intervalli successivi all'esecuzione della fase di eliminazione degli item non significativi.

Questo particolare risultato può essere giustificato considerando che la fase di eliminazione degli item associati ad un peso inferiore al threshold



(a) Precision (b) Recall

Figura 4.19: Prestazioni del sistema con fattore di threshold t=0.5



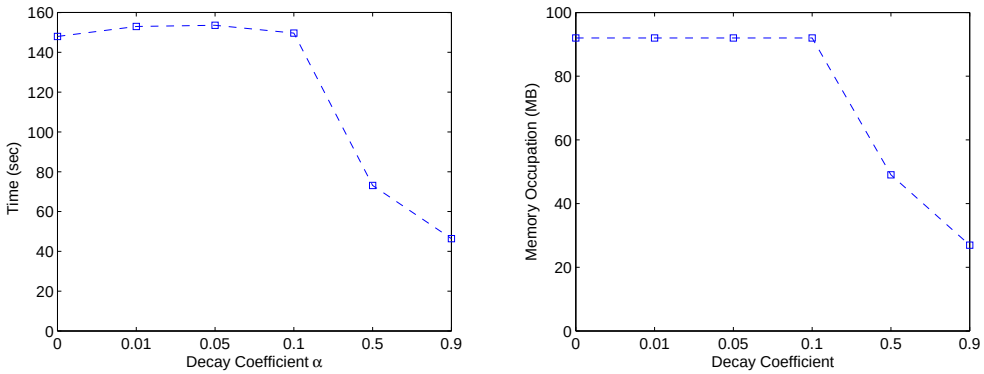
(a) Precision (b) Recall

Figura 4.20: Prestazioni del sistema con fattore di threshold t=0.8

impostato per il sistema opera come un filtro sull'insieme di item considerati per la generazione delle raccomandazioni finali. In un sistema privo del fattore di threshold, infatti, possono verificarsi situazioni nelle quali parte degli item contenuti nella lista di raccomandazione è associata ad un peso trascurabile ma ad un elevato numero di voti degli utenti del neighborhood; tali item, tipicamente, rappresentano un insieme di interessi espressi dagli utenti nel passato che non permettono di cogliere le attuali abitudini di voto dell'utente target. L'utilizzo del fattore di threshold, invece, permette di evitare uno scenario di questo tipo, eliminando progressivamente gli item non rappresentativi delle attuali abitudini di voto e limitando la ricerca degli item da raccomandare ad un sottoinsieme di preferenze recenti del neighborhood.

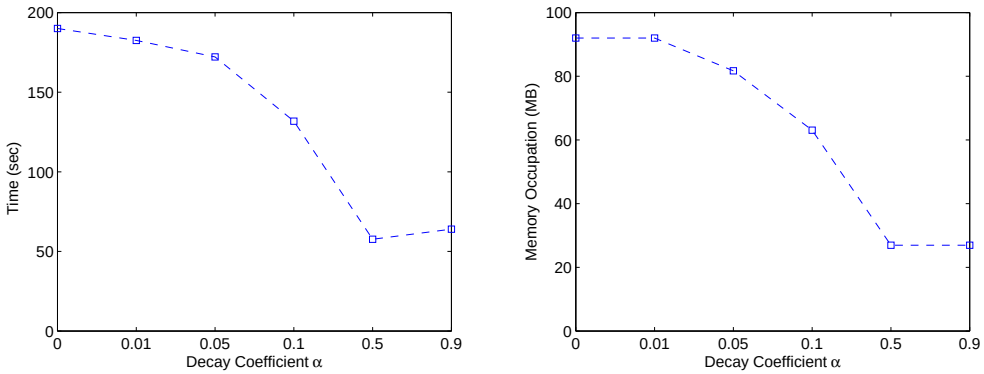
L'analisi comparativa dei valori di precision e dei valori di recall ottenuti nelle run associate a differenti valori di threshold permette anche di osservare come l'azione combinata del decadimento temporale e della fase di eliminazione degli item abbia un peso maggiore nel miglioramento dei valori di recall dell'algoritmo. La velocità con cui viene perfezionato il modello di interessi degli utenti grazie all'incremento del training set e all'utilizzo di un fattore di decadimento per i dati obsoleti, infatti, risulta inferiore alla velocità di riduzione del test set. Per tale motivo i valori di precision, calcolati ad ogni intervallo come la percentuale di item significativi presenti in una lista di dimensioni costanti, tendono a decrescere a causa della diminuzione del test set su cui vengono verificate le raccomandazioni generate; diversamente i valori di recall, calcolati ad ogni intervallo come la percentuale di item significativi presenti nella lista di raccomandazioni rispetto al totale di item significativi del sistema, tendono a crescere grazie alla riduzione del test set e quindi del numero complessivo di item significativi del sistema.

I grafici 4.21 e 4.22 mostrano i dati relativi ai tempi di esecuzione totali dell'algoritmo e alla memoria occupata rispetto ai differenti valori del parametro α scelto nelle simulazioni.



(a) Tempo di esecuzione (b) Memoria Occupata

Figura 4.21: Tempo di esecuzione totale e memoria occupata nelle run con fattore di threshold t=0.5



(a) Tempo di esecuzione (b) Memoria Occupata

Figura 4.22: Tempo di esecuzione totale e memoria occupata nelle run con fattore di threshold t=0.8

L'analisi dei risultati ottenuti permette di osservare come le migliori performance del sistema di raccomandazione siano associate a run

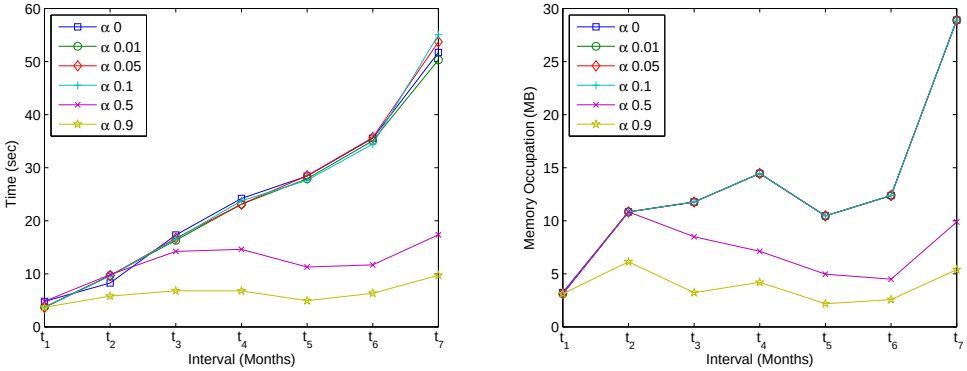
dell'algoritmo eseguite con un valore del fattore α tale da consentire l'esecuzione della fase di eliminazione degli item non significativi; in tale fase, infatti, vengono eliminati dalla lista di item votati da ciascun utente gli elementi associati ad un peso trascurabile, determinando così un risparmio dello spazio impiegato per il mantenimento di tali strutture dati e una conseguente riduzione dei tempi di analisi della lista per la ricerca degli item da raccomandare.

La scelta, perciò, di una combinazione ottimale dei parametri α e t consente di limitare, in scenari caratterizzati da un flusso continuo ed elevato di informazioni, la quantità di memoria e di tempo impiegati in ogni singolo intervallo temporale per la generazione delle raccomandazioni.

I grafici 4.23 e 4.24 mostrano l'andamento dei valori di memoria e di tempo impiegati nella generazione delle raccomandazioni al variare del fattore α scelto.

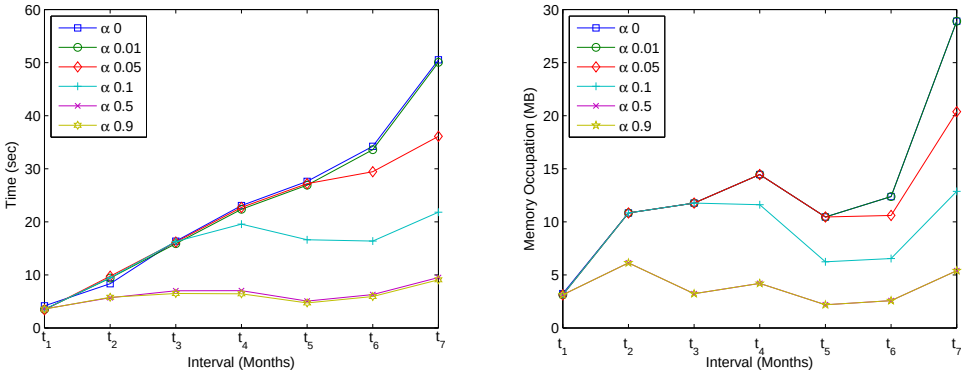
I risultati ottenuti dalle simulazioni permettono di osservare come le run associate ad un valore α tale da consentire l'esecuzione della fase di eliminazione degli item non significativi riescano a mantenere limitato il fattore di crescita dei tempi di computazione e di memoria impiegata; diversamente le run che non beneficiano della fase di eliminazione degli item presentano un comportamento analogo a quello dei sistemi di raccomandazione classici, nei quali la crescita dell'insieme di training dell'algoritmo si riflette in un aumento esponenziale delle risorse di memorizzazione impiegate.

L'utilizzo, perciò, di un algoritmo temporale basato sull'assegnazione di un peso decrescente alle preferenze degli utenti e sull'impiego di una funzione periodica di eliminazione dei dati obsoleti consente di limitare le risorse necessarie al mantenimento del sistema di raccomandazione e di ottimizzare



(a) Tempo di esecuzione (b) Memoria Occupata

Figura 4.23: Tempo di esecuzione e memoria occupata per intervallo temporale nelle run con fattore di threshold t=0.5



(a) Tempo di esecuzione (b) Memoria Occupata

Figura 4.24: Tempo di esecuzione e memoria occupata per intervallo temporale nelle run con fattore di threshold t=0.8

le prestazioni totali; tali vantaggi, tuttavia, vengono controbilanciati dalla minore affidabilità dell'algoritmo, che produce una serie di raccomandazioni caratterizzate da valori di precision e di recall inferiori rispetto all'esecuzione di un algoritmo di raccomandazione classico.

