Dipartimento di Impresa e Management
Corso di Laurea Magistrale in Marketing
Cattedra: Customer Intelligence e Logiche di Analisi dei Big Data

RECOMMENDATION SYSTEM: IMPLEMENTAZIONE DI ALGORITMI COLLABORATIVI PER SERVIZI VIDEO-STREAMING ON DEMAND (VOD)

RELATORE
Prof. Luigi Laura

CANDIDATO
Fabrizio Vitale
Matr. 692341

CORRELATORE
Prof. Paolo Spagnoletti

ANNO ACCADEMICO 2018/2019

A mio padre, Alberto.
Faro perpetuo hai illuminato il mio cammino.
Costanza, Ambizione e Metodo sono frutto dei tuoi insegnamenti.

A mia madre, Carmela.
Dolce uragano hai spazzato via tutti i miei ostacoli.
Determinazione, Umiltà ed il Fuoco dei miei occhi appartengono a te.

A mia sorella, Antonella.
Compagna di giochi e litigi. Mio specchio riflesso
ma allo stesso tempo meravigliosamente diversa da me.
In te ripongo la mia fiducia incondizionata.

Ai miei nonni,
Pasquale ed Antonietta.
Il vostro affetto risiede indelebile nel mio cuore.
Certo che da lassù continuiate a proteggermi.
Vincenzo e Rita.
Mio porto sicuro, parte essenziale dell'anima mia.
Darei qualsiasi cosa per congelare il tempo ed il vostro amore con esso.

Alla mia città, Napoli.
Terra mia che ribolli nel mio sangue.

Sommario

Nell'era della digitalizzazione, grazie alla diffusione del web sono stati abbattuti molti dei limiti strutturali che fisiologicamente caratterizzano da sempre i negozi fisici. Dal momento che i bisogni dei consumatori sono divenuti sempre più complessi, risulta necessario offrire una grande varietà di prodotti e servizi ad una ampia pluralità di nicchie. La nostra economia dunque, si sta spostando gradualmente da quella che era una cultura dei prodotti *mainstream*, verso un'iper-segmentazione del mercato. Avendo adesso la possibilità di accesso ad una infinita varietà di prodotti, il consumatore è libero di soddisfare i suoi bisogni, se pur specifici, e sostanzialmente divergenti da esigenze più comuni alla maggioranza degli individui. Viene tuttavia a presentarsi la problematica opposta: nella ricerca della soluzione migliore, troppe possibilità di scelta possono indurre il consumatore a vivere la fase decisionale in modo oppressivo e spossante piuttosto che liberatorio. I *Recommendation System* hanno proprio l'obiettivo di andare a risolvere questa tipologia di difficoltà.

Essi sono dei sistemi automatici che, sfruttando tecniche di *data mining*, si occupano di assistere gli utenti nelle decisioni online, consigliando l'acquisto o la fruizione di quegli articoli che corrispondono ai loro interessi.

Il seguente elaborato fornisce la descrizione di approcci e metodologie per l'implementazione di sistemi di raccomandazione. Il focus si indirizza sull'applicazione di motori di raccomandazione nel campo dei servizi video-streaming on demand (VOD), in particolare sono stati implementati algoritmi collaborativi su un *dataset* idoneo a scopi di ricerca e reperibile online.

INDICE

Introduzione **1**

1 Rilevanza: definizione del problema **5**

1.1 E-commerce e superamento del principio di Pareto 5
1.2 Long Tail theory 6
1.3 Choice Overload 8

2 Recommendation System **10**

2.1 Elementi essenziali: user, item, transaction 10
2.2 Benefici per l'utente.....11
2.3 Benefici per le aziende.....12
2.4 Caratteristiche essenziali.....13
2.5 Settori di Applicazione e Best practice.....13
2.6 Fasi del processo di raccomandazione.....15
2.7 Formalizzazione matematica del problema 17
2.8 Classificazione sintetica.....18

3 Approcci ed algoritmi **21**

3.1 Collaborative Filtering: Memory-Based.....21
3.1.1 User-based Collaborative Filtering 21
3.1.2 Item-based Collaborative Filtering.....24
3.1.3 Limiti generali dei sistemi collaborativi.....26
3.1.4 User-based e Item-based a confronto 27
3.2 Collaborative Filtering: Model-Based 29
3.2.1 Latent factor model: Singular Value Decomposition.....30
3.2.2 Association rule recommender system.....31
3.2.3 Clustering Recommendation system.....34
3.3 Content Filtering.....39
3.3.1 Tecniche di analisi del contenuto: VSM e TD-IDF 41
3.3.2 Content filtering vs Collaborative filtering: confronto.....43
3.4 Hybrid Filtering 44
3.5 Metodi di Valutazione: MAE, RMSE, Precision, Recall, ROC 46
3.5.1 Metriche statistiche di accuratezza: MAE, RMSE.....47
3.5.2 Metriche di supporto decisionale: precision, recall, ROC.....48
3.5.3 Diversità, novità e coverage 51

4 Analisi: implementazione di sistemi di raccomandazione per servizi Video Streaming on Demand con R studio 52

4.1 Obiettivo dell'analisi e campo di applicazione 52
4.2 Origine e contenuto del dataset "Movielens" 53
4.3 Metodologia 57
4.3.1 Ispezione e pulizia/manipolazione dati 57
4.3.2 Analisi esplorativa 60
4.4 Implementazione dei sistemi di raccomandazione su Movielens 67
4.4.1 Preprocessing: data frame in realRatingMatrix 68
4.4.2 Impostazione evaluationSchema: Train e Test 70
4.4.3 Implementazione recommender systems 71
4.4.4 Ottenere le predizioni 72
4.4.5 Valutazione sistema di raccomandazione 72
4.5 Procedura per la selezione del modello ottimale 73
4.5.1 UBCF: ottimizzazione parametri 74
4.5.2 IBCF: ottimizzazione parametri 81
4.5.3 UBCF e IBCF a confronto 83
4.5.4 Recommendation list: estrazione e visualizzazione 86

Conclusioni 90

Comandi R 92

Bibliografia 112

Sitografia 115

ELENCO DELLE FIGURE

1.1 Ranking mondiale capitalizzazione aziende 2006 vs 2016 4
1.2 Crescita esponenziale e-commerce 6
1.3 Long Tail Theory 7

2.1 Fasi del processo di raccomandazione 15
2.2 Classificazione sistemi di raccomandazione 19
2.3 Collaborative filtering vs Content-Based filtering 20

3.1 User-Based filtering vs Item-Based filtering 28
3.2 Latent Factor Model 31
3.3 Singular Value Decomposition 31
3.4 Clustering 35
3.5 Clustering: metodo gerarchico 36
3.6 Clustering applicato al Recommendation system 39
3.7 Content-Based filtering 39
3.8 Processo di raccomandazione Content-based 40
3.9 Vector Space Model 43
3.10 Hybrid Recommendation system 46
3.11 Matrice di confusione 48
3.12 ROC Curve 50

4.1 Homepage Movielens.org 54
4.2 Caratteristiche Movielens.org 55
4.3 Data Frame rating 58
4.4 Data Frame movies 59
4.5 Data Frame tags 59
4.6 Polarità rating 60
4.7 Distribuzione rating 61
4.8 Top 10 titoli più popolari 62
4.9 Top 10 titoli più apprezzati 62
4.10 Evoluzione titoli negli anni 63
4.11 Top 10 generi più popolari 64
4.12 Evoluzione generi negli anni 65
4.13 Tag più frequenti genere Adventure 66
4.14 Tag più frequenti genere Drama 66
4.15 Tag più frequenti genere Crime 67

4.16 Matrice dei rating 69
4.17 Matrice dei rating zoomata 69
4.18 ROC Curve UBCF Neighbor 25 76
4.19 Misure di errore UBCF Neighbor 25 78
4.20 ROC Curve UBCF 80
4.21 Misure di errore UBCF 80
4.22 ROC Curve IBCF 82
4.23 Misure di errore IBCF 82
4.24 Velocità Recommendation system implementati 84
4.25 ROC Curve UBCF vs IBCF vs Random 85
4.26 Misure di errore UBCF vs IBCF vs Random 85
4.27 Recommendation List utente 1 87
4.28 Distribuzione film Recommendation List 88
4.29 Recommendation List evidenziata 89

Introduzione

L'informatizzazione della nostra società ha notevolmente migliorato le nostre capacità di generare e raccogliere informazioni da una quantità di fonti sempre crescenti. Un'enorme mole di dati ha colpito quasi ogni aspetto della nostra vita. Secondo International Data Corporation, il 90% dei dati esistenti al giorno d'oggi sono stati generati soltanto negli ultimi due anni, con un flusso talmente crescente che le informazioni accumulate hanno superato 1 triliardo di *byte*, arrivando a raggiungere un vero e proprio record. Inoltre, il ritmo con cui queste informazioni sono prodotte è talmente veloce che ogni due giorni viene creato un volume di dati pari alla quantità di informazioni generate dall'umanità intera fino al 2003. Ogni minuto vengono inviate più di 200 milioni di e-mail, 300 mila tweet, su YouTube vengono caricati più di 60 ore di contenuti, ed effettuate più di 9 milioni di telefonate. Inoltre, si stima che entro il 2020 verranno creati circa 35 triliardi di dati. Ci troviamo quindi in quella che molti definiscono l'era dei *Big Data*. Questo concetto inizia a diffondersi dal momento in cui ci viene data la possibilità di raccogliere una quantità talmente grande e complessa di dati che rende necessaria l'elaborazione di nuovi strumenti e metodologie per estrapolare, gestire e processare informazioni in un intervallo di tempo ragionevole [1]. Sebbene non esista una definizione univoca di *big data*, sono diversi gli autori che nel corso degli anni hanno cercato di descrivere un *set* di caratteristiche tali da poter associare ad una raccolta di dati la connotazione di *"big"*.

Doug Laney (2001) in un suo articolo è stato il primo ad attribuirne 3 dimensioni caratterizzanti: il volume, la varietà e la velocità.

- Il **volume** si riferisce alla quantità dei dati, indipendentemente sia dalla tipologia, strutturati o destrutturati, che dalla fonte che li origina; essi infatti possono essere generati dagli stessi utenti oppure da macchine in modo automatico. Sono tuttavia i dati non strutturati, principalmente legati al mondo di Internet come immagini, video, mail e contenuti appartenenti al mondo dei social network, a crescere in modo continuo ed esponenziale.
- La **varietà** si lega proprio a quest'ultimo concetto, ovvero l'eterogeneità dei formati con cui si presentano i dati, che il più delle volte non sono più

rappresentabili da tabelle strutturate nei tradizionali *database* relazionali. Ciò genera certamente difficoltà per la raccolta e l'analisi dei dati e di conseguenza richiede la necessità di sviluppare e adottare nuove infrastrutture tecnologiche e nuove figure professionali emergenti per gestirla [2].

- La **velocità** invece si riferisce al fatto che i dati vengono prodotti ad un ritmo e una frequenza sempre crescente. Per questo motivo le organizzazioni hanno il dovere di processarli con la massima rapidità fino ad arrivare ad una interazione *real-time* che consente di prendere decisioni in tempi brevissimi, cogliere in modo istantaneo eventuali opportunità che il mercato è in grado di offrire e fronteggiare possibili crisi grazie ad un monitoraggio continuo e costante delle informazioni.

Alle suddette caratteristiche sono state poi aggiunti altri due attributi qualificanti: il valore e la variabilità (Kaisler et al., 2012).

- Per **valore** si intende la misura in cui i dati riescono ad apportare utilità al processo decisionale delle organizzazioni che li utilizzano. Lo scopo della raccolta e l'analisi dei dati infatti si concretizza nella scoperta di *insight* che hanno la possibilità di orientare decisioni e produrre quindi valore.
- La **variabilità** invece si associa alla differente interpretazione e di conseguenza significato che uno stesso dato può assumere a seconda dell'*environment* in cui viene processato. Il valore quindi, non è riconducibile direttamente al dato stesso, piuttosto al contesto da cui si ricava.

Per far fronte all'esigenza di sfruttare il gigante patrimonio informativo che potenzialmente è contenuto nelle ingenti raccolte di dati sono state sviluppate e implementate diverse discipline il cui scopo è quello di trasformare il dato in conoscenza e successivamente in decisioni. Tra le tante, il *Data Mining* risulta essere quella di riferimento. Il termine *"mining"* ha origini legate alle attività di estrazioni minerarie, in particolare alla rimozione di grandi quantità di materiale all'interno delle miniere per la ricerca di sostanze nobili. Analogamente, ribaltando il concetto nel mondo dei dati, il *data mining* è considerato come l'insieme di tecniche e metodologie che hanno per oggetto l'estrazione di un sapere o di una conoscenza a partire da grandi volumi di dati e l'utilizzo scientifico, industriale o operativo di questo sapere [3].

Scavando quindi all'interno del gigante universo di dati oggi a nostra disposizione, lo scopo del *data mining* consiste nell'individuazione dei cosiddetti *"pattern"*, ovvero

schemi ricorrenti, modelli o regolarità all'interno di un grande insieme di dati osservati. Ne consegue direttamente che partendo da essi è possibile costruire le basi per formulare e verificare nuove ipotesi su relazioni causali di determinati fenomeni e fare previsioni statisticamente significative su nuovi insiemi di dati non ancora esaminati [4]. Tra le differenti tecniche è bene fare una distinzione tra i metodi automatici e semiautomatici. Ai primi è strettamente correlato il concetto di *"machine learning"*: il termine significa appunto apprendimento automatico. Esso è considerato una branca dell'intelligenza artificiale che riguarda la costruzione e lo studio di sistemi che possono imparare dai dati stessi. L'obiettivo del *machine learning* è sviluppare metodi che sono in grado automaticamente di scoprire pattern nascosti e modellizzarli al fine di effettuare predizioni future basate su eventi passati. In sostanza quindi, il principio cardine risiede nel rendere una macchina in grado di generalizzare la propria esperienza (Bishop, 2006) fornendogli l'abilità di svolgere compiti nuovi, mai incontrati, dopo aver fatto esperienze su un insieme di dati di apprendimento, dandogli in poche parole la possibilità di svolgere ragionamenti induttivi.

Proprio a causa della natura stessa di queste discipline e delle evidenti implicazioni in termini di business, al giorno d'oggi esse hanno completamente invaso il mondo delle organizzazioni. La potenzialità infinita che il corretto utilizzo di queste discipline può offrire ruota intorno alla capacità di prevedere cosa desideriamo, i nostri comportamenti, le nostre azioni e le modalità con cui possiamo raggiungere un determinato obiettivo. I dati e la gestione intelligente di essi sono paragonati ormai al petrolio del futuro, il nuovo oro. Risulta evidente, da un'analisi fatta da Statista [5] sulle aziende che hanno la capitalizzazione maggiore a livello mondiale, che negli anni la situazione è cambiata in modo radicale (Figura 1.1). È emblematico che, mentre nel 2006 l'azienda con il valore maggiore, ExxonMobil, e il 50% della *Top six* appartenevano al settore dell'energia, in particolare del petrolio, nel 2016 cinque aziende su sei si occupano di *tech* e fanno del loro punto di forza proprio la raccolta e la gestione di una quantità immensa di dati.