Tale tipologia di algoritmo, perciò, può trovare applicazione in sistemi caratterizzati da un'alta densità degli item nei quali le liste di raccomandazione vengono generate ad intervalli ravvicinati, come i siti di e-commerce: in tali sistemi, infatti, la minore affidabilità dei risultati prodotti ad ogni intervallo viene in parte bilanciata dalla creazione di un flusso di liste di raccomandazione che contengono alcuni item di interesse e dalla riduzione consistente dei tempi di computazione.

4.3.4 Valutazione complessiva dell'algoritmo di raccomandazione

Nella quarta fase di test sono state valutate le prestazioni complessive dell'algoritmo di raccomandazione *Twitter time-aware Recommendation System*. A questo scopo il sistema è stato inizializzato con i seguenti settaggi:

Numero di intervalli : l'intero dataset è stato suddiviso in un insieme di 8 intervalli di 30 giorni ciascuno; le raccomandazioni generate ad ogni intervallo i-esimo hanno utilizzato come *training set* l'insieme di voti contenuti negli intervalli $[t_0, t_i]$ e come *test set* l'insieme di voti contenuti negli intervalli $[t_{i+1}, t_n]$

Decadimento temporale α : i valori del fattore di decadimento α vengono scelti all'interno dell'insieme $A = \{0, 0.01, 0.05, 0.1, 0.5, 0.9\}$.

Fattore di threshold t : i valori del fattore di threshold t vengono scelti all'interno dell'insieme $T = [0.5, 0.8]$

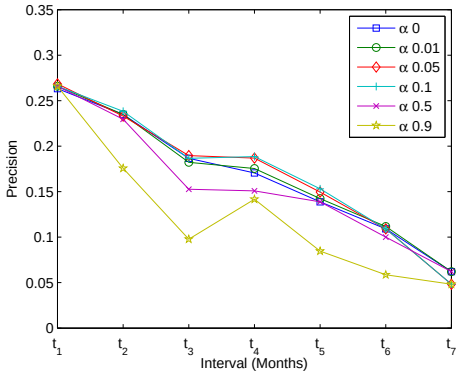
Parametri di clusterizzazione p e q : il parametro p viene settato a 8 e il parametro q viene settato a 20; ciascun utente, perciò, viene associato in ogni intervallo temporale a 20 nuovi cluster che rappresentano l'insieme degli interessi correnti.

I grafici 4.25 e 4.26 comparano i risultati in termini di precision e di recall ottenuti dall'algoritmo di raccomandazione che utilizza la fase di clusterizzazione degli utenti con i risultati associati al sistema privo di tale fase.

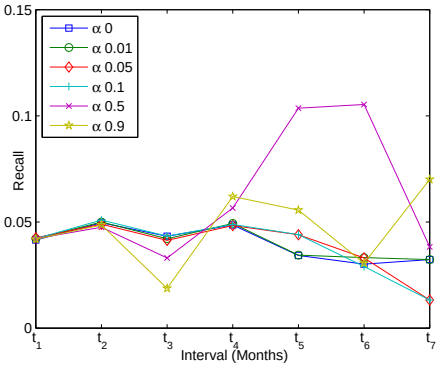
I dati ottenuti permettono di osservare come la procedura di clusterizzazione degli utenti determini un peggioramento delle performance totali del sistema di raccomandazione; l'assegnazione degli utenti ad un numero limitato di cluster, infatti, genera una semplificazione dei rapporti di similarità esistenti tra gli utenti, riducendo la capacità del sistema di cogliere relazioni significative per la generazione delle raccomandazioni.

La natura stessa dell'algoritmo di raccomandazione utilizzato introduce un fattore di variabilità nella precisione dei cluster generati per ciascun utente ad ogni intervallo temporale: l'insieme costante di cluster associato all'intervallo i-esimo, infatti, dipende da un numero di preferenze espresse dall'utente differente per ogni singolo individuo considerato. I cluster generati da un numero elevato di preferenze, perciò, risultano maggiormente rappresentativi degli interessi degli utenti rispetto ai cluster generati da un numero ridotto di prefereze, in grado di modellare solo parzialmente il profilo di voto dell'utente.

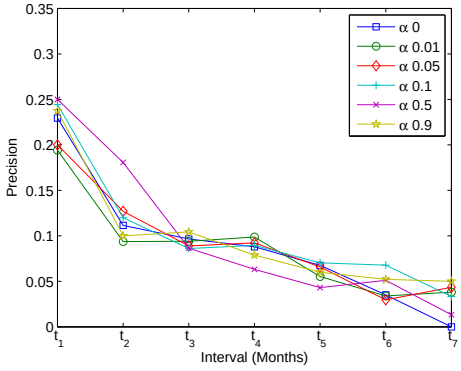
L'imprecisione associata a molti dei cluster presenti nel sistema, inoltre, causa la riduzione degli effetti del decadimento temporale α e del fattore di threshold t sulla precisione delle raccomandazioni generate; le curve associate ai differenti valori di α utilizzato, infatti, non presentano particolari



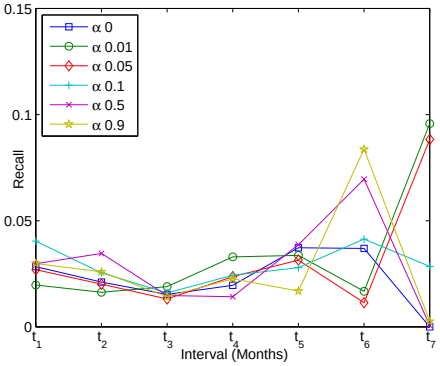
(a) Precision



(b) Recall

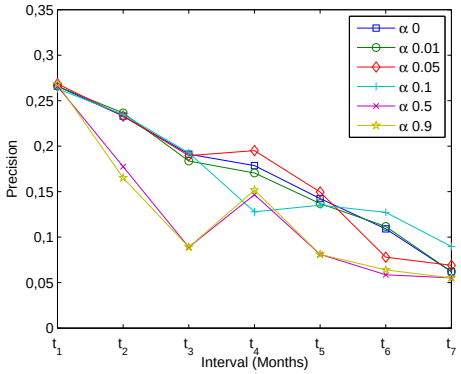


(c) Precision MinHash

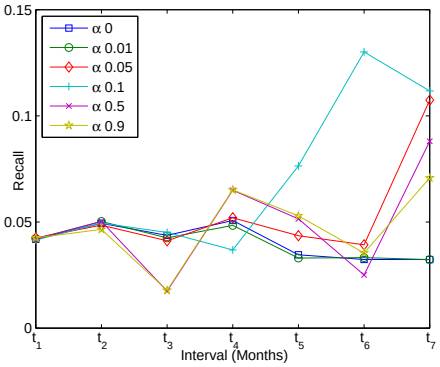


(d) Recall MinHash

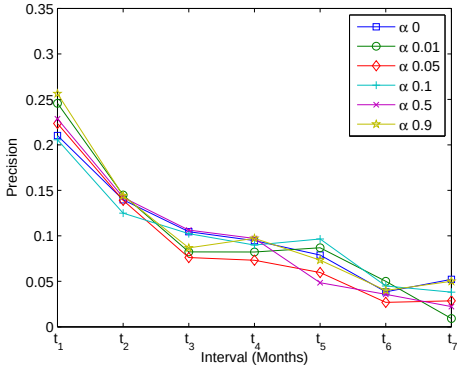
Figura 4.25: Comparazione tra le prestazioni del sistema nelle run eseguite con e senza l'utilizzo dell'algoritmo MinHash associate ad un fattore di threshold pari a 0.5



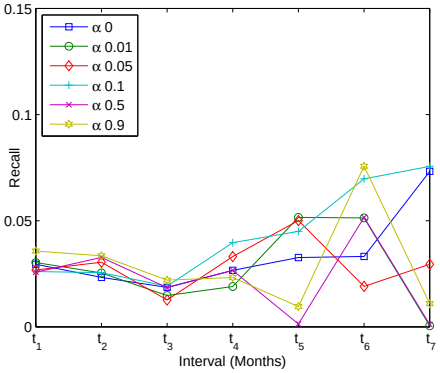
(a) Precision



(b) Recall



(c) Precision MinHash



(d) Recall MinHash

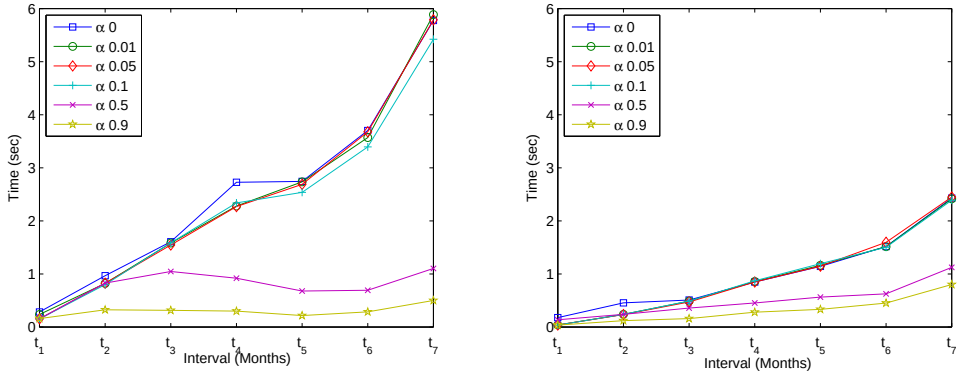
Figura 4.26: Comparazione tra le prestazioni del sistema nelle run eseguite con e senza l'utilizzo dell'algoritmo MinHash associate ad un fattore di threshold pari a 0.8

variazioni tra di loro, come è possibile osservare nei grafici 4.25c e 4.26c. Diversamente i valori di recall risultano ancora influenzati dalla presenza della fase di eliminazione dal sistema degli item e dei cluster non significativi; l'azione svolta dalla diminuzione progressiva dell'insieme di item significativi presenti nel sistema, perciò, riesce a bilanciare parzialmente l'inesattezza delle raccomandazioni generate.