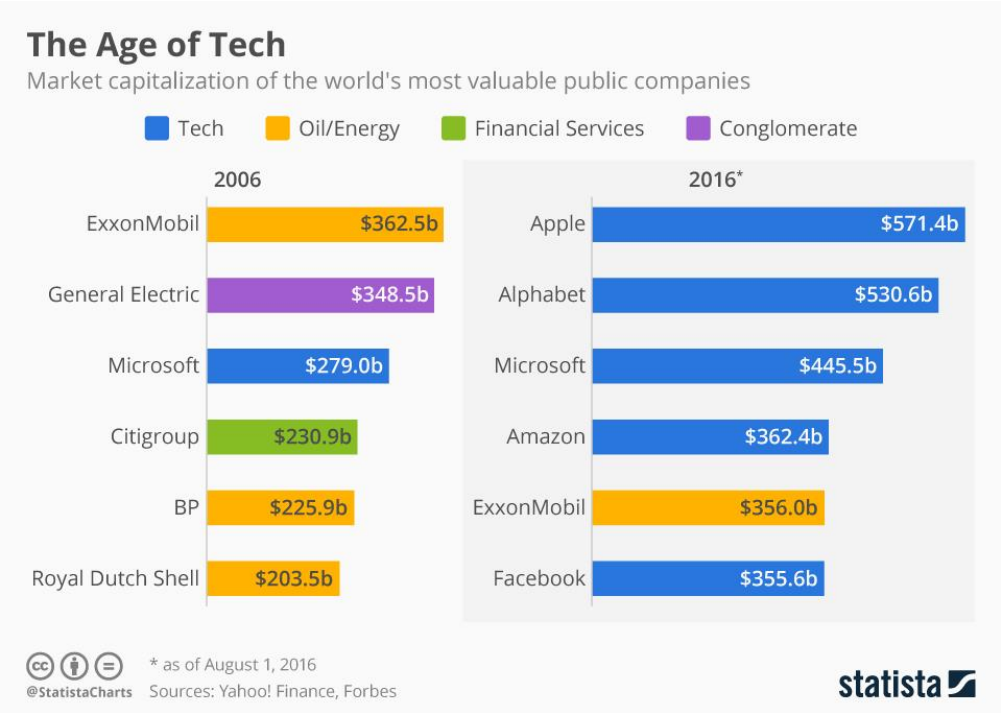

Figura 1.1. Ranking mondiale capitalizzazione aziende 2006 vs 2016 (Statista, 2017)

CAPITOLO 1

Rilevanza: definizione del problema

1.1 E-commerce e superamento del Principio di Pareto

Nell'era della digitalizzazione, grazie alla diffusione dell'e-commerce sono stati abbattuti molti dei limiti strutturali che fisiologicamente caratterizzano da sempre i negozi fisici. Questi ultimi hanno il compito di massimizzare il loro profitto cercando di sfruttare al massimo lo spazio a loro disposizione, che risulta quindi una risorsa scarsa da ottimizzare. Essi tendono di conseguenza ad inserire nel loro assortimento quel numero limitato di prodotti fronteggiati da una domanda consistente, andando così a trascurare gli articoli che possiamo definire di nicchia. Per tanto tempo, la maggioranza dei consumatori ha indirizzato le proprie scelte sulle cosiddette "*hits*", non perché avessero tutti preferenze comuni piuttosto per la mancanza fisica di un'offerta alternativa da parte dei *retailer*. Questa tendenza andava a confermare uno dei principi empirici più applicati del secolo scorso: il principio di Pareto. Secondo l'economista, la maggior parte degli effetti, indicativamente l'80% è dovuto ad un numero ristretto di cause, circa il 20%. Applicando il suddetto principio al nostro discorso, ne risulta che il 20% dei prodotti sarebbe responsabile dell'80% del fatturato mondiale.

Dal momento in cui i bisogni dei consumatori sono diventati sempre più complessi, e con la crescita esponenziale del web, lo scenario precedentemente descritto ha avuto una modifica sostanziale. L'e-commerce ha infatti assunto un ruolo rilevante in tutto il mondo, tale da raggiungere nel 2018 un valore di 1 miliardo e 614 milioni e si stima che con un tasso di crescita medio (CAGR) del 8,9% possa arrivare a fatturare 2 miliardi e 507 milioni nel 2023 (Figura 1.2). A testimoniare la diffusione dell'e-commerce è l'enorme numero di user, 3 miliardi e mezzo, e ci si aspetta che tocchi i 4 miliardi e mezzo nel 2023, raggiungendo una penetrazione di mercato del 58,5%. Di grande impatto anche la spesa media del consumatore online che nel 2018 è stata di circa 460 euro e le stime rivelano che l'ammontare incrementerà fino a 570 euro nel 2023 [6]. Sono evidenti i vantaggi che l'online ha offerto in termini di opportunità alle aziende: esse hanno la possibilità di superare i limiti spaziali e temporali che colpiscono i *retailer* tradizionali, riuscendo allo stesso tempo a ridurre i costi di

gestione. Per il consumatore invece, oltre a vantaggi in termini di ampiezza dell'offerta, che determina una maggiore libertà di esprimere e soddisfare i bisogni più specifici e particolari, generalmente si aggiunge anche una maggiore convenienza in termini di prezzo. Assume rilevanza così quello che Anderson definisce il superamento del Principio paretiano della scarsità dei fattori, a vantaggio della *Long Tail theory*. Quest'ultima partendo proprio dalla Curva paretiana ne fornisce una visione totalmente differente.

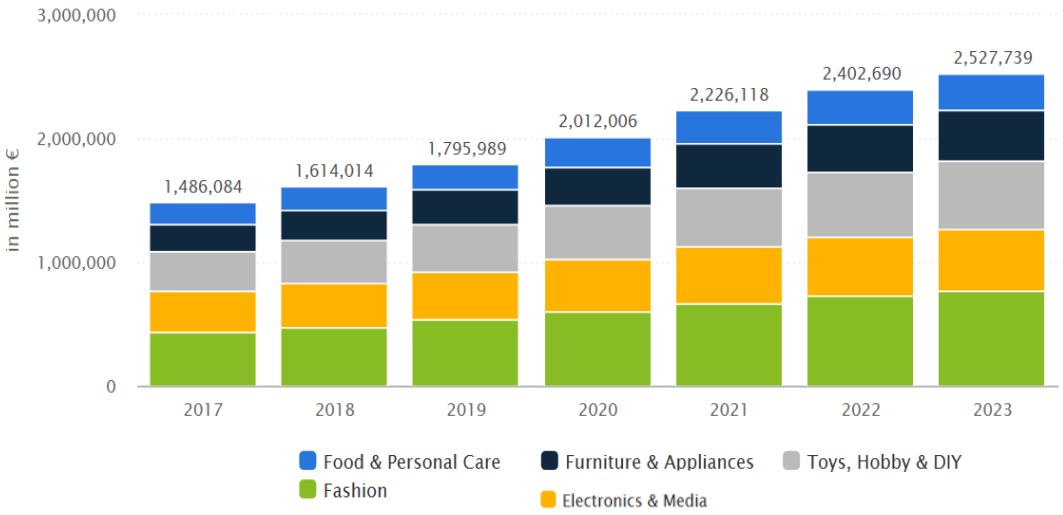

Figura 1.2. Crescita esponenziale e-commerce (Statista, 2019)

1.2 Long Tail Theory

Nel suo libro *"The Log Tail"* Anderson, afferma che grazie allo sviluppo dei mercati digitali la nostra economia si sta spostando gradualmente da quella che definiamo una cultura dei prodotti *mainstream*, rappresentati nella testa della curva (Figura 1.3) verso un grande numero di nicchie di mercato, posizionate nella coda.

Alla base della logica *"long tail"*, c'è il seguente ragionamento: offrire una grande varietà di prodotti e servizi a una ampia pluratità di nicchie di mercato può essere altrettanto profittevole rispetto alla tradizionale offerta di pochi prodotti di successo per un unico vasto pubblico. Questa teoria descrive un modello commerciale che è ormai ampiamente diffuso in tutto il mondo e non a caso è utilizzato da alcune delle aziende più innovative e leader del loro mercato di riferimento. A testimonianza di ciò, basti pensare che Amazon lista più di 3 miliardi di prodotti in 11 *marketplace* differenti in tutto il mondo, ed arriva a raggiungere più di 300 milioni di consumatori.

Allo stesso modo Netflix, all'interno della sua offerta, mette a disposizione 1569 TV Shows e 4010 film riuscendo a sottoscrivere circa 139 milioni di abbonamenti [7]. Nel settore dello streaming musicale invece, esemplare è Spotify, che guida il suo mercato con 40 milioni di canzoni per un totale di circa 96 milioni di utenti in tutto il mondo nel 2019 [8].

Questa logica, empiricamente testata da Anderson (2006), si è diffusa ed ha avuto successo grazie alla presenza di alcuni fattori abilitanti:

1. Assenza Limiti fisici e temporali: le aziende superano i limiti orari e geografici di un negozio fisico. Non essendo più legati ad un luogo preciso e senza orari di apertura e chiusura, i propri clienti potenziali aumentano in modo vertiginoso in quanto diventano tutti coloro che hanno la possibilità di accedere ad internet.
2. Gusti dei consumatori non univoci
3. Riduzione costi di stoccaggio: quando si tratta di un prodotto "digitale" i costi di stoccaggio tendono ad essere irrisori; basta confrontare il costo di avere dei dischi musicali in magazzino con quello di avere milioni di canzoni in uno spazio virtuale.
4. Democratizzazione strumenti produttivi: tutti gli individui, e non più solo i professionisti, hanno la facoltà di produrre contenuti come musica, cortometraggi, blog e tante altre tipologie di prodotti/servizi. È avvenuto un passaggio dal consumismo al "produttivismo partecipativo" [9].
5. Prezzi competitivi: grazie ai suddetti punti, sorgono una quantità esponenziale di players che con l'aiuto della trasparenza di informazioni fornita dal web, tendono a ridurre i prezzi a causa della competizione più accesa.

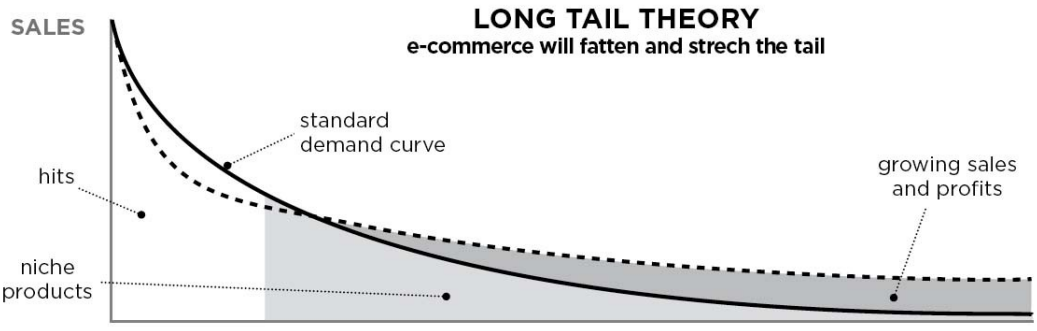

Figura 1.3. Long Tail Theory (Elbers 2008)

1.3 Choice overload

Avendo adesso la possibilità di accesso ad una infinita varietà di prodotti e servizi possiamo definirci completamente liberi di acquistare qualsiasi prodotto che soddisfi un nostro bisogno, se pur specifico, e sostanzialmente divergente da esigenze più comuni a tutti gli individui. Potrebbe però presentarsi quella che è la problematica opposta alla situazione precedente: troppe possibilità di scelta talvolta implicano nessuna decisione.

"Si può verificare quello che è chiamato *choice overload,* ovvero quel fenomeno secondo il quale date numerose possibilità di scelta, tutte appetibili, si viene a creare un blocco il cui risultato è una stasi in cui si è incapaci di valutare quale sia la soluzione migliore. Durante la fase decisionale infatti, sono coinvolte due regioni cerebrali: la corteccia anteriore cingolata, che pesa costi e benefici delle scelte, e lo striato, una parte del cervello che dà giudizi di valore. L'interazione tra queste due aree indica una valutazione in atto: il cervello soppesa la potenziale ricompensa con la fatica necessaria a vagliare tutte le opzioni. Man a mano che le scelte a disposizione aumentano, crescono sia le potenziali ricompense, sia il lavoro di cernita da fare. Occorre quindi trovare un equilibrio tra lo sforzo mentale, che non deve essere eccessivo, e la ricompensa, che si spera importante" [10].

Lato consumatore quindi, risulta evidente che uno scenario simile potrebbe portare non solo a non migliorare quella che era una situazione in cui si accontentava di avere un numero limitato di scelte, ma addirittura a peggiorarla. Schwartz in *"The paradox of choice"* (2004) cita uno studio in cui vengono descritte le sensazioni di alcuni consumatori all'interno di un supermercato durante la scelta di quale marmellata acquistare tra la moltitudine di quelle disponibili. Dopo un certo numero di opzioni, al crescere di queste ultime, essi tendevano a comprare di meno e allo stesso tempo si sentivano meno soddisfatti del loro acquisto. I consumatori arrivavano a provare una sensazione di sovraccarico che li portava a definire la propria scelta oppressiva e spossante piuttosto che liberatoria (Iyengar e Lepper, 2000). Tuttavia, nonostante siano stati accertati gli svantaggi che vengono associati ad un ventaglio di scelte eccessivo, gli individui ne sentono comunque il bisogno, non tanto per la quantità delle opzioni, piuttosto per rendersi parte attiva del processo decisionale. Gli stessi autori però concludono che sebbene sia importante che la varietà di scelta sia ampia, altrettanto importante è l'assistenza che le aziende devono fornire al consumatore per cercare ciò

che più corrisponde ai loro interessi, fronteggiando così la sensazione di oppressione e facilitando il suo processo decisionale.

Dal lato delle aziende invece il problema si declina in modo differente. Fermo restando, i benefici che esse traggono dalla gestione di un negozio digitale già descritto in precedenza, obiettivo dei venditori sarà quello di riuscire ad indirizzare l'enorme varietà di prodotti al consumatore giusto, renderli quindi facilmente visibili e reperibili, cercando di ridurre al minimo il costo cognitivo che il consumatore impiegherebbe in una potenziale ricerca. Avviene così un vero e proprio passaggio dall'era della ricerca a quella della scoperta, in cui c'è un reale capovolgimento: non è più il consumatore ad essere alla ricerca di un prodotto ma è il prodotto a cercare il consumatore. L'obiettivo ideale sarebbe quello di indirizzare sforzi pubblicitari e offerte di prodotti/servizi ritagliati su misura rispetto al *target* di riferimento andando ad instaurare idealmente una relazione *one-to-one* con i clienti che si troveranno a vivere una esperienza di interazione e acquisto completamente personalizzata: *"If I have 3 million customers on the* web*, I should have 3 million stores on the web". (Bezos J.)*

CAPITOLO 2

Recommendation Systems

Ottenere consigli e raccomandazioni da fonti di fiducia è una componente critica del naturale processo decisionale dell'essere umano. Esigenza che è diventata sempre più impellente nell'era dell'*informational overload* in cui il ventaglio di scelte in certi casi risulta eccessivamente ampio, fino a diventare demotivante.

In genere utilizziamo una varietà di strategie per selezionare cosa comprare, quanto spendere e come trascorrere il nostro tempo libero: ci affidano a ricerche sul web, a consigli di amici, al proprio istinto o addirittura chiediamo consulenze ad esperti del settore. Spesso però, queste raccomandazioni oltre a rivelarsi molto dispendiose sia in termini di tempo che in termini economici, il più delle volte sono anche confusionarie e nel peggiore dei casi non corrispondono ai nostri interessi reali.

I recommendation systems hanno l'obiettivo esatto di andare a risolvere questa tipologia di problematiche. Essi sono dei sistemi automatici che, sfruttando tecniche di *data mining*, si occupano di assistere gli utenti nelle decisioni online, consigliandogli l'acquisto o la fruizione degli item che corrispondono il più possibile ai loro interessi. Prima di entrare nel dettaglio sul funzionamento dei sistemi di raccomandazione e di fornire una appropriata classificazione sulle diverse tipologie utilizzate, è necessario descrivere gli elementi essenziali su cui si basa, illustarne i benefici che interessano aziende e utenti e spiegare in quali settori trovano maggiore applicazione.

2.1 Elementi essenziali

Qualsiasi tipologia di sistema di raccomandazione è costituito da 3 elementi essenziali: gli *item*, gli *user* e le *transaction.* (Ricci et al, 2011)

Gli *user* sono i protagonisti del recommandation system, i soggetti ai quali è indirizzato il consiglio di uno o più articoli, ma allo stesso tempo coloro che grazie alle loro caratteristiche (età, sesso, preferenze etc.) e comportamenti alimentano il *dataset* alla base della raccomandazione e quindi di tutto il sistema.