L'introduzione della fase di clusterizzazione degli utenti influisce direttamente sulle prestazioni della fase di *compare* dell'algoritmo: nei sistemi privi di tale fase, infatti, l'agoritmo di similarità opera un confronto tra insiemi di preferenze le cui dimensioni variano da individuo ad individuo; nel sistema *Twitter time-aware Recommendation System*, invece, l'algoritmo di similarità effettua un confronto su insiemi di cluster di dimensioni predefinite e limitate, garantendo così un tempo di esecuzione costante per ciascuna coppia di utenti.

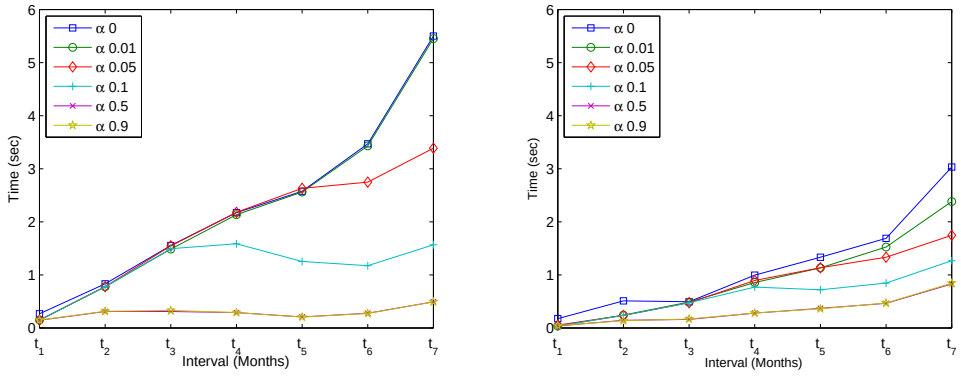
I grafici 4.27 e 4.28 mostrano i risultati relativi ai tempi di esecuzione della fase di compare dell'algoritmo di raccomandazione al variare del fattore di threshold t in uno scenario di funzionamento privo di algoritmo di clusterizzazione e in uno scenario di funzionamento che utilizza l'algoritmo MinHash.

I dati ottenuti evidenziano come l'introduzione dell'algoritmo di clusterizzazione nel sistema di raccomandazione determini una riduzione significativa dei tempi di computazione dei coefficienti di similarità tra gli utenti, in particolare per quelle run il cui valore di decadimento α non permette l'esecuzione della fase di eliminazione degli item; in tale scenario, perciò, l'aumento nel tempo dei valori rilevati dipende direttamente dalla crescita dell'insieme di utenti attivi nel sistema per i quali viene periodicamente calcolato il coefficiente di similarità.



(a) Sistema senza utilizzo MinHash (b) Sistema con utilizzo MinHash

Figura 4.27: Confronto dei tempi di esecuzione della fase di compare nelle run eseguite con e senza l'utilizzo dell'algoritmo MinHash associate ad un fattore di threshold pari a 0.5



(a) Sistema senza utilizzo MinHash (b) Sistema con utilizzo MinHash

Figura 4.28: Confronto dei tempi di esecuzione della fase di compare nelle run eseguite con e senza l'utilizzo dell'algoritmo MinHash associate ad un fattore di threshold pari a 0.8

Il sistema *Twitter time-aware Recommendation System* nella sua forma completa non riesce a garantire prestazioni ottimali sul piano dei valori di precision/recall ottenuti o sul piano dei tempi di computazione impiegati; la scelta di una combinazione ottimale dei valori del fattore di decadimento α e del fattore di threshold t, infatti, riesce a garantire prestazioni analoghe o migliori, senza l'overhead introdotto dal calcolo dei cluster e dalla loro memorizzazione nelle strutture dati.

Valutazione del sistema predittivo

Una fase di test del sistema ha avuto come obiettivo la valutazione della capacità predittiva dell'algoritmo implementato. A tale scopo il sistema di raccomandazione è stato utilizzato per generare, ad ogni intervallo t_k, una predizione di voto per gli item votati nei successivi intervalli $[t_{k+1}, t_n]$.

La predizione associata a ciascun item viene determinata mediante l'utilizzo di uno score calcolato attraverso una media pesata dei voti assegnati all'item dagli utenti contenuti nel neighborhhood:

$$pred(u,i) = \frac{\sum_{n \in Users} sim(u,n) \cdot r_{ni}}{\sum_{n \in Users} sim(u,n)} = \begin{cases} 1 & \text{se } pred(u,i) > 0 \\ -1 & \text{se } pred(u,i) < 0 \end{cases} \tag{4.26}$$

L'errore commesso dal sistema di predizione è stato calcolato utilizzando due differenti strategie:

RMSE : ad ogni intervallo temporale t_k il valore RMSE viene calcolato sulle predizioni generate solo all'interno dell'intervallo, secondo la formula:

$$RMSE = \sqrt{\frac{\sum_{i \in P_{t_k}} (\hat{r}_{ui} - r_{ui})^2}{|P_{t_k}|}} \tag{4.27}$$

dove P_{t_k} è l'insieme di predizioni generate nell'intervallo k-esimo.

In questo caso il valore RMSE fornisce una valutazione dell'errore istantaneo commesso dal sistema.

TA-RMSE : ad ogni intervallo temporale t_k il valore TA-RMSE viene calcolato sulle predizioni generate nella finestra temporale $[t_0, t_k]$, secondo la formula:

$$TA - RMSE_t = \sqrt{\frac{\sum_{i \in P_{[t_0,t_k]}} (\hat{r}_{ui} - r_{ui})^2}{\left|P_{[t_0,t_k]}\right|}} \tag{4.28}$$

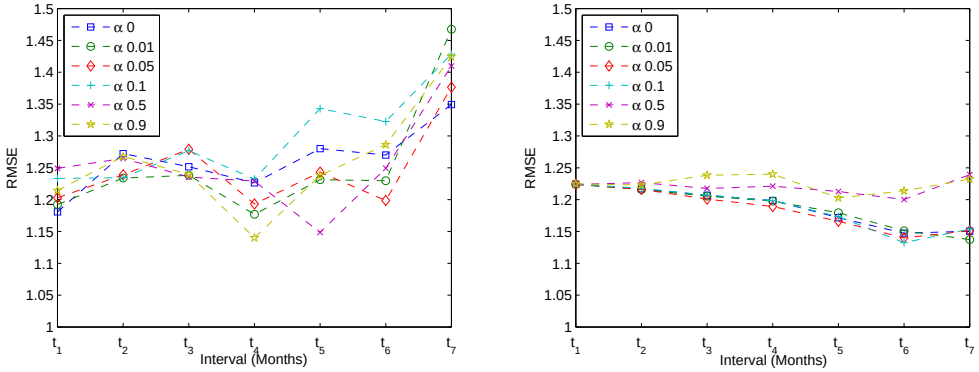
dove $P_{[t_0,t_k]}$ è l'insieme di predizioni generate fino all'intervallo t_k.

In questo caso il valore TA-RMSE fornisce una valutazione della media temporale dell'errore commesso dal sistema.

I grafici 4.29, 4.30 mostrano l'andamento dei valori del RMSE e del TA-RMSE associati ad un sistema caratterizzato da un fattore di threshold pari a 0.5 in due particolari scenari di funzionamento: nel primo l'algoritmo di raccomandazione utilizza la fase di clusterizzazione degli utenti per il calcolo delle similarità mentre nel secondo tale fase viene eliminata.

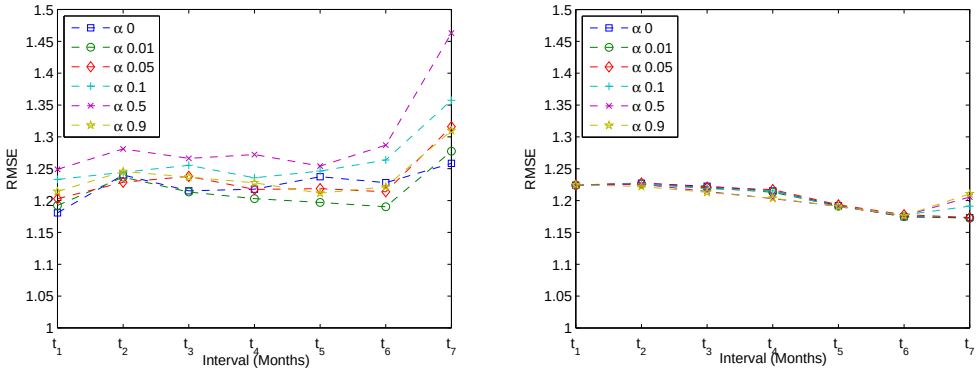
I risultati ottenuti mostrano come l'introduzione dei cluster nel sistema di raccomandazione causi un peggioramento delle prestazioni del sistema, determinando un aumento dell'errore commesso nella generazione delle predizioni. L'analisi comparativa dei due grafici, in particolare, mostra come il progressivo miglioramento della capacità predittiva del sistema presente nelle run prive della fase di clusterizzazione sia assente nelle run eseguite con l'utilizzo dell'algoritmo MinHash.

In tale scenario, infatti, la minore precisione dei valori di similarità associati a ciascun utente causa un andamento variabile dei valori di RMSE calcolati, con una tendenza all'aumento dell'errore commesso.