Gli *item* sono l'oggetto della raccomandazione da parte del sistema agli utenti. Essi sono definiti da un proprio *set* di attributi o caratteristiche che variano a seconda della

tipologia dell'articolo (per un libro potrebbero essere l'autore, il genere l'anno di pubblicazione etc.). Ad essi l'utente assocerà un'utilità soggettiva che dipende dai propri gusti e bisogni. Affinché il sistema sia in grado di comprendere gli interessi degli utenti, entrano in gioco le *transaction:* con esse possiamo intendere qualsiasi interazione *user-computer* che sia utile al fine di prevedere le preferenze del consumatore rispetto agli item. Queste ultime possono essere esplicite come l'invio volontario di un rating oppure implicite, ad esempio derivante dal *tracking* del comportamento online riguardo ricerche web o acquisti; in quest'ultimo caso gli interessi dell'utente sono dedotti dalle sue azioni

2.2 Benefici per l'utente

Lato consumatore, Herlocker et al. (2004) ci fornisce una classificazione dei principali vantaggi offerti dal sistema di raccomandazione:

- **Trovare item interessanti**: Il compito chiave dei sistemi di raccomandazione è certamente quello di raccomandare tra tutti i contenuti disponibili quelli che corrispondono agli interessi dell'utente. Ciò apporta vantaggi in termini di rilevanza degli articoli rispetto ai propri gusti, in termini di tempo per la ricerca dei contenuti più appropriati.
- **Migliorare il proprio profilo*:*** gli utenti contribuiscono a rilasciare i loro feedback con la finalità di delineare il loro profilo con maggiore precisione. In questo modo la loro interfaccia risulta personalizzata e di conseguenza anche la propria esperienza di navigazione. Al crescere delle valutazioni e quindi dei dati a disposizione aumenta anche la qualità delle raccomandazioni.
- **Esprimere sé stessi:** alcuni utenti, più che interessati all'accuratezza della raccomandazione, trovano giovamento nel fornire le proprie valutazioni semplicemente per il gusto di esprimere la propria opinione. Tale atteggiamento è stato riscontrato spesso sul sito Amazon.com dove gli utenti valutano i prodotti acquistati; questa tipologia di utenti è stata definita *"power user"*.
- **Aiutare gli altri:** in certi casi invece, gli utenti partecipano alle valutazioni con il solo intento di aiutare tutta la *community* di user a trovare i contenuti generalmente più interessanti. Essi provano soddisfazione nel fatto che altri utenti possano beneficiare del proprio contributo.

- **Intrattenimento:** anche se generalmente i sistemi di raccomandazione sono valutati sulla base di quanto essi aiutino gli utenti a prendere decisioni di consumo corrette, talvolta l'intento di questi ultimi è semplicemente quello di navigare sui siti senza un preciso bisogno imminente di acquisto ma al contrario con la sola volontà di intrattenimento. In questi casi piuttosto che l'accuratezza delle raccomandazioni risulta più importante l'interfaccia proposta, la facilità d'uso e la natura delle informazioni fornite.

2.3 Benefici per le aziende

F. Ricci et al., (2011) hanno descritto i principali vantaggi che le aziende digitali traggono dall'utilizzo dei sistemi di raccomandazione. Essi si traducono essenzialmente in un aumento complessivo delle vendite e la costruzione di una relazione solida con il consumatore. Di seguito la descrizione dei principali benefici:

- **Incremento del numero di articoli venduti:** dal momento che gli item sono consigliati al consumatore grazie ad un sistema di raccomandazione è più probabile che gli articoli proposti intercettino i bisogni e gli interessi del consumatore e che quindi vengano acquistati/fruiti.
- **Vendere articoli diversi:** anche prodotti/servizi di nicchia, particolarmente sconosciuti, grazie alla raccomandazione possono avere una visibilità maggiore e di conseguenza essere comprati.
- **Incrementare la soddisfazione dell'utente:** l'accuratezza delle raccomandazioni, combinata ad una interfaccia *easy to use* va a influenzare positivamente l'esperienza dell'utente all'interno del sito o applicazione.
- **Aumentare la loyalty:** quanto più un utente interagisce con il sito/app tanto più il suo profilo sarà ben definito e di conseguenza le raccomandazioni saranno più accurate, fino a *matchare* sempre di più le preferenze del cliente. Ciò alimenta un meccanismo a feedback positivo che spinge l'utente a non cambiare piattaforma/azienda ed essere *loyal*
- **Comprendere meglio i bisogni:** una volta raccolte le informazioni e i comportamenti del consumatore, il *provider* potrebbe decidere di utilizzare questa conoscenza per ottimizzare la gestione dello stock, la produzione, i messaggi promozionali etc.

2.4 Caratteristiche Recommendation Systems

Affinché un sistema di raccomandazione sia efficace per gli utenti è necessario che abbia alcune caratteristiche precise (Aggarwal, 2016):

- **Rilevanza**: condizione necessaria ma non sufficiente è che gli item raccomandati siano rilevanti, cioè siano contenuti che corrispondano agli interessi dell'utente.
- **Novità**: è importante che il sistema raccomandi item nuovi ed originali piuttosto che contenuti prevedibili e ripetitivi che l'utente ha già avuto modo di incontrare.
- ***Serendipity***: il termine indica la situazione in cui un individuo si imbatte in scoperte sorprendenti, fatte per puro caso oppure trovi una cosa non cercata e quindi imprevista (H.Walpole). In questo caso, il valore aggiunto della raccomandazione si esplicita quando l'item consigliato all'utente non solo è nuovo ma è anche inaspettato; potrebbe capitare che venga consigliato un contenuto particolarmente differente rispetto al genere di item che l'utente generalmente incontra ma per cui tuttavia prova un interesse latente.
- **Diversità**: Nel caso in cui il sistema di raccomandazione suggerisca una lista di elementi, è bene che non siamo troppo simili tra essi, in questo modo aumenta la probabilità che almeno un elemento tra quelli suggeriti sia coerente con le preferenze dell'utente (Ziegler et al., 2005).

2.5 Settori di applicazione e Best Practice

I sistemi di raccomandazione trovano applicazione in numerosi settori. È importante distinguere i differenti campi di applicazione poiché a seconda dell'*industry* vengono utilizzate metodologie di recommendation system differenti. Montaner, et al,.(2003) forniscono una tassonomia dei dominii di applicazioni più diffusi:

- Intrattenimento: film, musica e IPTV, social network
- Content: giornali, documenti, pagine web, applicazioni *e-learning*.
- E-commerce: Prodotti da acquistare
- Servizi: viaggi, mercato immobiliare, *matchmaking service*.

Saranno illustrati di seguito le best practice delle aziende che hanno fatto dei sistemi di raccomandazione una risorsa chiave del proprio business. Amazon utilizza il sistema

di raccomandazione come strumento di *targeted marketing* all'interno del proprio sito web. I clienti hanno la possibilità di accedere in modo completamente intuitivo a prodotti raccomandati in base ai loro interessi, filtrati per tipologia di prodotto, rating, articoli già acquistati o visti, cliccando su "Consigliati per te". Gli utenti inoltre possono anche vedere il motivo per cui un prodotto gli viene consigliato.

Linden, Smith e York (2003) spiegano che Amazon utilizza algoritmi di raccomandazione per rendere il proprio *store* personalizzato per ciascun consumatore. Il sito infatti cambia radicalmente la propria interfaccia in base agli interessi del consumatore e al proprio comportamento di navigazione. Secondo McKinsey & Company, il 35% dei ricavi di Amazon è generato dall'accuratezza del suo sistema di raccomandazione [11]. Per quanti riguarda invece il settore dei servizi di video-streaming si stima che i sistemi di raccomandazione apportino alle aziende un *saving* di circa 1 miliardo di dollari (Gomez e Hunt, 2016). Ciò consente alle imprese di investire una grande quantità di denaro sulla creazione di nuovi contenuti da fornire ai clienti, impattando positivamente sul ROI. Netflix in particolare lo utilizza per assistere gli utenti a cercare facilmente una serie tv o un film, stimando la probabilità che essi possano guardare un determinato titolo del catalogo sulla base di alcune variabili input: le interazioni con il servizio (cronologia di visione, rating di altri articoli), comportamenti di altri utenti con preferenze simili, informazioni sui titoli (genere, attori, regista, anno di uscita, ecc.). A queste ultime vengono aggregati anche altri fattori come il momento della giornata in cui utilizzano il servizio, su quali dispositivi vengono riprodotti e per quanto tempo. Netflix elabora tutte queste informazioni in algoritmi che vanno a sviluppare il sistema di raccomandazione proprietario che garantisce da anni un'esperienza personalizzata agli utenti. Il risultato eccellente che ad oggi garantisce l'algoritmo di Netflix è frutto della crescente base dati a disposizione e di numerosi miglioramenti che negli anni sono stati implementati; tra le fasi più importanti che ha attraversato Netflix, è importante ricordare che nel 2006 creò una competizione, il *"Netflix Prize contest"* in cui premiò con 1 milione di dollari un team che riuscì a migliorare in termini di accuratezza (RMSE) "*Cinetech*", l'algoritmo già in uso dall'azienda. A testimonianza del ruolo cruciale che svolge il Recommendation system, uno studio di McKinsey [11] certifica che il 75% di quello che gli utenti guardano su Netflix proviene da raccomandazioni da parte del sistema. Anche nel settore dello *streaming* musicale i sistemi di raccomandazione hanno

assunto un ruolo importante; obiettivo ideale in questo caso è riuscire a fornire a tutti clienti playlist personalizzate. La differenza essenziale rispetto al video-streaming sta nella grandezza del catalogo, circa 300 volte superiore, e nei feedback che sono principalmente impliciti poiché più che dare una valutazione ai brani ascoltati viene rilevato il comportamento degli utenti durante la fruizione del servizio. Emblematico è il successo di Spotify: ogni lunedì, oltre 100 milioni di utenti ricevono la *"Discovery Weekly",* una playlist raccomandatagli sulla base delle loro interazioni con il servizio. Essa contiene circa 30 canzoni che l'utente non ha mai ascoltato prima ma che con alta probabilità potrebbero piacergli. Spotify ha sviluppato un algoritmo che combina 3 diverse strategie di raccomandazione: esamina il comportamento individuale dell'utente e lo compara con le azioni di tutti gli altri, analizza i testi di tutte le canzoni, osserva e compara le caratteristiche sonore delle canzoni per comprenderne il ritmo e la tonalità. Il suo sistema di raccomandazione ha consentito a Spotify di incrementare nel 2016 il numero di utenti da 75 a 100 milioni nel 2015, nonostante la stringente competizione con il principale competitors Apple Music. (Leonard, 2016)

2.6 Fasi del processo di raccomandazione

Qualsiasi tipologia di Recommandation system ha bisogno di seguire 3 specifiche fasi (Figura 2.1) affinché sia in grado di fornire le proprie raccomandazioni (F.O. Isinkaye et al., 2015):

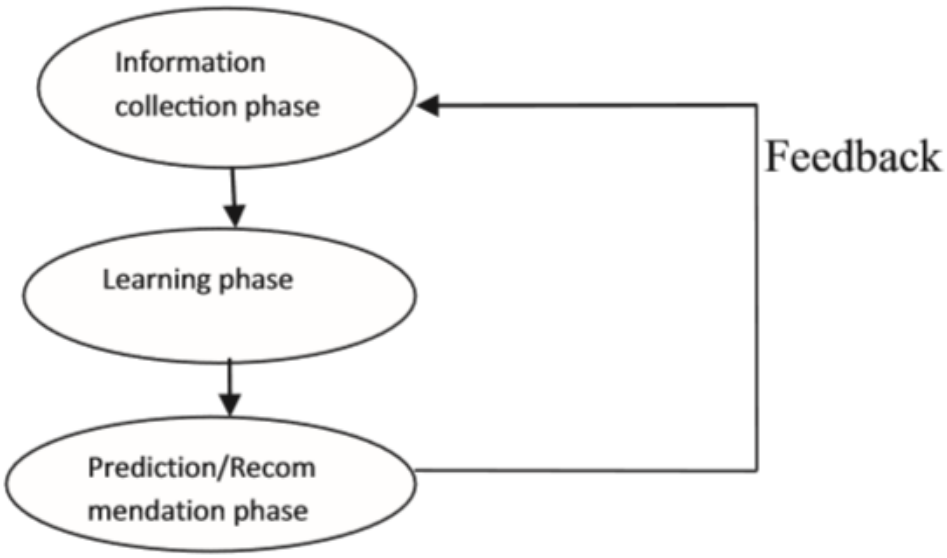

Figura 2.1. Fasi del processo di raccomandazione (F.O. Isinkaye et al., 2015)

Information collecting phase

La prima fase riguarda la raccolta delle informazioni rilevanti dell'utente con lo scopo di generare un profilo che sarà alla base della predizione finale. Un sistema di raccomandazione non può funzionare accuratamente finché non è stato costruito un profilo dell'utente ben strutturato. Esso può essere ricavato da diverse fonti: *explicit feedback*, che sono considerate le più efficaci e includono input espliciti da parte degli utenti riguardo i loro interessi rispetto gli item. *Implicit feedback*, attraverso il quale le preferenze sono dedotte indirettamente attraverso i comportamenti degli utenti. *Hybrid feedback* possono essere ottenuti attraverso la combinazione di *explicit* ed *implicit* feedback. Il profilo dell'utente è dunque utilizzato per raccogliere le informazioni necessarie alla costruzione di un modello. La qualità di qualsiasi recommendation system dipende in gran parte dalla sua abilità a rappresentare gli interessi correnti dell'utente. Modelli accurati sono indispensabili per ottenere accurate raccomandazioni per qualsiasi tecnica di predizione utilizzata.

Explicit feedback: in presenza di essi, il sistema normalmente spinge l'utente attraverso un'interfaccia per fornire rating rispetto a determinati item al fine di costruire e migliorare il suo modello. L'accuratezza della raccomandazione dipende dalla quantità di rating forniti dall'utente. L'unico svantaggio di questo metodo è che richiede uno sforzo per l'utente, il quale non sempre è disposto a partecipare. Nonostante ciò, il feedback esplicito fornisce dati molto affidabili e trasparenti per il processo di raccomandazione che risulta senza dubbio più efficace.

Implicit feedback: in questo caso invece, il sistema automaticamente deduce le preferenze dell'utente monitorando le sue azioni, come lo storico degli acquisti, la cronologia di navigazione, il tempo speso sulle pagine web etc. Questo metodo riduce in modo sostanziale lo sforzo dell'utente ma è meno accurato. Tuttavia, è stato affermato che potrebbe essere in un certo senso più oggettivo in quanto l'utente non incorre in alcun *bias* di rispondere in modo socialmente desiderabile.

Hybrid Feedback: esso combina i vantaggi del feedback implicito ed esplicito e cerca di minimizzarne le debolezze in modo da ottenere performance migliori. I dati impliciti possono essere utilizzati come *check* dei rating espliciti oppure chiedendo agli utenti di fornire feedback espliciti solo quanto scelgono volontariamente e non forzatamente di esprimerli.

Learning phase

In questa fase sono applicati algoritmi di apprendimento per filtrare e sfruttare le caratteristiche dell'utente dai feedback raccolti durante la fase di raccolta delle informazioni. A seconda del dominio di applicazione possono essere sfruttate tecniche differenti.

Prediction/recommendation phase

Fase finale in cui il sistema predice quale tipologia di item l'utente potrebbe preferire. L'uso di appropriate tecniche di raccomandazione a seconda della situazione è decisivo per garantire una performance qualitativa del sistema.

Queste ultime saranno descritte successivamente nel dettaglio, definendone le diverse logiche su cui si basano e commentandone i vantaggi e svantaggi di ciascuna.

2.7 Formalizzazione matematica del problema

Il compito alla base di un generico sistema di raccomandazione è prevedere se un utente potrebbe provare o meno interesse verso un item che non ha mai avuto l'occasione di incontrare, basandosi sul suo profilo e/o sulle caratteristiche dell'oggetto che verrà consigliato. Quanto più un item rispecchi le preferenze del consumatore tanto più sarà l'utilità associata ad esso; quest'ultima è generalmente rappresentata da un rating che verrà predetto dal sistema.

Il cuore del suddetto problema si riassume quindi nella stima di una funzione di utilità. (G. Adomavicius, A. Tuzhilin, 2005).

Formalizzandolo dal punto di vista matematico avremmo:

Dati:

C: {Utenti}

S: {item raccomandabili}

u: funzione di utilità che misura l'utilità di un item s per un utente c,

u: C X S→ R

R: {item raccomandati}.

Per ogni utente *c*, vogliamo scegliere l'item *s* che massimizzi *u*

$$\boldsymbol{c \in C \qquad s'_c = argmax_u\, u(c, s)}$$

Possiamo descrivere *C* come lo *user-space* in cui può essere definito un profilo che include varie caratteristiche dell'utente come l'età, il genere, reddito, etc. Nel caso più semplice, il profilo può contenere solo un singolo elemento, cioè un ID distintivo. Allo stesso modo, ogni elemento dello spazio *S*, ovvero quello degli item raccomandabili, è definito con un set di caratteristiche. Ad esempio, nella raccomandazione di film, dove *S* è l'insieme di tutti i film, ognuno può essere rappresentato non solo dal suo ID, ma anche dal titolo, genere, regista, anno di produzione, attori protagonisti, etc. Il problema centrale sta nel fatto che l'utilità non è mai definita in tutto lo spazio *C* x *S*, ma solo per una sottocategoria di esso. Di fatto essa è tipicamente rappresentata da un rating ed è inizialmente definita solo sugli item già conosciuti e votati dall'utente.
Il motore di raccomandazione dovrebbe essere capace di stimare (predire) i rating delle combinazioni item/user non presenti e fornire le raccomandazioni basate su queste predizioni. Le raccomandazioni effettive di un articolo a un utente vengono effettuate selezionando la valutazione più alta tra tutte le valutazioni stimate per quell'utente. In alternativa, è possibile consigliare gli *N* elementi migliori a un utente o ad un gruppo di utenti. I rating degli articoli non ancora valutati possono essere predetti utilizzando tecniche di raccomandazione diverse che si basano su logiche differenti.
Di seguito una classificazione sintetica delle varie metodologie che verranno poi approfondite singolarmente nel capitolo successivo.