(a) RMSE con l'utilizzo di MinHash (b) RMSE

Figura 4.29: Confronto dei valori di RMSE nelle run eseguite con e senza l'utilizzo dell'algoritmo MinHash associate ad un fattore di threshold pari a 0.5

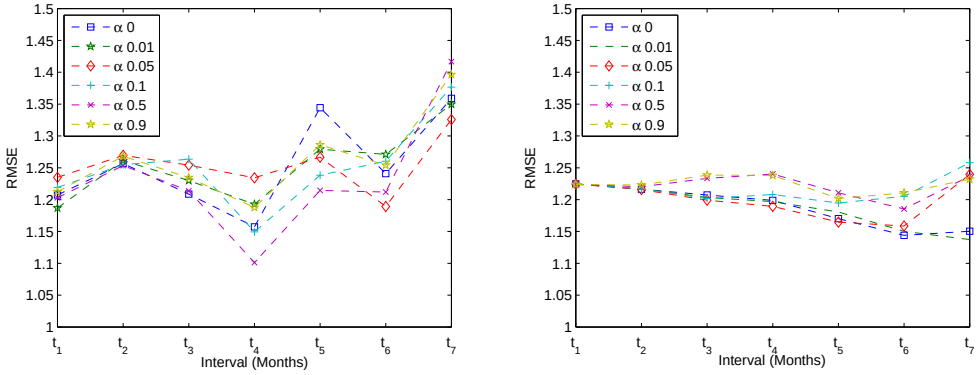


(a) TA-RMSE con l'utilizzo di MinHash (b) TA-RMSE

Figura 4.30: Confronto dei valori di TA-RMSE nelle run eseguite con e senza l'utilizzo dell'algoritmo MinHash associate ad un fattore di threshold pari a 0.5

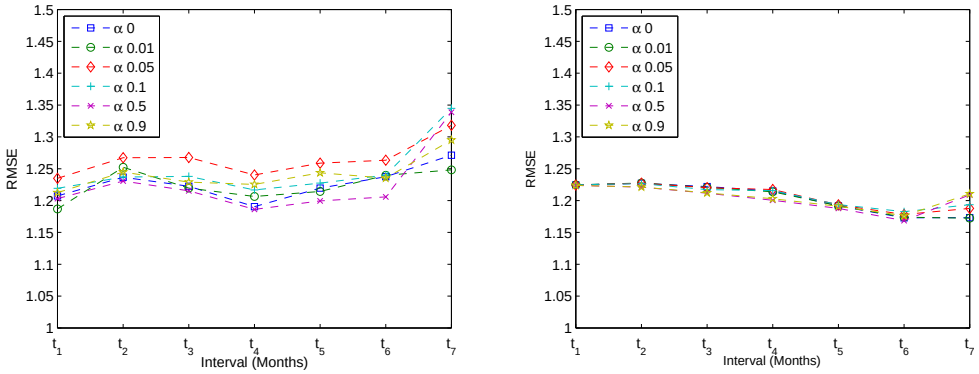
Risultati analoghi sono visibili nei grafici 4.31, 4.32, che rappresentano le run dell'algoritmo eseguite con un fattore di threshold pari a 0.8.

I dati a disposizione mostrano come l'utilizzo di un fattore di threshold più elevato, e quindi la conseguente eliminazione di un numero maggiore di item e di cluster non significativi dal sistema, permetta di ridurre le differenze esistenti tra i risultati associati ai diversi valori di α nelle run eseguite con l'utilizzo della fase di clusterizzazione. In tale scenario, infatti, l'eliminazione progressiva dei cluster non significativi presente nella maggior parte delle run forza il sistema a calcolare i valori di similarità per ciascun utente sul sottoinsieme di preferenze recenti, limitando in maniera significativa il peso dei cluster generati negli intervalli passati e fornendo, quindi, un modello istantaneo degli interessi degli utenti.



(a) RMSE con l'utilizzo di MinHash (b) RMSE

Figura 4.31: Confronto dei valori di RMSE nelle run eseguite con e senza l'utilizzo dell'algoritmo MinHash associate ad un fattore di threshold pari a 0.8



(a) TA-RMSE con l'utilizzo di MinHash (b) TA-RMSE

Figura 4.32: Confronto dei valori di TA-RMSE nelle run eseguite con e senza l'utilizzo dell'algoritmo MinHash associate ad un fattore di threshold pari a 0.8

Capitolo 5

Conclusioni

Twitter time-aware Recommendation System è un sistema di raccomandazione incrementale che presenta diversi elementi innovativi rispetto ai sistemi di raccomandazione classici:

1. il sistema di raccomandazione è progettato secondo una tipologia di architettura ibrida: parte della capacità computazionale del sistema viene trasferita ai nodi peer, che si occupano di realizzare una fase di pre-processamento dei dati. Il carico di lavoro sul server remoto, quindi, viene in parte ridotto, migliorando così le performance totali del sistema.

2. l'algoritmo di raccomandazione implementato integra in un'unico sistema alcune caratteristiche già utilizzate nei sistemi di raccomandazione, come la clusterizzazione degli utenti e il decadimento temporale dei dati presenti nel sistema, con un aspetto originale rappresentato dall'utilizzo di una combinazione delle preferenze positive e negative degli utenti per la generazione dei valori di similarità.

L'insieme di test svolti sul sistema per valutare l'impatto di ciascun elemento sulle prestazioni totali ha permesso di evidenziare il contributo positivo fornito dagli elementi comunemente utilizzati nei sistemi di raccomandazione; l'utilizzo di una fase di clusterizzazione degli utenti e di un fattore di decadimento temporale applicato ai dati raccolti nelle iterazioni precedenti, infatti, permette di limitare la crescita delle dimensioni delle strutture dati necessarie agli algoritmi di raccomandazione e di conseguenza di migliorare i tempi di computazione complessivi. L'utilizzo delle preferenze negative degli utenti nella fase di generazione dei valori di similarità, invece, non riesce a fornire un contributo positivo all'interno del processo di raccomandazione; l'impatto dei voti negativi sui valori di similarità, rappresentato dalla definizione di un fattore di *dissimilarità* tra utenti, infatti, è visibile in uno scenario di funzionamento del sistema di raccomandazione tipicamente non adottato, che prevede l'analisi delle informazioni associate all'intero set di utenti del sistema per generare le raccomandazioni.

I test svolti sul sistema di raccomandazione per valutare l'andamento delle prestazioni nel tempo, invece, hanno richiesto l'adozione di un nuovo metodo di validazione dei risultati in grado di tracciare uno schema rappresentativo dell'evoluzione nei differenti intervalli temporali dei singoli valori osservati. La nuova strategia di validazione adottata non ha permesso la comparazione dei risultati ottenuti con quelli associati ad altri sistemi di tipo temporale; l'insieme dei test eseguiti, perciò, è stato utilizzato per identificare un pattern di evoluzione temporale relativo alle prestazioni del sistema nei diversi intervalli.

La particolare struttura del dataset utilizzato nei test, tuttavia, caratterizzato da utenti con una frequenza di voto estremamente variabile, ha limitato in parte la possibilità di definire un modello di evoluzione temporale

degli interessi espressi, influenzando, così, i risultati ottenuti; i dati a disposizione, infatti, hanno permesso di evidenziare esclusivamente l'impatto positivo che l'introduzione del fattore temporale ha sulle dimensioni delle strutture dati memorizzate, fornendo invece dati di difficile interpretazione relativi ai valori di precision e di recall misurati.

Complessivamente, quindi, *Twitter time-aware Recommendation System*, nella sua variante di funzionamento che prevede l'utilizzo della matrice di similarità associata alle sole preferenze positive, rappresenta un sistema di raccomandazione caratterizzato da un'architettura e da un insieme di performance che lo rendono adatto all'utilizzo in scenari caratterizzati da un elevato flusso di informazioni da processare, quali i social network, nei quali l'interesse primario è generare una lista estesa di raccomandazioni contenente un sottoinsieme di item di interesse limitando le risorse impiegate nella fase di calcolo.

La parziale diminuzione della precisione delle raccomandazioni generate, infatti, viene bilanciata dalla riduzione sensibile dei tempi di calcolo totali e delle risorse impiegate dal sistema per la memorizzazione dei dati di interesse; tale aspetto, perciò, rappresenta il punto di forza dell'algoritmo di raccomandazione progettato e rende la sua applicazione consigliata in sistemi nei quali il rate di ingresso di nuovi dati è elevato ed esiste la necessità di produrre un flusso continuo di raccomandazioni al minimo costo.

Sviluppi futuri

Una delle limitazioni incontrate durante questo lavoro di tesi è rappresentata dall'esecuzione dei test su un dataset modificato la cui struttura non riesce ad emulare l'interazione di un utente con il social network Twitter. Tale sistema, infatti, è un sistema altamente dinamico nel quale ciascun utente

visualizza e produce nuovi tweet ad un rate elevato, determinando così un flusso di informazioni costanti ideali per l'aggiornamento incrementale dei valori di similarità calcolati.

Il dataset utilizzato, invece, è caratterizzato da un insieme di utenti che presentano un'interazione limitata con il sistema e che quindi generano un insieme di voti insufficiente per la costruzione di un profilo degli interessi affidabile; la distribuzione temporale variabile dei voti nei singoli intervalli, inoltre, penalizza fortemente gli elementi caratteristici del nostro algoritmo che dipendono dalla presenza di un pattern di evoluzione temporale per gli interessi espressi, come la fase di clusterizzazione o la generazione di una matrice di similarità incrementale.

Risulta interessante, quindi, valutare il comportamento del sistema di raccomandazione in uno scenario di funzionamento reale, attraverso l'esecuzione di una fase di test estesa successiva al rilascio dell'estensione *TweetRate* o attraverso la costruzione di un dataset sintetico che riesca ad emulare il comportamento degli utenti di Twitter. I dati ottenuti da queste nuove tipologie di dataset, infatti, permettono di identificare in maniera più chiara la natura adattiva dell'algoritmo sviluppato, il cui funzionamento varia al variare delle strutture dati sottostanti.