2.8 Classificazione sintetica

I sistemi di raccomandazioni possono sfruttare diversi approcci attraverso cui è possibile consigliare i contenuti agli utenti. Isinkaye, Folajini e Ojokoh (2015) forniscono quella che è considerata la tradizionale tassonomia delle metodologie più diffuse e principalmente utilizzate (Figura 2.2)
La prima classificazione presenta tre macrocategorie che si differenziano per la logica che è alla base del sistema di raccomandazione:

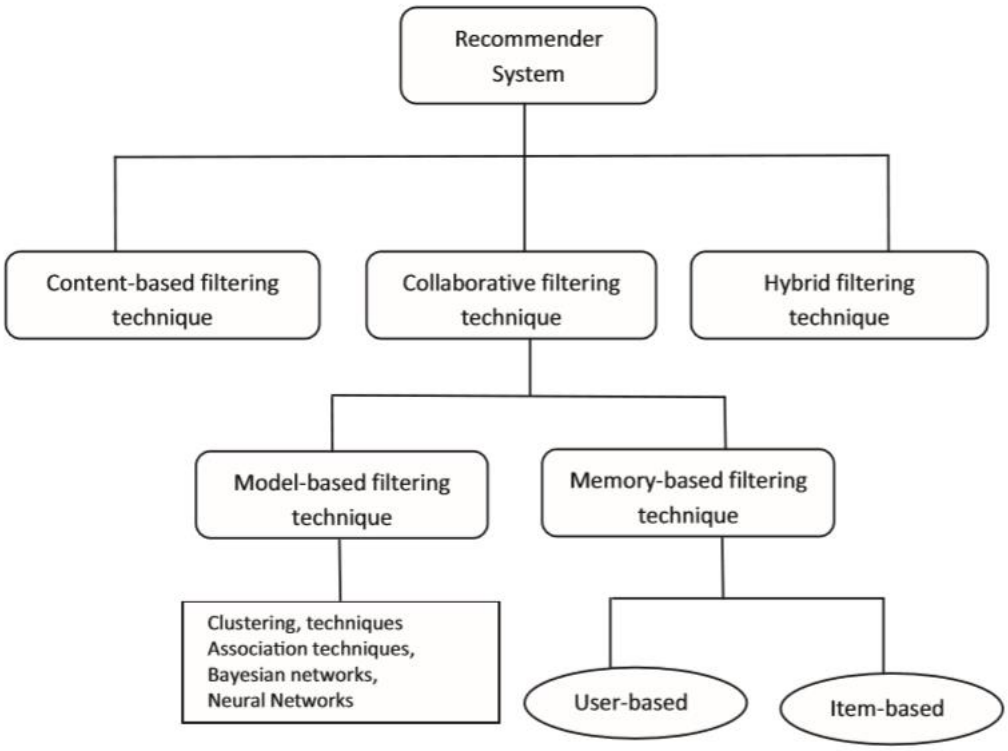

Figura 2.2. Classificazione sistemi di raccomandazione (Isinkaye et al., 2015)

- ***Collaborative-based filtering***: approccio più utilizzato, talvolta trattato come sinonimo di Recommendation System, è basato sulla raccolta di una grande quantità di informazioni su comportamenti, attività e gusti degli utenti. Esso è in grado di predire cosa piacerà agli utenti sulla base della somiglianza delle loro proferenze con quelle degli altri utenti.

- ***Content-based filtering*:** approccio che si fonda sulla descrizione delle caratteristiche degli oggetti da raccomandare e sulle preferenze degli utenti. Esso prova a raccomandare agli utenti gli articoli che hanno caratteristiche somiglianti a quelli che essi hanno già apprezzato in passato.

- ***Hybrid filtering***: metodologia che combina i due sistemi sopra citati.

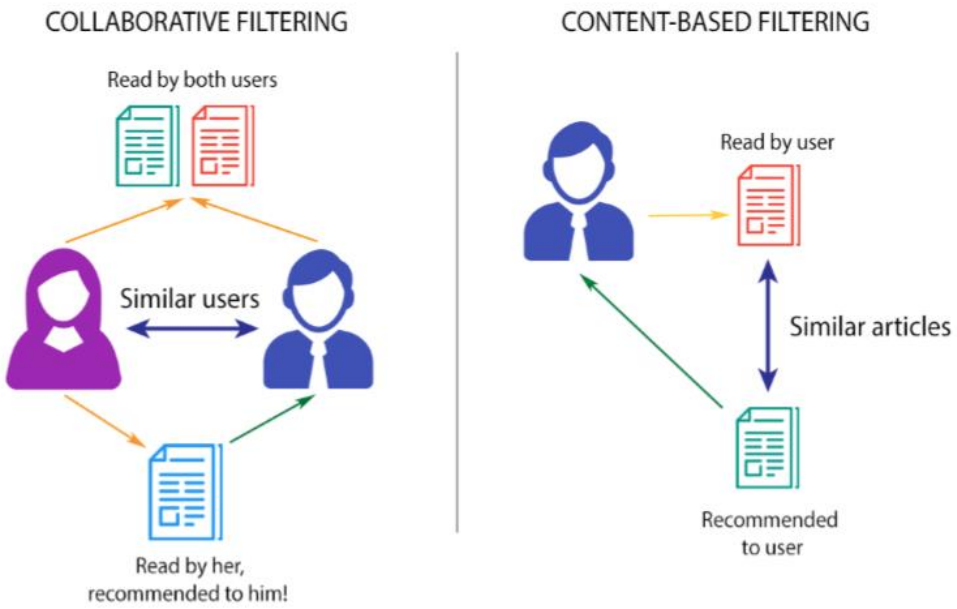

Figura 2.3. Collaborative filtering vs Content-Based filtering (Chaturvedi, 2017)

Una seconda tipologia di classificazione riguarda il metodo di raccomandazione collaborativo: esso può essere suddiviso in una ulteriore sottocategoria differenziata a seconda del modo in cui è preso in considerazione il *dataset* e sulla tipologia di algoritmi utilizzati per analizzarlo:

- ***Memory-based***: Di questa categoria fanno parte gli algoritmi che, per predire valutazioni future su articoli non ancora incontrati, si basano sullo storico di tutti i rating (in memoria) degli utenti. Per questo scopo vengono utilizzati due approcci differenti: *User-based filtering* in cui vengono calcolate le similitudini tra le preferenze degli utenti e *Item-based filtering* che invece considera la somiglianza tra i rating dei contenuti.

- ***Model-based***: al contrario, le tecniche di raccomandazione *Model-based* non usano direttamente la storia del sistema per fare previsioni, ma piuttosto se ne avvalgono per apprendere un modello, che poi è utilizzato per generare raccomandazioni. In questa categoria rientrano sistemi che fanno uso di reti *bayesiane*, reti neurali, tecniche di *clustering*, regole di associazione.

CAPITOLO 3

Approcci ed algoritmi

Dopo aver presentato una classificazione sintetica delle diverse tipologie di sistemi di raccomandazione, in questo capitolo, ogni categoria sarà analizzata nel dettaglio, descrivendone gli algoritmi utilizzati e fornendo vantaggi e svantaggi di ciascuna.

3.1 Collaborative Filtering: Memory-Based

L'idea principale dell'approccio collaborativo è sfruttare le informazioni sui comportamenti o opinioni passate di una esistente *community* di utenti per predire quale item è più probabile che interessi agli utenti correnti.
Il puro approccio collaborativo ha come unico input una matrice *user-item* di rating che produce generalmente i seguenti tipi di output:
1. Una predizione numerica che indica il livello di interesse che gli utenti correnti potrebbero esprimere se incontrassero gli item.
2. Una lista di *n* item raccomandati.
È possibile utilizzare due logiche differenti: *user-based* e *item-based*

3.1.1 User-based collaborative filtering

Nell'approccio *user-based*, dato come input iniziale un database con i rating e gli ID degli user correnti, vengono identificati altri utenti che hanno preferenze similari a quelli attivi in passato. Quindi, per ogni prodotto *p* che l'utente non ha ancora visto, viene stimata una predizione basata sui rating per *p* fatti dagli utenti passati.
Questa metodologia si basa su due assunzioni:
1) Se gli utenti hanno avuto gusti simili in passato essi avranno gusti simili anche in futuro.
2) Le preferenze degli utenti restano stabili e coerenti lungo il tempo.
Di seguito viene fornito un esempio (Tabella 3.1) che ha l'obiettivo di spiegare il funzionamento alla base del sistema di raccomandazione collaborativo *user-based* (Riadattato da: Dietmar et al., 2010).

	Item 1	Item 2	Item 3	Item 4	Item 5
User X	5	3	4	4	?
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

Tabella 3.1. Matrice rating user-item

La tabella di cui sopra mostra un database di rating effettuati da 5 utenti. Consideriamo come nostro utente obiettivo lo User X, ovvero quello su cui intendiamo effettuare la nostra raccomandazione. User X ha valutato l'Item 1 con un 5 su una scala da 1 a 5, ciò significa che prova un forte interesse per quell'item. L'obiettivo del sistema di raccomandazione qui è determinare se allo User X piacerà o meno l'Item 5, che egli non ha ancora incontrato. Per fare ciò abbiamo bisogno di effettuare 2 procedure:

1. Identificare gli utenti le cui preferenze sono simili a quelle dello User X
2. Utilizzare i rating di questi ultimi, chiamati *"neighbor"*, effettuati per l'item 5, con lo scopo di predire se User X proverà o meno interesse per esso.

Prima di dare una interpretazione matematica al suddetto problema, introduciamo le convenzioni:

$U = \{u_1,\ldots,u_n\}$per identificare l'insieme degli utenti

$P = \{p_1,\ldots,p_m\}$ for identificare l'insieme degli item

R come una matrice di rating $r_{i,j}$ $n \times m$ con $i \in 1\ldots n, j \in 1\ldots m$

I possibili valori dei rating sono definiti su una scala numerica che va da 1 (non mi piace per niente) a 5 (mi piace tantissimo).

Se un certo utente i non ha valutato un item j, la corrispondente cella della matrice $r_{i,j}$ resterà vuota. Per determinare l'insieme di utenti simili, è possibile utilizzare diverse tecniche che misurano la "distanza" e quindi la somiglianza tra gli user. La tecnica più utilizzata è il coefficiente di correlazione di Pearson che Herlocker et al. (1999) affermano essere la più efficace nell'approccio *user-based*.

La similarità *sim(userX,user1)* dell'utente *X* e dell'utente *1*, data la matrice dei rating *R*, è definita dalla formula seguente:

$$sim(userX, user1) = \frac{\sum_{p \in P}(r_{x,p} - \bar{r_x})(r_{1,p} - \bar{r_1})}{\sqrt{\sum_{p \in P}(r_{x,p} - \bar{r_x})^2}\sqrt{\sum_{p \in P}(r_{1,p} - \bar{r_1})^2}}$$

Il simbolo $\overline{r_x}$ corrisponde alla media dei rating dell'utente *X*.

Utilizzando i valori della tabella, risulta che la similarità tra User X e lo User 1 è 0,85.

Il coefficiente di correlazione di Pearson assume valori da 1 (massima correlazione positiva) a -1 (massima correlazione negativa).

È necessario quindi effettuare il precedente calcolo tra l'utente X e tutti gli altri.

Di seguito tutti i risultati delle similarità:

sim(userX,user1)=0,85; *sim(userX,user2)*=0,70; *sim(userX,user3)*=0; *sim(userX,user4)* = - 0,79

Osserviamo che l'utente 1 e l'utente 2 risultano molto simili allo User X nei loro comportamenti di rating passati. Il coefficiente di Pearson inoltre tiene in considerazione il fatto che utenti diversi potrebbero avere un modo diverso di interpretare la scala dei rating, ad esempio qualcuno tende a non usare gli estremi della scala di misura.

Per fare una predizione per l'item 5, bisogna ora decidere quali dei *"neighbor user"* dovremmo tenere in considerazione e che peso attribuire ai rating di ognuno nella predizione. In questo caso risulta intuitivo tenere in considerazione user 1 e user 2 per predire il rating dello User X in quanto presentano una correlazione positiva rispettivamente di 0,85 e 0,79 e quindi un comportamento di rating simile.

La formula matematica che consente di predire la valutazione di un utente *a* per un item *p* che tiene conto anche della prossimità dei *neighbor N* più vicini è la seguente:

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b)\left(r_{b,p} - \overline{r_b}\right)}{\sum_{b \in N} sim(a,b)}$$

In quest'esempio abbiamo scelto che la predizione del rating di User X per l'item 5 è basata sui rating dei *neighbor* user 1 e 2; effettuando i calcoli utilizzando i dati della tabella, risulta che *pred(User X, Item5)* = 4,87.

Evidentemente il sistema di raccomandazione avrebbe raccomandato all'Utente X l'item 5. Nell'esempio precedente, abbiamo deciso di non prendere in considerazione tutti gli utenti per stimare la valutazione dell'utente X. Abbiamo infatti inserito solo quelli che avevano una correlazione più che positiva con l'utente corrente. Se includessimo nel calcolo tutti, questo influirebbe negativamente sulla performance, sia in termini di velocità di calcolo ma anche sull'accuratezza della raccomandazione, dal

momento che sarebbero prese in considerazione le valutazioni di altri utenti aventi interessi e comportamenti differenti.

Una tecnica famosa per ridurre la grandezza del *neighborhood* è definire una soglia minima di similarità tra gli utenti oppure fissare un numero di utenti massimo da tenere in considerazione. Anand e Mobasher (2005) e Herlocker et al. (1999) hanno discusso i potenziali problemi di queste tecniche: se la soglia è troppo alta, il numero di *neighbor* sarò troppo piccolo e ciò porterebbe ad una riduzione di copertura, che comporta una bassa capacità di consigliare un numero consistente di item. Se invece la soglia è troppo bassa, la grandezza del *neighborhood* non sarebbe significativamente ridotta e si aggiungerebbe *"noise"* alle predizioni.

3.1.2 Item-based collaborative filtering

L'approccio *item-based*, ugualmente al metodo precedentemente descritto, possiede come input iniziale una matrice *user-rating* e condivide anche la stessa tipologia di output, ovvero una predizione di quali item non ancora incontrati dovrebbero essere raccomandati agli utenti. La differenza sostanziale rispetto all'approccio *user-based* risiede nel fatto che la predizione non si basa sulla somiglianza tra i comportamenti degli utenti bensì sulla similarità fra i diversi item, sempre sulla base dei rating effettuati per essi. Basandoci sulla tabella utilizzata nell'esempio precedente, se avessimo esattamente lo stesso obiettivo, cioè predire se l'Item 5 potrebbe interessare o meno all'utente X, dovremmo seguire le seguenti procedure:

1) Considerare ogni item come un vettore di rating
2) Comparare ciascun *item-vector*, misurando il grado di somiglianza fra gli item e selezionare quelli tra loro più simili "*neighbor*"
3) Predire i rating che l'utente obiettivo potrebbe associare agli item non ancora incontrati, utilizzando i rating effettuati per gli item somiglianti "*neighbor*"

In questo esempio, il punto di partenza è identificare quali sono gli item che presentano una elevata somiglianza, in termini di rating degli utenti, con l'Item 5. Per fare ciò bisogna definire una misura di similarità. Per questo tipo di approccio la metrica più utilizzata che produce risultati più accurati non è l'indice di correlazione di Pearson ma la *cosine similarity*. Essa misura la similarità tra due vettori *n*-dimensionali basandosi sulla misurazione del coseno tra di loro. Questa misura è comunemente

usata nel *information retrivial* e *text mining* per comparare due documenti di testo in cui i documenti sono rappresentati come vettori di parole.

La similarità tra due item *a* e *b,* trasformati rispettivamente in vettori di rating $\vec{a}$ e $\vec{b}$, si calcola formalmente con la seguente formula:

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|}$$

Il simbolo · rappresenta il prodotto scalare dei vettori. $|\vec{a}|$ è la distanza Euclidea del vettore, che è definita come la radice quadrata del prodotto scalare del vettore con sé stesso. Prendendo ad esempio i valori della tabella, possiamo calcolare la *cosine similarity* tra l'Item 5 e l'Item 1 in questo modo:

$$sim(I5, I1) = \frac{3 * 3 + 5 * 4 + 4 * 3 + 1 * 1}{\sqrt{3^2 + 5^2 + 4^2 + 1^2} * \sqrt{3^2 + 4^2 + 3^2 + 1^2}} = 0{,}99$$

Il range di valori possibili della *cosine similarity* va da 0 a 1, dove i valori che si avvicinano ad 1 indicano una forte somiglianza. Uno dei difetti di questa metrica sta nel fatto che non considera il comportamento di rating medio degli utenti. Questo problema però, può essere risolto utilizzando l'*adjusted cosine similarity* che sottrae la media di tutti i rating dell'utente dai rating. In questo caso, il range di valori della misura si trova tra -1 e 1 proprio come nel caso dell'indice di Pearson.

Una volta calcolate tutte le somiglianze, bisogna selezionare i rating dell'utente obiettivo, User X, effettuati sugli item somiglianti a quello che abbiamo intenzione di raccomandare o meno, in questo caso l'item 5, associando però a ciascuno un peso a seconda del grado di somiglianza.

Formalmente, possiamo predire il rating di un utente *u* per un item *p* con la seguente formula (Dietmar et al., 2010):

$$pred(u, p) = \frac{\sum_{i \in ratedItems(u)} sim(i, p) * r_{u,i}}{\sum_{i \in ratedItems(a)} sim(i, p)}$$

3.1.3 Limiti generali dei sistemi di raccomandazione collaborativi

Nelle applicazioni reali tuttavia, i database sono molto più grandi fino a comprendere milioni di utenti e rating, ciò potrebbe apportare una grande complessità di calcolo. In più nella matrice dei rating generalmente ogni utente avrà valutato solo una piccola sottocategoria di item tra quelli disponibili. Sebbene questo approccio sia considerato quello più accurato e tendenzialmente in uso, fino a diventare spesso un vero e proprio sinonimo di Recommendation system, è importante analizzare quelli che sono le problematiche e i limiti che lo riguardano:

***Cold-Start problem*:** l'ingresso di nuovi item e nuovi utenti lancia una sfida significativa ai sistemi di raccomandazione. Il primo a formalizzare questo problema è stato Shein et al., (2002). Il problema sorge dal momento che un item non può essere raccomandato se qualche utente non l'ha mai valutato in precedenza. Ciò interessa non solo i contenuti che vengono aggiunti come novità, ma anche per quelli che possiamo definire di nicchia, raramente valutati, per cui gli utenti con gusti particolarmente unici verrebbero penalizzati. Questo limite può essere però superato con l'aiuto di un altro tipo di approccio che verrà approfondito nei paragrafi successivi, quello *Content-Filtering*, che basandosi solo sulle caratteristiche dell'item e non sui comportamenti degli utenti ha la facoltà di produrre raccomandazioni per tutti gli item.

Ancor più difficile da mitigare è la problematica dovuta all'ingresso di nuovi utenti, dal momento che senza aver individuato uno storico delle preferenze (rating) di un utente non è possibile trovare utenti che presentano comportamenti simili né costruire un profilo *Content-based*.

***Sparsity*:** dal momento che la situazione tipica di un *provider* di servizi o prodotti prevede un grandissimo set di prodotti ma un numero limitato di valutazioni da parte degli utenti, i sistemi di raccomandazione collaborativi si trovano a fronteggiare una complicazione ricorrente, ovvero la sparsità della matrice *user-rating*. Ciò ha un effetto particolarmente negativo per il sistema di raccomandazione collaborativo poiché tende a decrescere la probabilità di trovare un insieme di utenti con gusti simili. Questa problematica accade quando il rapporto *item-user* è molto alto, situazione frequente nella fase iniziale di avvio del sistema di raccomandazione. Tra le diverse soluzioni proposte, la più accreditata consiste in metodi di *dimensionality reduction* (Karatzoglou, 2013*)*, appartenenti alla categoria degli approcci *Model-based* che verranno approfonditi in seguito.

3.1.4 User-based e Item-based a confronto

Pur essendo entrambi sistemi *"collaborative"* e quindi basati sul calcolo delle somiglianze, la differenza principale tra l'approccio basato sugli utenti e quello fondato sugli item risiede negli elementi che sono alla base della computazione delle somiglianze e quindi del sistema di raccomandazione. L'approccio *item-based* parte dal calcolo delle similarità tra item e le utilizza per raccomandare i contenuti agli utenti. L'approccio *user-based* invece, parte dagli utenti e calcolando la somiglianza tra essi, fornisce raccomandazioni basate su cosa altri utenti con *pattern* di comportamento simili potrebbero apprezzare. (Figura 3.1)

Ne risulta che, pur avendo lo stesso obiettivo finale, il procedimento che porta alla stima della raccomandazione è differente. Ciò implica che i due approcci generano una serie di vantaggi e svantaggi differenti, dunque tra i due non esiste uno generalmente preferito all'altro, piuttosto vengono impiegati a seconda della situazione. Per questo è importante elencarne le ripercussioni sostanziali ai fini della raccomandazione (Ricci et al., 2011).

Accuratezza: l'accuratezza dei metodi di raccomandazione *neighborhood* dipende in gran misura dal rapporto tra il numero di utenti e item all'interno del sistema. Nel caso in cui il numero di utenti è molto più grande del numero di item, come ad esempio capita in Amazon, l'approccio *item-based* produce generalmente raccomandazioni più accurate. Al contrario, sistemi che hanno un numero di utenti minore rispetto agli item, ad esempio siti che consigliano *paper* scientifici in cui ci sono milioni di utenti per centinaia di articoli, l'approccio *user-based* è tendenzialmente migliore.

Efficienza: l'efficienza computazionale del sistema di raccomandazione e la memoria necessaria al *processing* dipende anch'essa dal rapporto utenti/item. Quando il numero di utenti supera il numero di item, il metodo *item-based* richiede meno memoria e tempo di computazione, rendendolo di conseguenza più scalabile.

Stabilità: la scelta tra i due approcci dipende anche dalla frequenza di aggiornamenti e quindi cambiamenti negli utenti e nei contenuti all'interno del sistema. Se l'elenco degli item disponibili tende ad essere più statico rispetto alla lista degli utenti correnti, il metodo *item-based* è preferibile dal momento che è possibile effettuare raccomandazioni ai nuovi utenti senza la necessità di calcolare nuovamente le similarità tra i contenuti. Viceversa, nelle applicazioni in cui la lista degli item è costantemente in aggiornamento, l'approccio *user-based* risulta più stabile.

Giustificazione: un vantaggio dei metodi basati sugli item è certamente attribuibile alla semplice giustificazione della raccomandazione agli utenti. Infatti, può essere presentata all'utente come motivazione una lista degli item *neighbor* utilizzati nella predizione e il loro grado di similarità con il contenuto raccomandato. Inoltre, dando la possibilità all'utente di modificare la lista, lo si può rendere partecipe al processo in modo interattivo. La giustificazione non può essere fornita con i metodi *user-based*, in quanto gli utenti nuovi generalmente non conoscono gli user simili utilizzati per la predizione e quindi psicologicamente potrebbero non comprendere le motivazioni della raccomandazione.

Serendipity: Nell'approccio *item-based*, il rating stimato che un utente potrebbe fornire ad un item è basato sulle valutazioni effettuate per item simili. Di conseguenza, il sistema di raccomandazione che utilizza questo metodo tende a consigliare agli utenti contenuti simili a quelli già apprezzati in precedenza. Da un lato, ciò tende a generare raccomandazioni più sicure, dall'altro non concede agli utenti la possibilità di scoprire tipologie di contenuti nuove, per cui potrebbe provare un interesse forte ma latente. Al contrario, l'approccio basato sugli utenti, dal momento che origina dal calcolo della similarità tra gli utenti, è più probabile che generi raccomandazioni originali e sorprendenti.

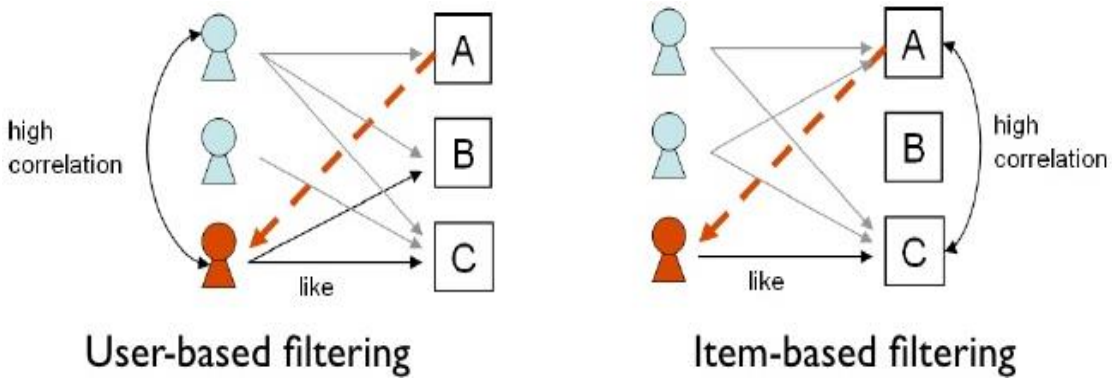

Figura 3.1. User-Based filtering vs Item-Based filtering [11]

3.2 Collaborative Filtering: Model-Based

Lo svantaggio principale delle tecniche *memory-based* descritte nei paragrafi precedenti risiede nella necessità di utilizzare un ammontare consistente di memoria. Questa problematica inizia ad essere più invadente dal momento che un gran numero di utenti utilizzano il sistema e quindi la matrice dei rating tende a diventare sempre più fitta. Ciò apporta un consumo di risorse computazionali ingente che incide negativamente sia sulla velocità delle raccomandazioni che sull'accuratezza. Le tecniche *model-based* hanno proprio l'obiettivo di risolvere queste tipologie di problemi (Thi Do et al., 2010). Esse forniscono una raccomandazione partendo dallo sviluppo di modelli *user-rating*; gli algoritmi appartenenti a questa categoria sfruttano un approccio probabilistico e concepiscono il processo di raccomandazione collaborativo come il calcolo di un valore atteso di una predizione dell'utente, dati i suoi rating su altri item. La principale differenza rispetto agli algoritmi *memory-based* sta nel fatto che in questo caso, partendo dal database iniziale, si crea un modello che è in grado di apprendere pattern nascosti tra i dati e li utilizza per effettuare predizioni. Per effettuare nuove raccomandazioni quindi, si sfrutta il modello senza la necessità di impiegare il *dataset* completo ogni volta. La costruzione del modello è ottenuta tramite l'utilizzo di algoritmi di *machine learning* supervisionato o non supervisionato. Di seguito saranno elencati i principali vantaggi che questa metodologia garantisce [12]:

- **Scalabilità**: generalmente i modelli risultanti dagli algoritmi *model-based* sono molto più piccoli dei *dataset* utilizzati per generarli e nonostante ciò risultano abbastanza efficienti. Ciò impatta in modo consistente sulla scalabilità complessiva del sistema.
- **Velocità della predizione**: conseguenza diretta della grandezza inferiore, gli algoritmi *model-based* tendono ad essere più veloci nella predizione poiché il tempo richiesto per interrogare il modello è nettamente inferiore di quello impiegato per l'intero *dataset*.
- **Evita *overfitting***: se il *dataset* su cui è costruito il modello è abbastanza rappresentativo del mondo reale non si va in contro ad una situazione di *overfitting*, in quanto il modello alla base è in grado di adattarsi non solo al *dataset* su cui è stato plasmato bensì anche a dati nuovi.

Nonostante ciò, queste tecniche presentano anche svantaggi che è importante riuscire a limitare per preservare l'accuratezza delle raccomandazioni:

- **Qualità della predizione:** Il fatto di non utilizzare l'intero *dataset* e quindi tutte le informazioni a disposizione, potrebbe comportare che le predizioni effettuate non siano completamente accurate. Tuttavia, ciò dipende strettamente dal modo in cui il modello è stato costruito e dalla tipologia di algoritmo utilizzato.
- **Poca flessibilità:** Lo sviluppo di un modello è una pratica che consuma tempo e risorse. Infatti, non è sempre agevole aggiungere nuovi dati e informazioni ad un sistema *model-based*, il che lo rende poco flessibile.

Nei paragrafi successivi saranno approfondite alcune delle tecniche di *machine learning* più utilizzate appartenenti alla categoria dei sistemi di raccomandazione *model-based*.

3.2.1 Latent factor model: Singular Value Decomposition

Alcune delle problematiche tipiche dei sistemi di collaborazione collaborativi, che spesso ne inficiano la performance, dipendono dalle caratteristiche generali della matrice *user-rating*. Tra i limiti più ricorrenti abbiamo la scalabilità, in quanto al crescere del numero di utenti e item, l'utilizzo di memoria risulta sempre più dispendioso e la sparsità della matrice, poiché gli utenti valutano solo una percentuale molto ristretta di contenuti rispetto al totale di quelli disponibili.

Il *latent factor model* è proprio una tecnica di *dimensionality reduction*, afferente alle metodologie di *machine learning* non supervisionato, utilizzata nel contesto dei sistemi di raccomandazione per mitigare i limiti sopracitati. Essa è un metodo in grado di estrapolare un set di fattori latenti dai comportamenti di rating, riuscendo così a caratterizzare sia gli utenti che gli item grazie ad una nuova matrice, ridotta, composta da vettori che identificano i nuovi fattori. Per *latent factor* si intende un elemento che descriva una determinata proprietà o caratteristica di un item o di un utente, ad esempio per i film potrebbe essere il genere. Avviene quindi una mappatura di ogni utente e item all'interno di uno spazio latente di dimensione *r* (Figura 3.2); sarà quindi molto semplice comprendere le relazioni fra essi e poterli comparare. Un'alta corrispondenza tra *item factor* e *user factor* porta quindi alla raccomandazione.

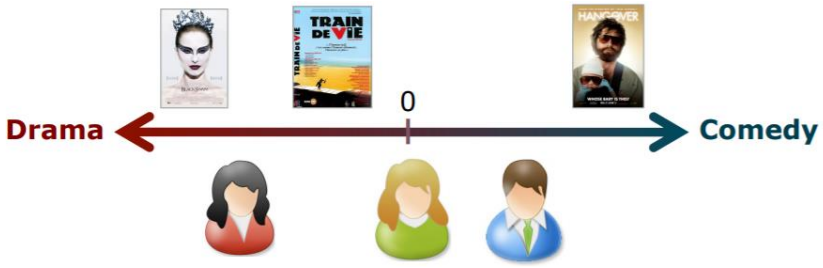

Figura 3.2. Latent Factor Model [14]

Uno degli algoritmi più utilizzati per scoprire i fattori latenti è il *Singular Value Decomposition* (Deerwester et al., 1990), inizialmente associato all'*informational retrivial*, ma poi applicato in tantissimi campi, tra cui quello dei sistemi di raccomandazione. In particolare, Sarwar et al., (2000) fornisce una spiegazione dettagliata del funzionamento di quest'algoritmo applicato al recommendation system. In sostanza, quello della raccomandazione diviene un problema di ottimizzazione: l'obiettivo del *Singular Value Decomposition* è quello di estrarre una matrice $\hat{X}$ che esprima l'utilità degli utenti *m* verso gli item *n* non ancora incontrati. (Figura 3.3)
È possibile ottenerla grazie al prodotto di tre matrici U, S, V^T:

$$\hat{X} = U * S * V^T$$

1. ***U*** rappresenta la relazione tra *m* utenti e *r* fattori latenti.
2. ***S*** è una matrice diagonale e descrive la forza di ogni fattore latente *r*.
3. ***V*** trasposta indica la similarità tra *n* item e *r* fattori latenti.

$$\underset{m \times n}{\hat{X}} \approx \underset{m \times r}{U} \; \underset{r \times r}{S} \; \underset{r \times n}{V^{\mathsf{T}}}$$

Figura 3.3. Singular Value Decomposition [15]

3.2.2 Association rule recommender system

L'*Association rule mining* è una tecnica generalmente utilizzata per identificare relazioni nascoste ma ricorrenti in corrispondenza di un grande numero di transazioni.