Un possibile scenario di ricerca futuro è rappresentato dall'estensione del Sistema di Raccomandazione sviluppato attraverso l'introduzione della possibilità di esprimere un voto appartenente ad un insieme di valori discreti; la sostituzione dei voti *mi piace*, *non mi piace* con un insieme di valori più grande, infatti, permette di cogliere in maniera più specifica le differenze presenti tra gli interessi dei singoli utenti, costruendo un modello delle preferenze più complesso.

Un'altro scenario di ricerca futuro è rappresentato dalla possibilità di

integrare nel Sistema di Raccomandazione elementi di valutazione legati alla struttura del grafo sociale che caratterizza le connessioni tra gli utenti; in particolare risulta interessante introdurre nell'algoritmo di generazione delle raccomandazione un valore di *significance* che permetta di assegnare ai voti di ciascun neighbors un peso proporzionale al numero di connessioni condivise nel grafo sociale con l'utente target.

Appendice A

Implementazione del Sistema TweetRate

In questa appendice vengono mostrati gli *UML diagrams* e i *Sequence diagrams* associati all'estensione Chrome *TweetRate* e al simulatore *TweetRateSimulator* realizzato per testare il sistema di raccomandazione. I diagrammi permettono di analizzare la struttura delle classi realizzate per il simulatore e il flusso di messaggi e di chiamate a funzione che caratterizzano sia l'estensione Chrome che il simulatore Java.

A.1 TweetRate

Installazione Plugin nuovo utente

La figura A.1 rappresenta il sequence diagram associato all'installazione dell'estensione da parte di un nuovo utente. L'evento di installazione genera una sequenza di azioni il cui obiettivo è quello di inizializzare le strutture dati locali utilizzate per la memorizzazione del profilo utente e dei voti già espressi, di effettuare la richiesta del token di utilizzo delle API di Twitter e di inizializzare correttamente l'interfaccia utente aggiungendo i pulsanti di voto.

- Fase di inizializzazione strutture dati locali:

    1. a seguito dell'evento di installazione l'*EventPage*, che rappresenta il livello di business dell'estensione, si interfaccia con il DBMS WebSQL per la creazione delle tabelle locali di profilo e dei voti.
    2. il *Plugin* invia il messaggio di login all'*EventPage*.
    3. l'*EventPage* contatta il server OAuth per ottenere il token di utilizzo delle API di Twitter.
    4. l'*EventPage* utilizza il token ricevuto per ottenere dai server di Twitter le informazioni di profilo associate all'utente; in seguito memorizza il profilo sul server SQL remoto e sul DBMS locale, insieme al token di accesso alle API.

- Fase di inizializzazione dell'interfaccia:

    5. il *Plugin* modifica l'interfaccia standard di Twitter, aggiungendo i pulsanti di voto ad ogni tweet visualizzato nella bacheca del'utente.

- Fase di voto:

  6. l'utente vota un tweet sulla propria bacheca.
  7. il *Plugin* inserisce il voto nel DBMS locale; in seguito invia all'*EventPage* il voto tramite un messaggio.
  8. l'*EventPage* preleva dai server Twitter i metadati associati al tweet votato; in seguito memorizza il voto e i metadati nel server remoto.

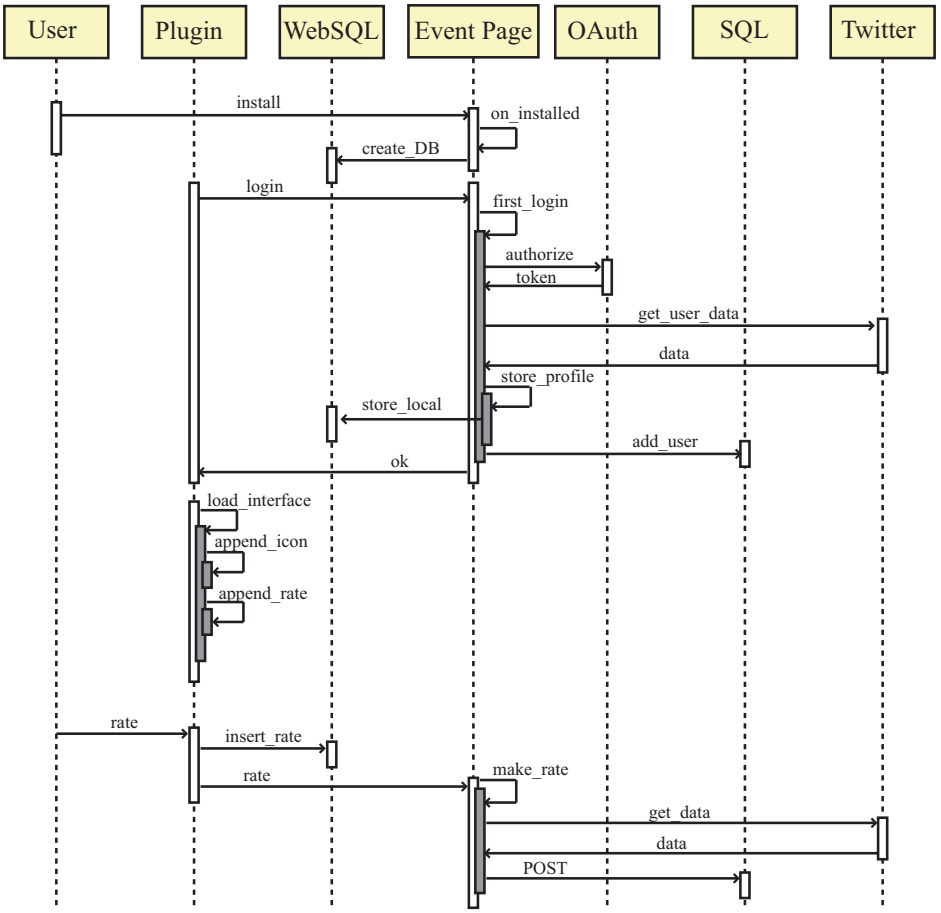


Figura A.1: Sequence Diagram login nuovo utente

Login utente già registrato

La figura A.2 rappresenta il sequence diagram associato al login in Twitter di un utente che ha già installato ed utilizzato l'estensione *TweetRate* in passato. L'evento di login genera una sequenza di azioni il cui obiettivo è quello di inviare al database remoto i dati memorizzati nel database locale ed inizializzare l'interfaccia utente aggiungendo i pulsanti di voto:

- Fase di login:
  1. l'utente effettua il login a Twitter.
  2. il *Plugin* invia il messaggio di login all'*EventPage*.
  3. l'*EventPage* preleva i dati memorizzati localmente nel WebSQL (voti non ancora inviati al DBMS remoto) ed effettua la procedura di invio al DBMS remoto.
- Fase di inizalizzazione dell'interfaccia:
  4. il *Plugin* invia un messaggio all'*EventPage* per prelevare gli ultimi voti espressi dall'utente che devono essere visualizzati sulla bacheca di Twitter.
  5. l'*EventPage* preleva dal DBMS remoto gli ultimi voti dell'utente e li invia al *Plugin*.
  6. il *Plugin* modifica l'interfaccia standard di Twitter, aggiungendo i pulsanti di voto ad ogni tweet visualizzato nella bacheca del'utente ed evidenziando i tweet già votati.
- Fase di voto:
  7. l'utente vota un tweet sulla propria bacheca.

8. il *Plugin* inserisce il voto nel DBMS locale; in seguito invia all'*EventPage* il voto tramite un messaggio.
9. l'*EventPage* preleva dai server Twitter i metadati associati al tweet votato; in seguito memorizza il voto e i metadati nel server remoto.