Una tipica applicazione di questa tecnica è impiegata nella ricerca di gruppi di prodotti che nei supermercati tendenzialmente vengono spesso comprati insieme. Ai fini del business questa informazione è utilizzata per tecniche di *cross-selling*, scopi promozionali o per orientare il *layout* del punto vendita. L'idea alla base delle regole associative può essere intuitivamente ricollegata al sistema di raccomandazione collaborativo: l'obiettivo potrebbe infatti essere quello di scoprire delle relazioni ricorrenti tra utenti o item. Ad esempio, è possibile stabilire che, se un determinato utente esprime interesse per alcuni item selezionati, allora con grande probabilità potrebbe apprezzare altri item che non ha ancora incontrato ma che hanno una relazione nascosta con i precedenti, date le valutazioni passate di altri utenti. Viene quindi generata una lista di item basati su statistiche in grado di descrivere la co-presenza di un insieme di item all'interno di una grande mole di "transazioni".

Formalmente, indichiamo le transazioni con T, esse sono un sottoinsieme rispetto alla lista di tutti i prodotti disponibili P:[p1,….pm]. Definiamo le regole associative come A→B con A e B che sono entrambi sottoinsiemi di P. Una regola associativa esprime che se gli elementi di A sono contenuti in una transazione T, è altamente probabile che anche gli elementi di B siano contenuti nella stessa transazione. (Sarwar et al., 2000)

L'obiettivo principale degli algoritmi che sfruttano regole associative è scovare in modo automatico determinate regolarità e calcolare la qualità della regola (Agrawal, Srikant, 1994). Le misure più utilizzate per stimare la qualità della regola sono il *supporto* e la *confidenza*. Il *supporto* di una regola A→B è calcolato come la percentuale delle transazioni che contengono tutti gli elementi di A e B rispetto al numero complessivo delle transazioni T. In altre parole, è la probabilità di co-presenza di A e B in una transazione. La *confidenza* è definita come il rapporto tra le transazioni che contengono tutti gli elementi di A e B ed il numero di transazioni che contengono solo A; corrisponde quindi alla probabilità condizionale di B dato A.

Dal punto di vista matematico:

$$supporto = \frac{N° \; transazioni \; contenenti \; X \cup Y}{N° \; transazioni}$$

$$confidenza = \frac{N° \; transazioni \; contenenti \; X \cup Y}{N° \; transazioni \; contenenti \; X}$$

È possibile utilizzare i concetti sopracitati per la creazione di un sistema di raccomandazione collaborativo basato sulle regole di associazione. Considerando per fini esemplificativi la seguente matrice di rating (Tabella 3.2, riadattata da Dietmar et al., 2010), sarà illustrata la logica alla base di questa tipologia di recommendation system e la procedura generalmente utilizzata per generare raccomandazioni *association rule-based*.

	Item 1	Item 2	Item 3	Item 4	Item 5
User X	1	0	0	0	?
User 1	1	0	0	1	1
User 2	1	0	1	0	1
User 3	0	0	0	1	1
User 4	0	1	1	0	0

Tabella 3.2. Matrice rating user-item binaria

All'interno della matrice *user-rating*, le valutazioni degli utenti verso gli item, per semplicità, non sono più misurate su una scala da 1 a 5, bensì è stato assegnato il valore 1 per gli utenti che hanno gradito un determinato item, viceversa il valore 0 per coloro che hanno espresso un giudizio negativo. Come si evince dalla Tabella 3.2 l'obiettivo delle regole associative è scovare delle relazioni nascoste tra user o item al fine di prevedere se un contenuto mai incontrato, Item 5 in questo caso, potrebbe interessare ad un determinato utente, User X in questo esempio.

Algoritmi di rule-mining possono essere utilizzati per analizzare questo piccolo database, calcolando una serie di regole associative e i valori di confidenza e supporto a esse associate. È necessario però focalizzarsi solo su quelle regole che possiamo considerare rilevanti, ovvero che presentano valori di confidenza e supporto che eccedono una soglia minima stabilita all'origine e spesso definita dalla letteratura.

Il calcolo di tutte le possibili regole associative con i corrispettivi valori di supporto e confidenza e le seguenti raccomandazioni seguono generalmente la procedura descritta da Sarwar et al. (2000):

1. Determinare l'insieme di tutte le regole associative A → B rilevanti per l'utente obiettivo. In questo caso l'Utente X ha espresso una valutazione positiva solo per l'item1, quindi risulta logico partire da quest'ultimo per determinare le regole rilevanti.

2. Elencare tutte le possibili combinazioni, focalizzandosi però sui contenuti non ancora incontrati dall'utente obiettivo, per poi dare un ranking che descriva la forza della regola. Per semplicità nella Tabella 3.2 abbiamo un solo contenuto non ancora valutato, l'item 5
3. Ordinare i contenuti da raccomandare in base ai valori di supporto e confidenza della regola. In questo caso se consideriamo come obiettivo, di effettuare una raccomandazione per l'Utente X, la regola associativa che verrà valutata sarà
   Item 1 → Item 5. Sarà verificata quindi, sulla base delle precedenti interazioni degli altri utenti con i contenuti (matrice *user-rating*), la probabilità che chi abbia espresso un feedback positivo per l'item 1 esprimerà una valutazione altrettanto positiva per l'item 5. Con l'ausilio delle metriche descritte in precedenza, il supporto di questa regola sarà 0.5 poiché due utenti su quattro hanno espresso un interesse sia per l'item 1 che l'item 5, mentre il valore di confidenza sarà 1 poiché in questo caso un giudizio positivo sul primo contenuto ha implicato sempre un feedback positivo anche sull'item 5. Nel caso in cui questi *score* siano considerati al di sopra dei valori soglia, allora l'item 5 sarà raccomandato allo User X.

3.2.3 Clustering Recommendation system

La *cluster analysis* è un insieme di tecniche di analisi multivariata dei dati volte alla selezione e raggruppamento di elementi omogenei in un insieme di dati [16]. Essa appartiene alle metodologie di *machine learning* non supervisionato, poiché ricade in quelle tecniche che tramite l'esplorazione dei dati, hanno lo scopo di scovare relazioni e strutture intrinseche al loro interno. L'obiettivo finale è trovare nel *dataset* dei gruppi, propriamente detti *cluster,* i cui oggetti sono simili (relazionati) tra di loro all'interno del gruppo, mentre sono diversi (non relazionati) da quelli assegnati agli altri gruppi (Figura 3.4). Più grande è la similarità all'interno del raggruppamento, più grande è la differenza tra essi e migliore sarà il *clustering*. I vantaggi in termini di business che questa tecnica può essere in grado di fornire sono molteplici, basti pensare alle modalità con cui si effettua una segmentazione del mercato, alla scelta di promozioni *ad hoc* per determinati gruppi di clienti o la creazione di sistemi di raccomandazione. Quest'ultimo argomento sarà approfondito in seguito, dopo aver prima di tutto dato una *overview* sul funzionamento del *clustering*, le sue diverse classificazioni e gli algoritmi di cui si serve.

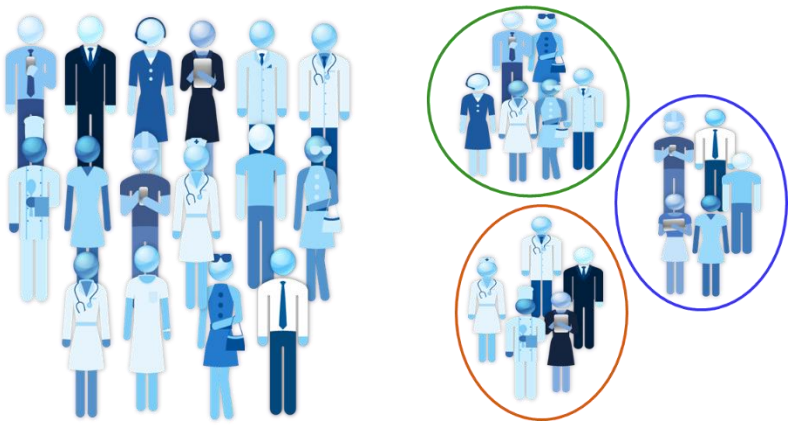

Figura 3.4. Clustering [17]

Funzionamento: il punto di partenza è ovviamente la matrice dei dati *n x p*, costituita da *n* unità statistiche (osservazioni) e da *p* variabili. È necessario poi trasformare quest'ultima in una matrice delle distanze/similarità *n x n* tra le coppie di osservazioni. Si calcolano le distanze in presenza di variabili quantitative, la similarità in corrispondenza di dati qualitativi. Dopo di che si sceglie uno specifico algoritmo che detti le regole sulla modalità di raggruppamento delle unità in *cluster* sulla base della loro distanza/similarità [18] .

Il calcolo della distanza tra unità statistiche risulta essere quindi il punto di origine, basilare per qualsiasi tecnica di *clustering*. È perciò necessario comprendere il modo in cui essa viene calcolata. Prendendo in considerazione una matrice di dati, costituita da *n* osservazioni e dunque da *n* vettori riga, ciascuno rappresenta una unità statistica e contiene quindi *p* elementi, ovvero i valori assunti dalla *j*-esima variabile. È possibile calcolare la distanza tra due unità statistiche, *i* ed *h* in modi differenti. La misura più generale è la Distanza di Minkowsky:

$$d(i,h) = \left[\sum_j |x_{ij} - x_{hj}|^k\right]^{\frac{1}{k}}$$

k è definito come l'ordine della distanza e può assumere 3 diversi valori. A seconda di essi avremo 3 modalità per calcolare le distanze tra unità statistiche:

- per *k*=1 si ricava la Distanza di Manhattan

- per k=2 la Distanza Euclidea
- per $k \rightarrow \infty$ la Distanza di Lagrange

Metodi di classificazione: Dopo aver misurato le distanze è necessario scegliere il metodo di classificazione da utilizzare. Abbiamo due principali macrocategorie di riferimenti: i metodi gerarchici, aggregativi e scissori, e quelli non gerarchici.

I metodi gerarchici: definiscono i *cluster* aggregando o dividendo le unità statistiche. Nei metodi scissori si inizia da una partizione formata da un unico *cluster* contenente tutte le unità e in seguito avviene una disaggregazione, finché ogni elemento va a formare un *cluster* a sé stante. Viceversa, in quelli aggregativi, nella partizione iniziale ciascun oggetto forma un proprio *cluster* e successivamente vengono aggregati fino a formare un unico ampio *cluster*. Il risultato finale che la metodologia gerarchica fornisce è una serie di partizioni, rappresentate graficamente da un dendogramma: esso presenta sull'asse delle ascisse l'insieme di tutte le unità statistiche e sull'asse delle ordinate il livello di distanza. Ogni ramo del dendogramma corrisponde ad un *cluster*, mentre la linea che congiunge i rami rappresenta la distanza in cui le unità si aggregano (Figura 3.5).

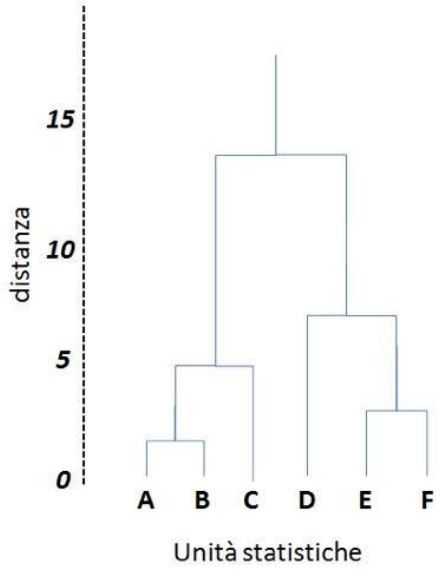

Figura 3.5. Clustering: metodo gerarchico [19]

Nel processo di aggregazione vengono utilizzati diversi criteri che vanno a definire il concetto di distanza, e che sono responsabili dell'assegnazione delle unità statistiche ai *cluster*.

Descriviamo quelli più famosi evidenziando le principali differenze:

- Nel metodo del **legame singolo** la distanza tra gruppi è considerata come la più piccola delle distanze calcolabili a due a due tra tutti gli elementi. Ciò favorisce la formazione di *cluster* costituiti da elementi omogenei piuttosto che la ricerca di una differenza netta tra *cluster* diversi.
- Nel metodo del **legame completo**, viceversa, si esamina la maggiore delle distanze. Qui è privilegiata la differenza tra gruppi rispetto all'omogeneità degli elementi del *cluster*.
- Nel metodo del **centroide** si considera invece la distanza tra i centroidi dei gruppi, cioè i baricentri. Questo criterio tende a bilanciare i pro e i contro degli altri due metodi.

La scelta finale del numero ottimo di *cluster* è effettuata utilizzando la cosiddetta distanza di fusione. Essa è osservabile semplicemente dal dendogramma: se nel passaggio da X *cluster* a X+1 si evidenzia un incremento particolarmente importante del livello di distanza, bisogna fermarsi a X gruppi, che quindi diventano il numero di *cluster* ottimali [20].

I metodi non gerarchici: a differenza dei metodi gerarchici, essi sono caratterizzati da una procedura che ripartisce direttamente le unità statistiche in X *cluster*, fornendo quindi una partizione unica. Inoltre, caratteristica distintiva è che una volta che un elemento entra a far parte di un *cluster* sono possibili spostamenti, con lo scopo di ottenere una partizione ottimale. Tuttavia, il metodo si basa su una definizione apriori del numero di *cluster*; ciò può comportarne un limite, in quanto la classificazione finale è influenzata dalla partizione iniziale.

Il procedimento è descritto nelle seguenti fasi:

1. Scelta dei centri
2. Raggruppamento delle unità statistiche intorno ai centri in modo che ogni elemento appartenga al gruppo il cui centro è più vicino ad esso.
3. Calcolo dei centroidi degli n *cluster*
4. Calcolo della distanza di ciascuna unità statistica da ogni centroide; se la distanza minima non risulta quella con il centroide del *cluster* di appartenenza allora l'unità viene riallocata al gruppo il cui centroide è più vicino
5. Ricalcolo dei centroidi

6. Fasi 4 e 5 vengono ripetute in modo iterativo fino a raggiungere una situazione in cui non sono necessari ulteriori spostamenti.

3.2.4 Clustering applicato ai sistemi di raccomandazione collaborativi

Il principio alla base del sistema di raccomandazione che utilizza la tecnica di *clustering* risiede nell'assunzione che utenti appartenenti ad uno stesso *cluster* abbiano gli stessi interessi e quindi tendono a esprimere valutazioni simili rispetto agli item che gli vengono proposti (Thi Do et al., 2010). Considerando gli utenti u_i e le valutazioni espresse per gli item r_{in} ricaviamo che ogni utente è rappresentato da un rating vector $u_i = (r_{i1,}\ r_{i2,}\ r_{in})$ che rappresenta l'insieme di tutte le valutazioni espresse rispetto ai contenuti presenti. La misura di somiglianza tra due utenti è uguale alla distanza tra le loro valutazioni, calcolabili con i metodi descritti nei paragrafi precedenti. Minore è la distanza $(u_{1,}\ u_n)$ tra essi più gli utenti saranno considerati simili.
Il *Clustering collaborative filtering* prevede due step:

1. Ripartire gli utenti in *cluster*; ciascuno ricaverà dai loro rating una media delle valutazioni degli utenti per ogni item, andando a definire un *rating vector* del *cluster*.
2. L'utente che ha bisogno della raccomandazione rispetto un determinato item, viene assegnato in base al suo profilo ad uno dei *cluster* ed il suo rating probabile, previsto per quel contenuto, sarà associato al rating medio del cluster di riferimento.