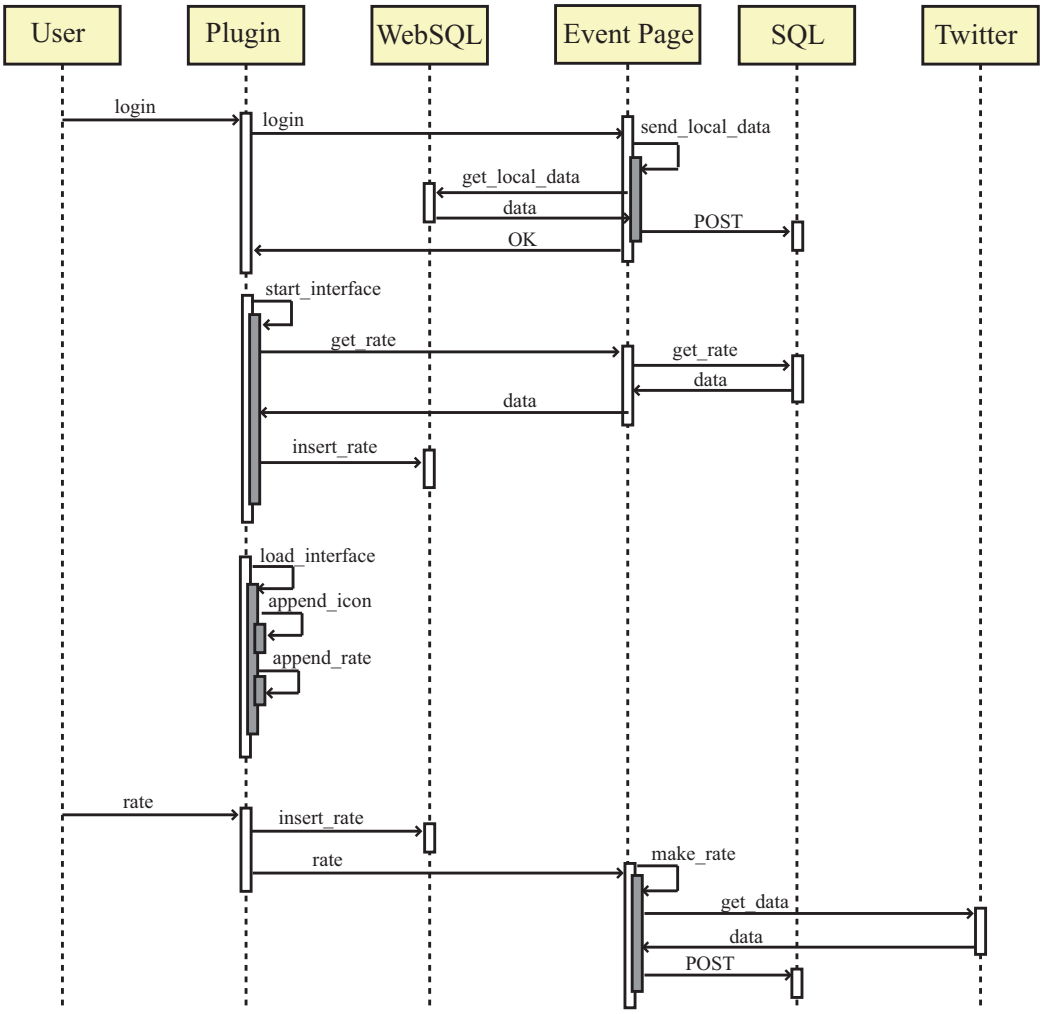


Figura A.2: Sequence Diagram login utente già registrato

A.2 TweetRateSimulator

La figura A.3 rappresenta il diagramma UML del simulatore *TweetRateSimulator*; in particolare il diagramma contiene le principali classi Java utilizzate negli algoritmi di lettura dei dati, di calcolo delle similarità e di raccomandazione/predizione. Le classi accessorie, come le classi di ordinamento o come particolari strutture dati di appoggio, sono state omesse per migliorare la leggibilità.

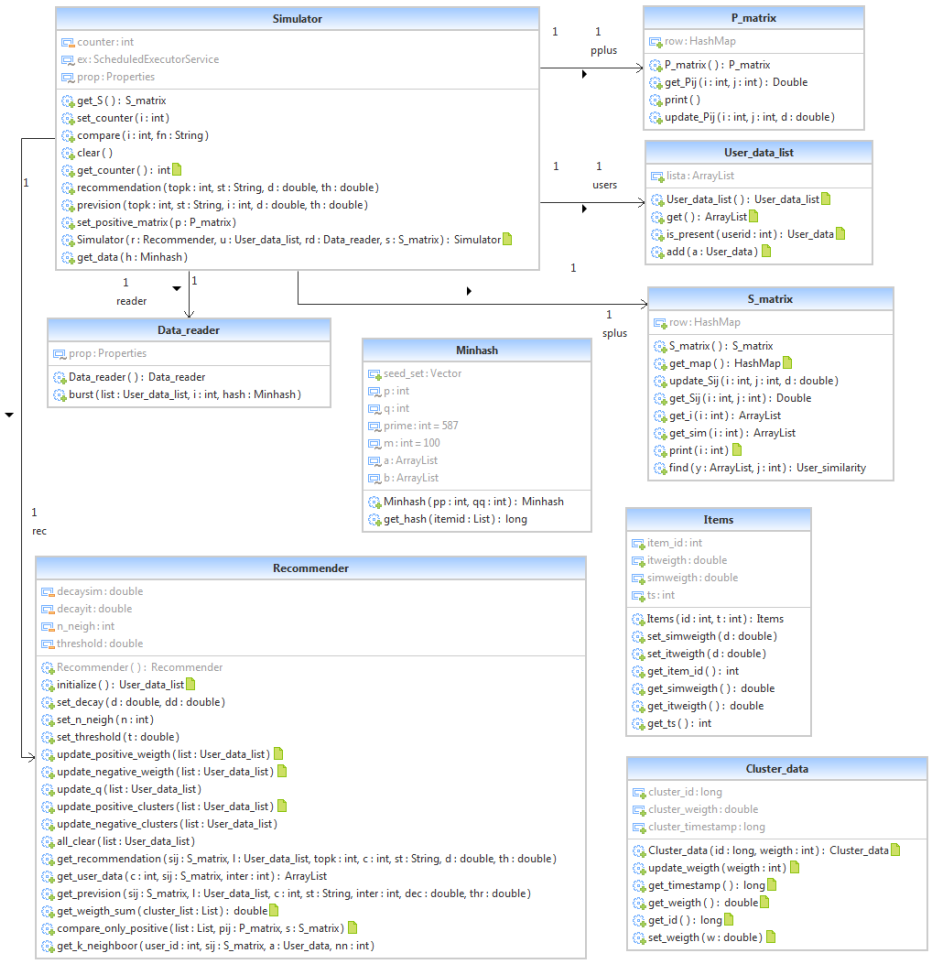


Figura A.3: Diagramma UML del simulatore TweetRateSimulator

Fase di inizializzazione del simulatore

La figura A.4 rappresenta il sequence diagram associato alla fase di inizializzazione delle strutture dati e delle variabili utilizzate all'interno del simulatore Java *TweetRateSimulator*.

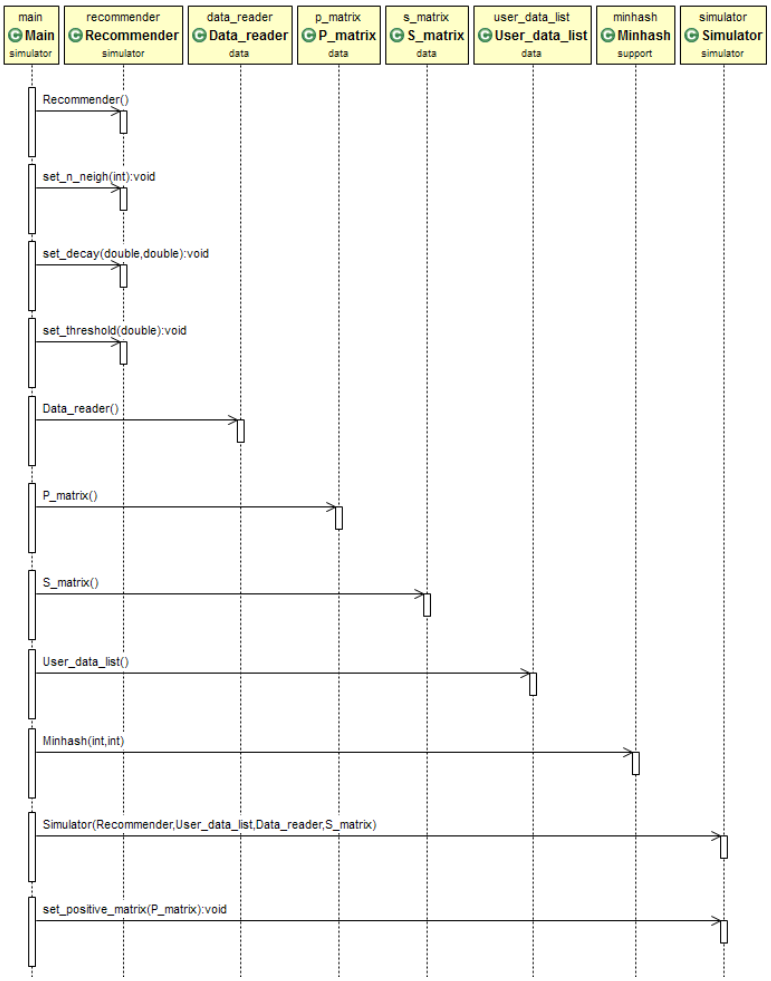


Figura A.4: Sequence Diagram della fase di inizializzazione del simulatore TweetRateSimulator

Fase di compare

La figura A.5 rappresenta il sequence diagram associato alla fase di generazione dei valori di similarità tra gli utenti:

1. la funzione *Data_reader.burst()* si occupa di leggere l'insieme dei voti contenuti nell'intervallo corrente.

2. la funzione *MinHash.get_hash()* genera, per ciascun insieme di item votati, i relativi cluster di appartenenza.

3. le funzioni *Recommender.update_positive_weigth()* e *Recommender.update_negative_weigth()* aggiornano i valori di peso degli item votati nelle iterazioni precedenti.

4. la funzione *Recommender.compare_only_positive()* aggiorna i valori di similarità della matrice *S_matrix* attraverso l'insieme di cluster generati nell'intervallo corrente.

5. le funzioni *Recommender.update_q()* e *Recommender.update_positive_clusters()* aggiornano i valori di q associati a ciascun utente e la lista di cluster generati durante l'intero periodo di osservazione del sistema.