Risulta tuttavia importante capire le modalità con cui un utente venga assegnato al giusto *cluster*. Tra le diverse tecniche di *clustering*, la più utilizzate per questo tipo di applicazioni è la metodologia non gerarchica, che si serve in particolare dell'algoritmo *k-means*. Essa segue quindi la procedura descritta nel paragrafo precedente, configurando gli utenti come le unità statistiche e le loro valutazioni come le variabili tenute in considerazione per il calcolo delle distanze. (Figura 3.6)

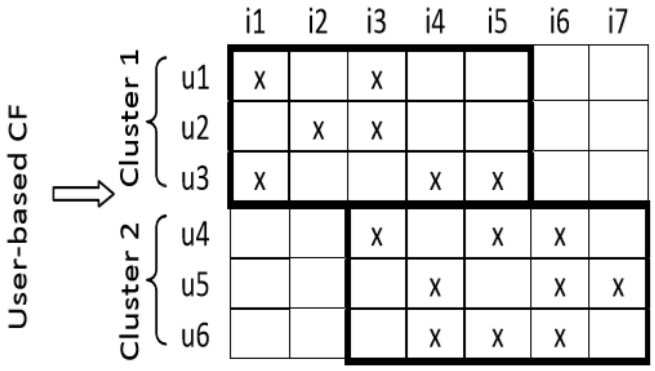

Figura 3.6. Clustering applicato al Recommendation System

3.3 Content Filtering

A differenza delle metodologie precedentemente descritte, I sistemi di raccomandazione che si fondano su un approccio *Content-based* suggeriscono gli item agli utenti basandosi sull'analisi del loro contenuto e dello *user-profile*. Il contenuto di ciascun item è rappresentato da un insieme di "descrittori" la cui tipologia varia a seconda dell'ambito di applicazione, mentre il profilo dell'utente viene dedotto dai suoi comportamenti (impliciti o espliciti) nei riguardi degli item che gli vengono man mano proposti. In sostanza, il sistema cerca di raccomandare articoli dal contenuto simile agli utenti che hanno già espresso interesse per quella tipologia di contenuti. (Balabanovic et al., 1997)

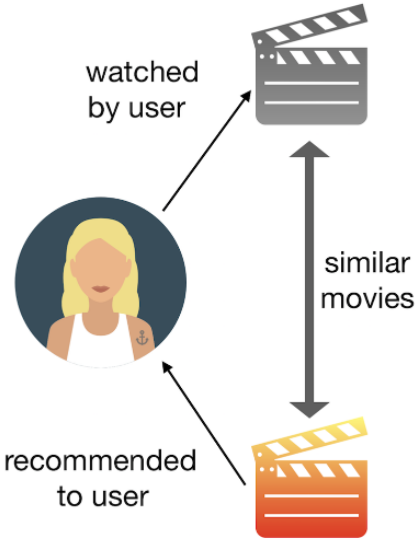

Figura 3.7. Content-Based filtering [21]

È possibile dunque schematizzare il processo di raccomandazione in 3 fasi (Lops et al., 2011) illustrate nella Figura 3.8

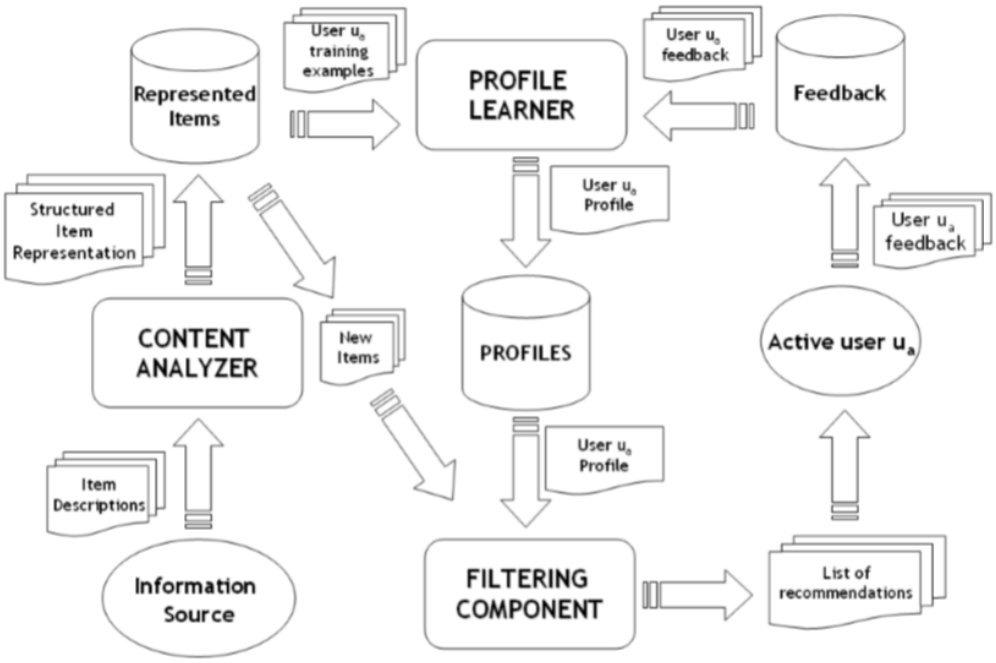

Figura 3.8. Processo di raccomandazione Content-Based

- ***Content analyzer***: fase in cui avviene la classificazione degli item. I contenuti degli item non sono sempre facilmente rappresentabili dai suoi "descrittori", vengono infatti utilizzate tecniche ad hoc per estrarre le caratteristiche che li rappresentano in modo sintetico. Sono proprio queste ultime ad essere il punto di partenza delle due fasi successive.
- ***Profile learner***: fase in cui vengono raccolti i dati in grado di rappresentare le preferenze dell'utente, funzionali alla creazione del suo profilo. In genere, sono utilizzate tecniche di *machine learning* capaci di attribuire al comportamento dell'utente un modello dedotto dalle valutazioni degli item incontrati in passato.
- ***Filtering component***: utilizzando come input lo *user-profile* e combinandolo con le caratteristiche degli item, suggerisce agli utenti, sottoforma di *recommandation list*, solo quei contenuti che risultano per lui rilevanti.

3.3.1 Tecniche di analisi del contenuto: Vector Spece Model e Term Frequency-Inverse Document Frequency (TD-IDF)

Punto di partenza essenziale di un sistema *content-based* è la corretta rappresentazione degli item sulla base del loro contenuto e quindi delle proprie caratteristiche. Esse, vengono generalmente dedotte dai cosiddetti "descrittori". Per essi si intende attributi, *keywords* o descrizioni testuali in grado di etichettare gli item; ad esempio, se consideriamo la raccomandazione di un film, gli attributi chiave potrebbero essere gli attori, il regista, il genere, il tema. Tuttavia, nella maggior parte dei casi i dati su cui costruire la rappresentazione dell'item non sono strutturati, spesso sono infatti testuali. Contestualizzando quindi, l'item come un documento testuale e il suo contenuto come un insieme di termini, l'obiettivo è quello di identificare quelle parole che meglio descrivono il contenuto del documento. La maggior parte dei sistemi di raccomandazione utilizza come tecnica principale il *Vector Space Model* (VSM). Essa è una tecnica utilizzata nell' *Information Retrivial* (IR) e fornisce una rappresentazione spaziale di documenti di testo. Ogni documento è rappresentato da un vettore in uno spazio *n*-dimensionale, in cui ciascuna dimensione corrisponde ad un termine dal vocabolario complessivo di una determinata collezione di documenti. Dopo di che, ogni documento è rappresentato da un vettore di termini opportunamente ponderati, dove il peso $w_{i,j}$ indica il grado di associazione tra il documento ed il termine. Formalmento avremo:

$J = \{j_1, j_2, \ldots, j_N\}$ rappresenta l'insieme di tutti i documenti

$I = \{i_1, i_2, \ldots, i_K\}$ indica l'insieme di tutti i termini

Le procedure da seguire per questa tipologia di tecnica sono le seguenti [22]:

1. Creare un dizionario complessivo di tutte i termini presenti nei documenti ("*Bag of words*")
2. Formare per ogni documento un vettore contenente il dizionario di termini, in cui verrà assegnato valore 1 se il termine è presente, viceversa il valore 0
3. Assegnare un peso $w_{i,j}$ ad ogni termine all'interno di ciascun documento
4. Misurare la *cosine similarity* tra i vettori per comprendere la somiglianza tra i documenti.

L'algoritmo più utilizzato per assegnare un peso ai termini è *Term Frequency-Inverse Document Frequency* (TD-IDF). Esso parte dall'assunzione che termini occorrenti in modo frequente in un documento (*term frequency*) ma raramente nel resto dei documenti (*inverse document frequency*) è più probabile che siano rilevanti nel descrivere l'argomento di un documento.

Dati:

$tf_{i,j} =$ numero di occorrenze del termine *i* nel documento *j*

$\mathrm{d}f_i =$ numero di occorrenze dei documenti contenenti il termine *i*

$N =$ numero totale di documenti

Il peso $w_{i,j}$ di un termine *j* all'interno di un documento *i* può essere esplicitato matematicamente con la seguente espressione:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Di conseguenza, un documento sarà rappresentato da un nuovo vettore contenente per ogni termine uno score $w_{i,j}$ che spiega la rilevanza di ciascuna parola nella descrizione del suo contenuto, fungendo praticamente da etichetta.

A questo punto per poter proiettare su uno spazio *n*-dimensionale i vettori è necessario calcolare l'ampiezza dell'angolo tra essi tramtie la *cosine similarity*. Successivamente sono calcolati anche i vettori che identificano gli *user-profile* sulla base dei loro comportamenti nei confronti dei contenuti già incontrati. Infine, è necessario calcolare la *coisine similarity* anche tra gli item e gli utenti in modo tale da poter stimare ogni utente per quali attributi prova interesse, raccomandandogli solo quegli item che contengono quelle determinate caratteristiche.

Di seguito (Figura 3.9), una rappresentazione grafica semplicistica del *Vector Space Model* in cui gli attributi chiave sono i termini *orange, banana, apple.* Grazie alla rappresentazione vettoriale dei documenti nella 3 dimensioni date dai termini è possibile identificare in che misura essi sono in grado di descrivere gli item ed il loro livello di somiglianza. Infine, in base alla posizione dell'utente obiettivo nello spazio dimensionale sarà effettuata la raccomandazione.

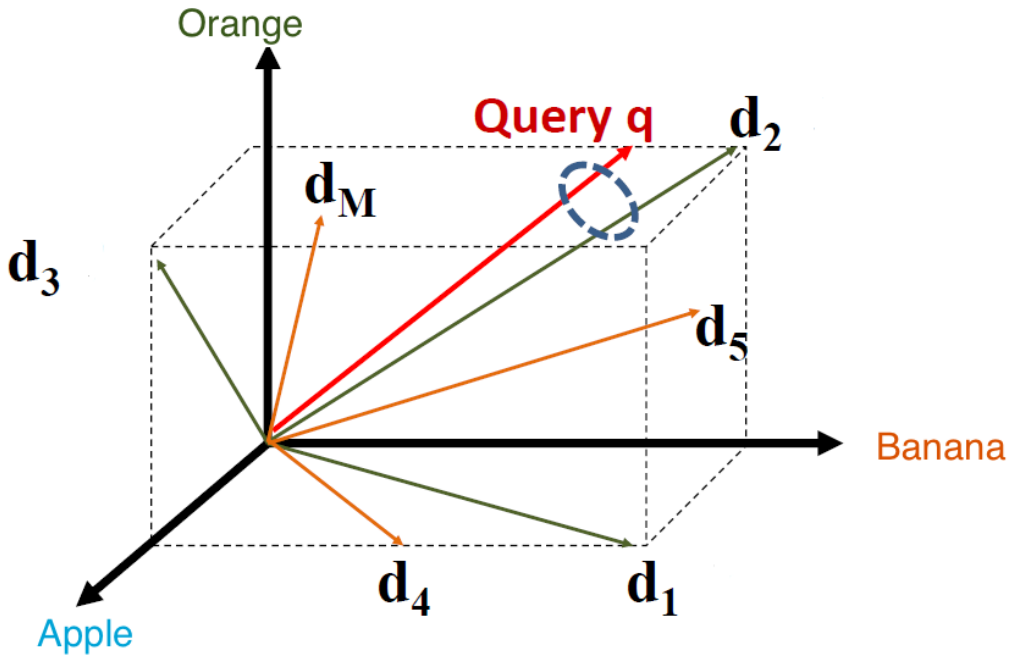

Figura 3.9. Vector Space Model [23]

3.3.2 Content filtering e Collaborative filtering a confronto

Compresa la logica alla base dei sistemi di raccomandazione *content-based* e notate le sostanziali differenze con i sistemi collaborativi, è bene analizzarne vantaggi e svantaggi:

Vantaggi:

- ***User-indipendence*:** al contrario dei sistemi collaborativi che necessitano delle valutazioni di tanti utenti, per poter identificare una similarità fra essi. Quelli *content-based* hanno bisogno soltanto dei rating forniti dall'utente oggetto della *recommendation*, finalizzati alla costruzione del suo profilo. Ciò apporta inevitabilmente un vantaggio in termini di quantità di dati e velocità di *processing*.
- **Trasparenza:** la raccomandazione fornita dai sistemi collaborativi risulta spesso una "scatola nera" in quanto l'utente non conosce gli utenti a cui è associato e di conseguenza ignora la motivazione del suggerimento. Nei sistemi *content-based* la spiegazione della raccomandazione può essere facilmente fornita mostrando le caratteristiche (descrittori) responsabili dell'inserimento di un determinato item nella *recommendation list*.

- **Item nuovi:** i sistemi collaborativi soffrono del "*cold-start*", e quindi per poter essere in grado di raccomandare un contenuto hanno bisogno che sia valutato da un numero sostanziale di utenti. Quelli *content-based* sono capaci di raccomandare item non ancora valutati dagli utenti in quanto il sistema non si basa solo sulle preferenze degli utenti bensì sul contenuto dell'item.

Svantaggi:

- **Contenuto da analizzare limitato:** la tecnica *content-based* ha un suo limite naturale nel numero e nella tipologia di caratteristiche associate all'oggetto della raccomandazione. Inoltre, per poterle correttamente classificare c'è bisogno di conoscere il dominio del campo di applicazione. Il sistema, quindi, non sarà capace di effettuare le sue raccomandazioni se il contenuto da analizzare non contiene abbastanza informazioni per discriminare se un determinato item piaccia o meno all'utente.
- **Iper-specializzazione:** la logica alla base dei sistemi *content-based* determina che agli utenti vengono raccomandati solo i contenuti che hanno una corrispondenza con il loro profilo; di conseguenza gli verranno consigliati solo gli item simili a quelli già incontrati. Ciò genera una serie di raccomandazioni che potrebbero risultare scontate e poco originali.
- **Utenti nuovi:** mentre l'aggiunta di nuovi contenuti non rappresenta un problema, l'approccio *content-based* soffre l'ingresso di nuovi utenti nel sistema. C'è infatti la necessità di raccogliere un grande numero di valutazioni prima che il sistema riesca a capire le preferenze dell'utente e costruire un profilo accurato in grado di generare raccomandazioni affidabili.

3.4 Hybrid filtering

I recommendation system ibridi combinano due o più tipologie di sistemi con lo scopo di effettuare raccomandazioni più accurate. L'obiettivo quindi è assicurare una performance complessivamente migliore, andando a mitigare gli svantaggi sia dei sistemi collaborativi che di quelli *content-based*. L'idea alla base risiede nel fatto che grazie alla combinazione di diversi algoritmi, gli svantaggi di uno possono essere superati con all'impiego di un altro algoritmo.

Netflix è un ottimo esempio dell'utilizzo di un sistema di raccomandazione ibrido (Gomez-Uribe et al., 2015). Esso infatti, suggerisce film agli utenti sfruttando sia la somiglianza di preferenze e abitudini di ricerca tra gli utenti (*collaborative filtering*) che offrendo film con caratteristiche simili a quelli che gli utenti hanno valutato positivamente (*content-based filtering*).

L'unione di diversi approcci può essere eseguita in diversi modi: implementando separatamente gli algoritmi e combinando i risultati, utilizzando tecniche *content-based* all'interno di un sistema collaborativo, impiegando tecniche collaborative all'interno di un sistema basato sul contenuto oppure creando un sistema completamente unificato.

Burke (2002) ci fornisce una classificazione dei metodi ibridi più diffusi:

Ibridazione ponderata: essa effettua le proprie raccomandazioni combinando i risultati di diversi *recommenders* e integrando gli *score* di ciascuna tecnica utilizzata tramite una combinazione lineare di essi. Un esempio di un sistema con tali caratteristiche è il *P-tango* (Calypso et al. 1999); esso unisce l'approccio collaborativo con quello *content-based*. Inizialmente vengono assegnati pesi uguali ai due approcci, per poi eventualmente correggerli in base al risultato corretto o meno delle predizioni. Il beneficio di un'ibridazione ponderata sta nel fatto che tutti i punti di forza dei diversi approcci sono utilizzati in modo diretto durante il processo di raccomandazione.

***Switching hybridization*:** questa tipologia di sistema ha la capacità di passare da un approccio all'altro secondo regole euristiche, alternandoli a seconda della situazione e cercando di aggirare le limitazioni specifiche di ciascuna tecnica. Un esempio di sistema di questo tipo è il *DailyLearner* (Billsus e Pazzani, 1999); esso parte con un sistema *content-based* e nel caso in cui si fronteggia un problema, come l'ingresso di nuovi utenti, avviene il passaggio verso un approccio collaborativo. Tuttavia, questa tecnica ha lo svantaggio di apportare una maggiore complessità al processo di raccomandazione a causa del continuo passaggio da un approccio all'altro.