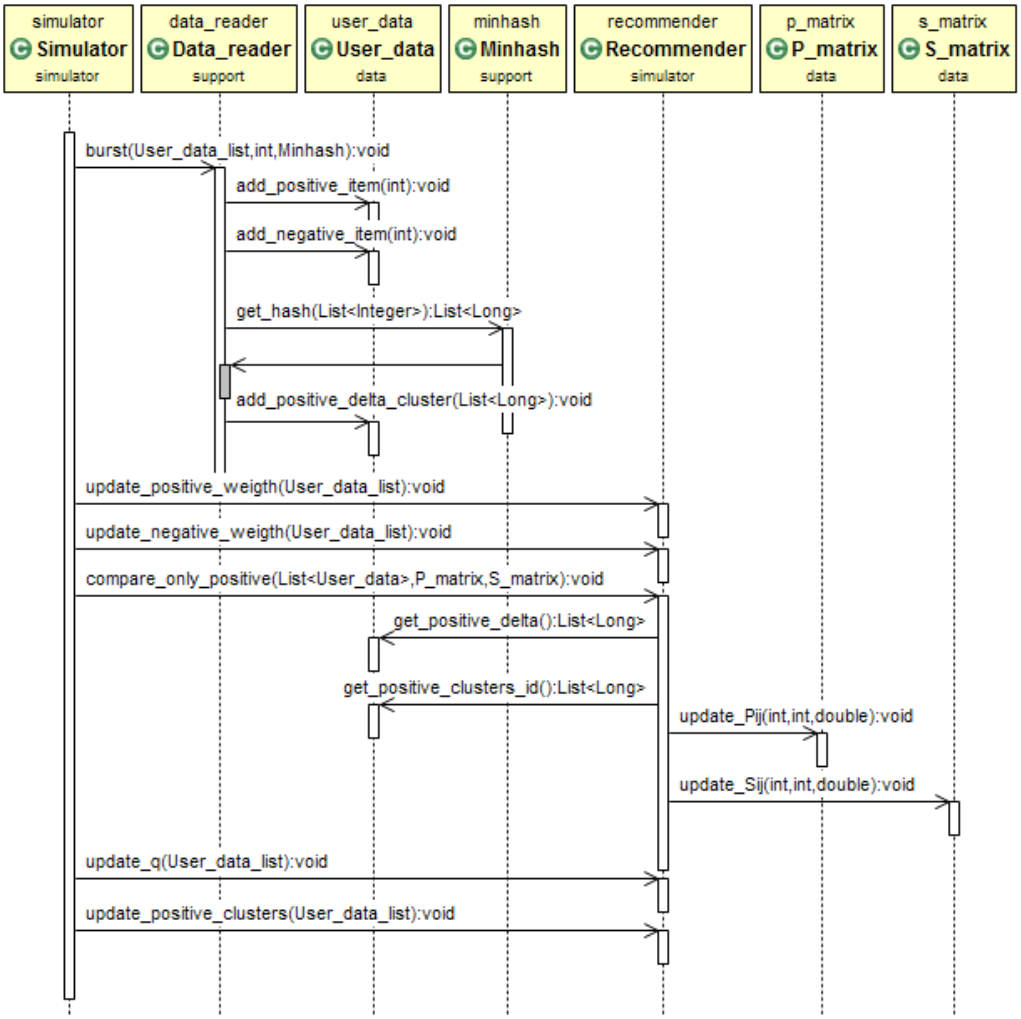


Figura A.5: Sequence Diagram della fase di compare del simulatore TweetRateSimulator

Fase di recommend

La figura A.6 rappresenta il sequence diagram associato alla fase di generazione delle raccomandazioni:

1. la funzione *S_Matrix.get_map()* preleva dalla matrice di similarità *S_Matrix* la riga associata all'utente corrente.

2. la funzione *Recommender.get_k_neighboor* si occupa di generare la lista di utenti *nearest-neighbors* relativi all'utente corrente; in particolare viene eseguita la funzione di ordinamento *QuickSort.sort()* per ordinare la lista calcolata nell'iterazione precedente e viene eseguita la funzione *InsertiorSort.insertionSort()* per inserire eventuali utenti con valori di similarità elevati.

3. le funzioni *User_data.get_positive_items()*, *User_data.get_negative_items()*, *User_data.get_old_pos()* e *User_data.get_old_neg()* vengono utilizzate per identificare l'insieme di item votati dagli utenti simili ma non ancora votati dall'utente corrente.

4. la funzione *QuickSelectItem.quickselectTopK()* viene utilizzata per ordinare gli item in base allo score associato attraverso l'applicazione della formula di raccomandazione; vengono, quindi, raccomandati i *topk* item.

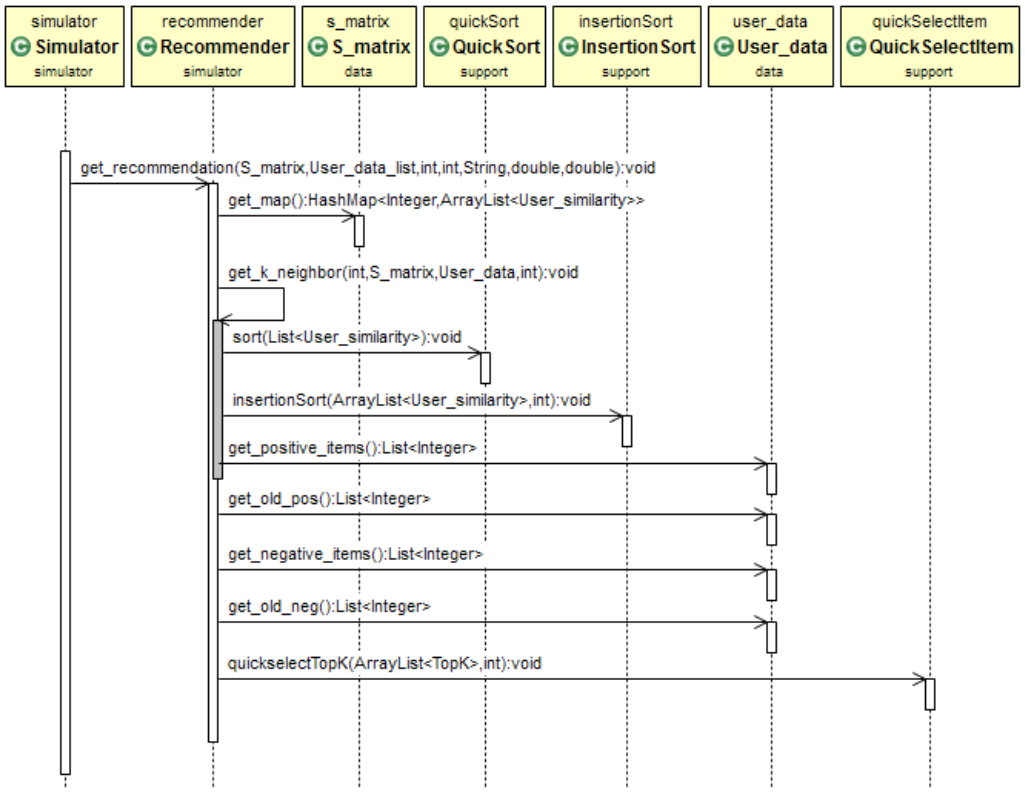


Figura A.6: Sequence Diagram della fase di recommend del simulatore TweetRateSimulator

Bibliografia

[1] Meenakshi Sharma and Sandeep Mann. A survey of recommender systems: Approaches and limitations.

[2] Dhoha Almazro, Ghadeer Shahatah, Lamia Albdulkarim, Mona Kherees, Romy Martinez, and William Nzoukou. A survey paper on recommender systems. *arXiv preprint arXiv:1006.5278*, 2010.

[3] Robin Burke, Alexander Felfernig, and Mehmet H Göker. Recommender systems: An overview. *AI Magazine*, 32(3):13–18, 2011.

[4] Jian-guo Liu, M Chen, Jian-chi Chen, Fei Deng, H Zhang, Z Zhang, and T Zhou. Recent advances in personal recommender systems. *International journal of information and systems sciences*, 5:230–247, 2009.

[5] Emmanouil Vozalis and Konstantinos G Margaritis. Analysis of recommender systems algorithms. In *Proceedings of the 6th Hellenic European Conference on Computer Mathematics and its Applications (HERCMA-2003), Athens, Greece*, volume 2003, 2003.

[6] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-

art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[7] Uri Hanani, Bracha Shapira, and Peretz Shoval. Information filtering: Overview of issues, research and systems. *User Modeling and User-Adapted Interaction*, 11(3):203–259, 2001.

[8] Òscar Celma. The long tail in recommender systems. In *Music Recommendation and Discovery*, pages 87–107. Springer, 2010.

[9] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[10] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[11] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

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

[13] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

[14] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173, 2011.

[15] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[16] David Maltz and Kate Ehrlich. Pointing the way: active collaborative filtering. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 202–209. ACM Press/Addison-Wesley Publishing Co., 1995.

[17] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[18] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[19] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of machine learning*, pages 829–838. Springer, 2010.

[20] Lin Guo and Qin Ke Peng. A combinative similarity computing measure for collaborative filtering. *Applied Mechanics and Materials*, 347:2919–2925, 2013.

[21] Michael Leben. Applying item-based and user-based collaborative filtering on the netflix data.

[22] Manos Papagelis and Dimitris Plexousakis. Qualitative analysis of user-based and item-based prediction algorithms for recommendation agents. *Engineering Applications of Artificial Intelligence*, 18(7):781–789, 2005.

[23] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 1999.

[24] Mustansar Ghazanfar and Adam Prugel-Bennett. Novel significance weighting schemes for collaborative filtering: Generating improved recommendations in sparse environments. 2010.

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

[26] Jon Herlocker, Joseph A Konstan, and John Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information retrieval*, 5(4):287–310, 2002.

[27] Alan Said, Ben Fields, Brijnesh J Jain, and Sahin Albayrak. User-centric evaluation of a k-furthest neighbor collaborative filtering recommender algorithm. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 1399–1408. ACM, 2013.

[28] Koji Miyahara and Michael J Pazzani. Collaborative filtering with the simple bayesian classifier. In *PRICAI 2000 Topics in Artificial Intelligence*, pages 679–689. Springer, 2000.

[29] Daniel Billsus and Michael J Pazzani. Learning collaborative information filters. In *ICML*, volume 98, pages 46–54, 1998.

[30] Thomas Hofmann and Jan Puzicha. Latent class models for collaborative filtering. In *IJCAI*, volume 99, pages 688–693, 1999.

[31] Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[32] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008.

[33] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[34] Robert M Bell and Yehuda Koren. Improved neighborhood-based collaborative filtering. In *KDD Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. sn, 2007.

[35] Zan Huang, Hsinchun Chen, and Daniel Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):116–142, 2004.

[36] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 10:623–656, 2009.

[37] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[38] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[39] Tiffany Ya Tang, Pinata Winoto, and Keith CC Chan. On the temporal analysis for improved hybrid recommendations. In *Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on*, pages 214–220. IEEE, 2003.

[40] SongJie Gong, HongWu Ye, and HengSong Tan. Combining memory-based and model-based collaborative filtering in recommender system. In *Circuits, Communications and Systems, 2009. PACCS'09. Pacific-Asia Conference on*, pages 690–693. IEEE, 2009.

[41] David M Pennock, Eric Horvitz, Steve Lawrence, and C Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 473–480. Morgan Kaufmann Publishers Inc., 2000.

[42] Joao Vinagre. Time-aware collaborative fitering: a review.

[43] Pedro G Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1-2):67–119, 2014.

[44] Pedro G Campos, Fernando Dıez, and Iván Cantador. A performance comparison of time-aware recommendation models.

[45] Tong Queue Lee, Young Park, and Yong-Tae Park. An empirical study on effectiveness of temporal information as implicit ratings. *Expert systems with Applications*, 36(2):1315–1321, 2009.

[46] Yi Ding and Xue Li. Time weight collaborative filtering. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 485–492. ACM, 2005.

[47] Yi Ding, Xue Li, and Maria E Orlowska. Recency-based collaborative filtering. In *Proceedings of the 17th Australasian Database Conference-Volume 49*, pages 99–107. Australian Computer Society, Inc., 2006.

[48] Tong Queue Lee, Young Park, and Yong-Tae Park. A time-based approach to effective recommender systems using implicit feedback. *Expert systems with applications*, 34(4):3055–3062, 2008.

[49] Dan Li, Peng Cao, Yucui Guo, and Min Lei. Time weight update model based on the memory principle in collaborative filtering. *Journal of Computers*, 8(11):2763–2767, 2013.

[50] Nan Zheng and Qiudan Li. A recommender system based on tag and time information for social tagging systems. *Expert Systems with Applications*, 38(4):4575–4587, 2011.

[51] Nathan N Liu, Min Zhao, Evan Xiang, and Qiang Yang. Online evolutionary collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 95–102. ACM, 2010.

[52] Neal Lathia, Stephen Hailes, and Licia Capra. knn cf: a temporal social network. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 227–234. ACM, 2008.

[53] Neal Lathia, Stephen Hailes, and Licia Capra. Temporal collaborative filtering with adaptive neighbourhoods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 796–797. ACM, 2009.

[54] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. Springer, 2011.

[55] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[56] Robert M Bell, Yehuda Koren, and Chris Volinsky. The bellkor 2008 solution to the netflix prize. *Statistics Research Department at AT&T Research*, 2008.

[57] Liang Xiang and Qing Yang. Time-dependent models in collaborative filtering based recommender system. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT'09. IEEE/WIC/ACM International Joint Conferences on*, volume 1, pages 450–457. IET, 2009.

[58] Alexandros Karatzoglou. Collaborative temporal order modeling. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 313–316. ACM, 2011.

[59] Sung-Hwan Min and Ingoo Han. Detection of the customer time-variant pattern for improving recommender systems. *Expert Systems with Applications*, 28(2):189–199, 2005.

[60] Linas Baltrunas and Xavier Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*, 2009.

[61] Huanhuan Cao, Enhong Chen, Jie Yang, and Hui Xiong. Enhancing recommender systems under volatile userinterest drifts. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1257–1266. ACM, 2009.

[62] Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 723–732. ACM, 2010.

[63] Gediminas Adomavicius and Alexander Tuzhilin. Extending recommender systems: A multidimensional approach. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-01), Workshop on Intelligent Techniques for Web Personalization (ITWP2001), Seattle, Washington, August*, pages 4–6. Citeseer, 2001.

[64] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems (TOIS)*, 23(1):103–145, 2005.

[65] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[66] Kostas Stefanidis, Irene Ntoutsi, Kjetil Nørvåg, and Hans-Peter Kriegel. A framework for time-aware recommendations. In *Database and Expert Systems Applications*, pages 329–344. Springer, 2012.

[67] Bernardo A Huberman, Daniel M Romero, and Fang Wu. Social networks that matter: Twitter under the microscope. *arXiv preprint arXiv:0812.1045*, 2008.

[68] Statistiche twitter, `http://expandedramblings.com/index.php/march-2013-by-the-numbers-a-few-amazing-twitter-stats/#.U6ROqfl_uSp`.

[69] Xujuan Zhou, Yue Xu, Yuefeng Li, Audun Josang, and Clive Cox. The state-of-the-art in personalized recommender systems for social networking. *Artificial Intelligence Review*, 37(2):119–132, 2012.

[70] Werner Geyer, Jill Freyne, Bamshad Mobasher, Sarabjot Singh Anand, and Casey Dugan. Recommender systems and the social web. In *2nd workshop on recommender systems and the social web, Proceedings of the fourth ACM conference on Recommender systems (RecSys, 10)*, pages 379–380, 2010.

[71] Su Mon Kywe, Ee-Peng Lim, and Feida Zhu. A survey of recommender systems in twitter. In *Social Informatics*, pages 420–433. Springer, 2012.

[72] John Hannon, Mike Bennett, and Barry Smyth. Recommending twitter users to follow using content and collaborative filtering approaches. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 199–206. ACM, 2010.

[73] The oauth 2.0 authorization framework, `http://tools.ietf.org/html/rfc6749`.

[74] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *Signal Processing Magazine, IEEE*, 25(2):128–131, 2008.

[75] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001.

[76] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[77] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[78] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. Setting goals and choosing metrics for recommender system evaluations. In *Proceedings of the Second Workshop on User-Centric Evaluation of Recommender Systems and Their Interfaces (UCERSTI 2)*, 2011.

[79] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011.

[80] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *The Journal of Machine Learning Research*, 10:2935–2962, 2009.

[81] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[82] Michael Hahsler. Developing and testing top-n recommendation algorithms for 0-1 data using recommenderlab, 2010.

[83] Neal Lathia, Stephen Hailes, and Licia Capra. Evaluating collaborative filtering over time. In *Proceedings of the SIGIR 2009 Workshop on the Future of IR Evaluation*, pages 41–42. Citeseer, 2009.

[84] Neal Lathia, Stephen Hailes, Licia Capra, and Xavier Amatriain. Temporal diversity in recommender systems. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 210–217. ACM, 2010.

[85] Kumar Arunachalam and Dr P Thambidurai. Collaborative web recommendation systems-a survey approach. *Global Journal of Computer Science and Technology*, 9(5), 2010.

[86] Alejandro Bellogín, Iván Cantador, Fernando Díez, Pablo Castells, and Enrique Chavarriaga. An empirical comparison of social, collaborative filtering, and hybrid recommenders. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(1):14, 2013.

[87] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[88] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying

collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[89] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. A comparative study of collaborative filtering algorithms. *arXiv preprint arXiv:1205.3193*, 2012.

[90] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[91] Linyuan Lü, Matúš Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. Recommender systems. *Physics Reports*, 519(1):1–49, 2012.

[92] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[93] Alan Said, Alejandro Bellogín, and Arjen de Vries. A top-n recommender system evaluation protocol inspired by deployed systems. 2013.

[94] Loren Terveen and Will Hill. Beyond recommender systems: Helping people help each other. *HCI in the New Millennium*, 1:487–509, 2001.

[95] Chong Lin Zheng, Kuang Rong Hao, and Yong Sheng Ding. A collaborative filtering recommendation algorithm incorporated with life cycle. *Advanced Materials Research*, 765:630–633, 2013.

[96] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

[97] Paul B Kantor, Lior Rokach, Francesco Ricci, and Bracha Shapira. *Recommender systems handbook*. Springer, 2011.

[98] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[99] Toby Segaran. *Programming collective intelligence: building smart web 2.0 applications*. O'Reilly Media, Inc., 2007.

[100] Linas Baltrunas and Francesco Ricci. Item weighting techniques for collaborative filtering. In *Knowledge Discovery Enhanced with Semantic and Social Information*, pages 109–126. Springer, 2009.

[101] Christian Desrosiers and George Karypis. A novel approach to compute similarities and its application to item recommendation. In *PRICAI 2010: Trends in Artificial Intelligence*, pages 39–51. Springer, 2010.

[102] Fabio Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 273–280. ACM, 2013.

[103] Shumeet Baluja, Michele Covell, and Sergey Ioffe. Permutation grouping: intelligent hash function design for audio & image retrieval. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 2137–2140. IEEE, 2008.

[104] Kliček Božidar, Oreški Dijana, and Begičević Nina. Temporal recommender systems. In *Proceedings of the 10th WSEAS international conference on Applied computer and applied computational science*, pages 248–253. World Scientific and Engineering Academy and Society (WSEAS), 2011.

[105] Pedro G Campos, Alejandro Bellogín, Fernando Díez, and J Enrique Chavarriaga. Simple time-biased knn-based recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, pages 20–23. ACM, 2010.

[106] Pedro G Campos, Fernando Díez, and Manuel Sánchez-Montañés. Towards a more realistic evaluation: testing the ability to predict future tastes of matrix factorization-based recommenders. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 309–312. ACM, 2011.

[107] Zhengdong Lu, Deepak Agarwal, and Inderjit S Dhillon. A spatio-temporal approach to collaborative filtering. In *Proceedings of the third ACM conference on Recommender systems*, pages 13–20. ACM, 2009.

[108] Andreas Töscher, Michael Jahrer, and Robert Legenstein. Improved neighborhood-based algorithms for large-scale recommender systems. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, page 4. ACM, 2008.

[109] Jing Wang and Jian Yin. Enhancing accuracy of user-based collaborative filtering recommendation algorithm in social network. In *System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2012 3rd International Conference on*, volume 1, pages 142–145. IEEE, 2012.