Ibridazione a cascata: essa impiega un processo di affinamento iterativo. Una prima tecnica di raccomandazione è impiegata per costruire un *ranking* iniziale di preferenze tra i diversi item da raccomandare, una seconda invece, si occupa di rifinire le raccomandazioni a partire dal set dato dalla tecnica precedente.

Ibridazione mista: combina i risultati di differenti tecniche di raccomandazione allo stesso tempo. Invece di avere una sola raccomandazione per item, ognuno ha una raccomandazione multipla che deriva da approcci differenti.

Feature-combination: le caratteristiche generate da una specifica tecnica di raccomandazione sono inserite all'interno di un'altra tecnica. Ad esempio, il calcolo delle somiglianze fra utenti, che è una caratteristica tipica di un approccio collaborativo può essere incorporato in un sistema *content-based*.

Meta-level: due tecniche di raccomandazione sono combinate utilizzano il modello generato da una tecnica come input per un'altra. Di conseguenza, il modello diventa sempre più ricco di informazioni. Questa tipologia di sistema è in grado di risolvere il problema della sparsità, tipico degli approcci collaborativi. Un esempio di tecnica *Meta-Level* è il *LaboUr* (Schwab, et al. 2001) che usa un apprendimento istantaneo per creare profili degli utenti *content-based*, comparandoli poi in modo collaborativo.

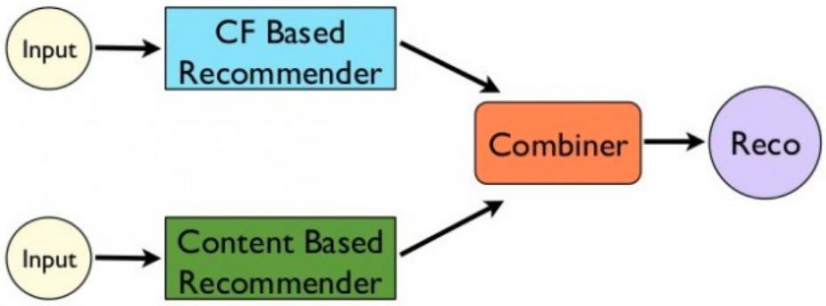

Figura 3.10. Hybrid Recommendation system

3.5 Metodi di valutazione

La qualità di un algoritmo di raccomandazione può essere giudicata valutandone l'accuratezza. Essa è considerata come il rapporto tra le raccomandazioni effettuate correttamente sul totale delle possibili raccomandazioni che il sistema avrebbe potuto effettuare. Hernandez del Olmo (2008) definisce le metriche di accuratezza come la capacità del recommandation system di avvicinarsi alla realtà e utilizza la seguente espressione per calcolarla:

$$Accuratezza = \frac{n° \; di \; raccomandazioni \; riuscite}{n° \; totale \; di \; raccomandazioni}$$

Tuttavia, generalmente vengono usate unità di misura più flessibili per misurare l'accuratezza di un sistema di raccomandazione; esse sono divise in due categorie: metriche statistiche di accuratezza e metriche di supporto decisionale (Sarwar et al., 2001). L'adeguatezza di ciascuna metrica dipende dalle caratteristiche del *dataset* e dalla tipologia del sistema da implementare (Friedman et al., 1997).
Di seguito saranno descritte le due macrocategorie e i relativi strumenti di misurazione.

3.5.1 Metriche statistiche di accuratezza

Questa tipologia di metriche valuta l'accuratezza delle raccomandazioni comparando i rating predetti direttamente con i rating effettuati degli utenti. Le unità di misura maggiormente utilizzate appartenenti a questa categoria sono *Mean Absolute Error* (MAE) e *Root Mean Square Error* (RMSE)
MAE è la metrica più popolare e comunemente impiegata; essa misura la deviazione dei rating raccomandati dagli specifici valori che gli utenti in seguito assegnano agli item. Definendo *u* gli utenti *e i* gli item, consideriamo *p* i rating attuali (reali) che gli utenti assegnano ai contenuti e *r* i rating predetti dal sistema. *N* invece è il numero totale di rating riguardo l'interno *item set*.
Formalizzando la metrica dal punto di vista matematico è possibile calcolarla grazie alla seguente formula (Claypool et al. 1999):

$$MAE = \frac{\sum_{\{u,i\}} \left| p_{u,i} - r_{u,i} \right|}{N}$$

Minore è il valore MAE, più il motore di raccomandazione è considerato accurato nel predire correttamente le valutazioni degli utenti riguardo gli item proposti.
Altrettanto utilizzata è la metrica *Root Mean Square Error* (RMSE); essa pone una maggiore enfasi sulla grandezza assoluta dell'errore e come in precedenza, più basso è il suo valore, migliore risulterà l'accuratezza delle raccomandazioni.
Dal punto di vista matematico è possibile calcolarla tramite la seguente formula (Cotter et al., 2000):

$$RMSE = \sqrt{\frac{\sum_{\{u,i\}} \left(p_{u,i} - r_{u,i} \right)^2}{N}}$$

Le metriche statistiche di accuratezza trattano tutti gli item in modo equo. Tuttavia, per la maggior parte dei sistemi di raccomandazione la principale preoccupazione non è effettuare il maggior numero di predizioni corrette rispetto ai rating, piuttosto riuscire a prevedere in modo accurato le valutazioni di quegli item per cui un utente proverà interesse. Motivo per cui i ricercatori considerano una predizione "buona" per gli item che evidenziano valutazioni elevante, viceversa, "cattivi" per quelli che manifestano rating bassi. Nel contesto dell'*informatinal retrivial* (IR), identificare gli item buoni da quelli cattivi è definito in modo più formale come una differenza fra i contenuti "rilevanti" e quelli "non rilevanti" per gli utenti.

3.5.2 Metriche di supporto decisionale

Questa macrocategoria di metriche si ricollega direttamente alla problematica appena descritta. Esse aiutano gli utenti a selezionare solo gli item che sono considerati per loro qualitativi rispetto alla totalità del numero di item disponibili. (Sarwar et al., 1998); questa tipologia di metriche configurano la procedura di predizione come una operazione binaria che distingue gli item rilevanti da quelli non rilevanti. Le metriche più famose appartenenti a questa categoria sono: *Precision*, *Recall* e *Receiver Operaring Characteristics* (ROC).

Per comprendere al meglio questo tipo di metriche è necessario introdurre il concetto di Matrice di Confusione: nella classificazione statistica, essa è una tabella in cui le previsioni sono rappresentate nelle colonne e lo state reale è mostrato all'interno delle righe (Figura 3.11).

		Reality		
		Actually Good	Actually Bad	
Prediction	Rated Good	True Positive (tp)	False Positive (fp)	All recommended items
	Rated Bad	False Negative (fn)	True Negative (tn)	
		All good items		

Figura 3.11. Matrice di confusione (Pelanek, 2018)

Da questa tabella è possibile comprendere la performance di un modello predittivo di classificazione in modo da determinarne l'efficacia e l'accuratezza.

Contestualizzando la matrice riguardo i sistemi di raccomandazione avremo:

- *True Positive:* i contenuti predetti come rilevanti e valutati allo stesso modo dagli utenti.
- *False Positive:* i contenuti predetti come rilevanti ma valutati negativamente dagli utenti.
- *False Negative:* i contenuti predetti come non rilevanti ma ritenuti rilevanti per dagli utenti.
- *True Negative:* i contenuti predetti come non rilevanti e valutati allo stesso modo dagli utenti.

Da essi è poi possibile ricavare:

- *True Positive Rate*: la percentuale degli item rilevanti che sono stati raccomandati.
- *False Positive Rate*: la percentuale degli item non rilevanti che sono stati raccomandati.

Sulla base di questi concetti vengono poi misurate le metriche precedentemente elencate:

Precision (Manning et al. 2008) è considerata come il rapporto tra il numero di item rilevanti raccomandati e il numero totale di item raccomandati (rilevanti e non rilevanti).

$$Precision = \frac{n° \; items \; rilevanti \; raccomandati}{n° \; totale \; items \; raccomandati}$$

Recall invece, è definito da Manning (2008) come il rapporto tra il numero di item rilevanti raccomandati e il numero di tutti i possibili item rilevanti.

$$Recall = \frac{n° \; items \; rilevanti \; raccomandati}{n° \; totale \; items \; rilevanti}$$

Dalla letteratura (Sarwar et al., 2001) è possibile però ritrovare una metrica che è in grado di sintetizzare le due suddette unità di misura in un solo indicatore*:*

F-measure. Lo *score* è interpretabile come la media ponderata di *Precision* e *Recall*, dove il risultato di massima accuratezza si ha quando il valore è 1, mentre nel caso peggiore il valore sarà 0. Esso è molto utile in quanto rende semplice e chiara la comparazione dei risultati di un sistema di raccomandazione tra diversi algoritmi e *dataset*:

$$F\ measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Ulteriore unità di misura utilizzata è ***Receiver Operaring Characteristics Curve*** (ROC Curve): essa è un metodo grafico per la valutazione di un classificatore binario. Poiché per questa tipologia di problemi è necessario scegliere un valore soglia, determinante nella scelta dell'appartenenza a una delle due classi (positiva o negativa), la curva ROC rappresenta tutti i valori di *True Positive Rate* e *False Positive Rate* al variare del valore soglia.

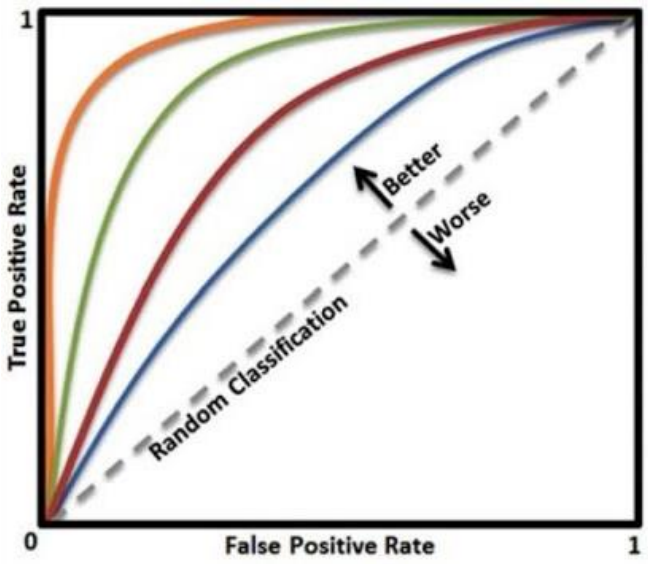

Figura 3.12. ROC Curve

Il classificatore è tanto migliore quanto più la ROC è spostata verso nord-ovest, in quanto l'area al di sotto di essa (AUC) esprime il potere diagnostico del *test*: essa equivale alla probabilità che scelto un item casualmente esso sia correttamente classificato come rilevante o non rilevante per l'utente. La peggiore curva possibile coincide con la diagonale, fornisce un valore informativo del test nullo e corrisponde ad un AUC = 0.5. La ROC che coincide con i due lati fornisce invece il potere

informativo massimo, significa che le classificazioni non presentano errori e corrisponde ad un valore AUC = 1.

3.5.3 Diversità, Novità e Coverage

L'accuratezza però non è tutto, infatti, un utente considererà soddisfacente una serie di raccomandazioni se esse contengono contenuti che non conosce ma che allo stesso tempo sposano i suoi gusti (McNee, 2006). Appunto per differenziare l'accuratezza dalla soddisfazione o utilità sono state introdotte metriche di Diversità, Novità e *Coverage*. La Novità si riferisce a quanto un item proposto è differente rispetto a contenuti già incontrati in precedenza (Vargas e Castells, 2011). La *Coverage* si rifà alla percentuale di item che un sistema di raccomandazione riesce a produrre per un utente. La Diversità generalmente si applica ad un insieme di item, e si riferisce a quanto i contenuti raccomandati siano differenti fra loro. Ziegler (2005) propone la metrica *Intra-List Similarity* per misurarla, dove per *L* intendiamo la *Recommendation list*, per i_j e i_k due item:

$$ILS_L = \frac{1}{2}\sum_{i_j}\sum_{i_k} sim(i_j, i_k)$$

CAPITOLO 4

Analisi: implementazione di sistemi di raccomandazione per servizi Video Streaming on Demand con R studio

Dopo aver dato una ampia visione sulla rilevanza dei sistemi di raccomandazione e sulle varie tipologie di approcci e algoritmi di cui si serve, il quarto capitolo sarà dedicato ad una analisi personale volta all'implementazione di un vero e proprio sistema di raccomandazione tramite l'utilizzo del software statistico R studio.
Quest'ultimo capitolo sarà organizzato secondo la seguente struttura:

- Spiegazione dell'obiettivo dell'analisi e del campo di applicazione
- Descrizione dell'origine e del contenuto del *dataset*
- Metodologia seguita per l'analisi
- Implementazione e valutazione dei modelli sviluppati
- Confronto modelli e discussione dei risultati

4.1 Obiettivo dell'analisi e campo di applicazione

Lo scopo della seguente analisi sarà fornire empiricamente la metodologia con cui, a partire da un determinato *dataset*, è possibile costruire operativamente un sistema di raccomandazione. Nello specifico, saranno implementati recommandation system basati su un approccio collaborativo ed in particolare verranno messi a confronto i due algoritmi più utilizzati in letteratura, lo *User-based Collaborative Filtering* e l'*Item-based collaborative filtering*. Dopo aver costruito i motori di raccomandazione ne saranno discussi i risultati ed evidenziati le peculiarità non che i limiti dell'uno e dell'altro, con l'obiettivo di verificare se essi risultino o meno coerenti con lo stato dell'arte. Il campo di applicazione è circoscritto al settore cinematografico.

Con il diffondersi del web, la produzione di film, serie tv, cortometraggi o anche semplicemente materiali video creati dagli utenti, hanno avuto una crescita esponenziale e le prospettive future sembrano andare ancora in questa direzione. Inoltre, la modalità di visualizzazione dei contenuti ha avuto un radicale cambiamento negli anni; grazie alla diffusione di servizi di video-streaming on demand (VOD), quali Netflix, Amazon Prime, Plus and Hulu, gli utenti tendono a privilegiare sempre di più questi ultimi alle modalità tradizionali di visualizzazione. Il beneficio di poter decidere autonomamente quando vedere un film oppure una serie tv, avendo la possibilità di guardarla tutta di un fiato è mitigato però dalla enorme grandezza del catalogo di titoli a disposizione. La vastità dei titoli disponibili induce nell'utente una sensazione di incertezza che ricade talvolta in un freno all'utilizzo delle piattaforme. Motivo per cui risulta cruciale avere un sistema di raccomandazione che scelga per noi, suggerendo o mettendo in primo piano i film coerenti con i propri interessi; è infatti questo uno dei motivi del successo delle piattaforme Streaming Video On Demand.

Nella seguente analisi si andrà proprio a simulare la situazione appena descritta. Compito del sistema di raccomandazione implementato, sarà quindi fornire una *recommendation list* per ogni utente, in cui gli item in questione da suggerire saranno titoli di film/serie tv non ancora visionati e per i quali gli utenti potrebbero provare interesse.

4.2 Origine e contenuto del dataset "Movielens"

Il *dataset* oggetto di analisi, si chiama *"Movielens"* e contiene più di 100 milia rating fatti da 610 utenti per quasi 10 mila film. Esso è tratto dall'omonimo sistema di raccomandazione via web che, fornisce ai suoi utenti suggerimenti su quale film vedere sulla base dei loro interessi potenziali. È stato ideato da *"Grouplens"*, un laboratorio di ricerca nel Dipartimento di *Computer Science* and *Engineering* dell'Università del Minnesota, "Twin Cities", specializzato in sistemi di raccomandazione e studio di comunità online. Movielens è un sito web senza scopi di lucro, bensì con finalità di ricerca, e offre a chiunque la possibilità di contribuire, iscrivendosi al sito ed entrando a far parte della *community*. Una volta registrati, agli utenti vengono proposti casualmente un determinato numero di film oppure gli viene data la possibilità di cercare autonomamente i titoli che preferiscono. A questo punto gli viene chiesto di esprimere una propria valutazione da 0 a 5 per un numero minimo

di 15 titoli, tali da poter iniziare a costruire un profilo utente attendibile nella rappresentazione delle sue preferenze. Ecco che, grazie al sistema di raccomandazione collaborativo integrato nel sito web, spunta una sezione nominata *"top picks"* in cui viene presentata all'utente una lista di film suggeriti con il rispettivo *pradicted rating* per ciascuno di essi (Figura 4.1). Al crescere del numero di valutazioni effettuate, crescerà anche l'attendibilità del profilo e di conseguenza le previsioni del sistema di raccomandazione saranno sempre più accurate.

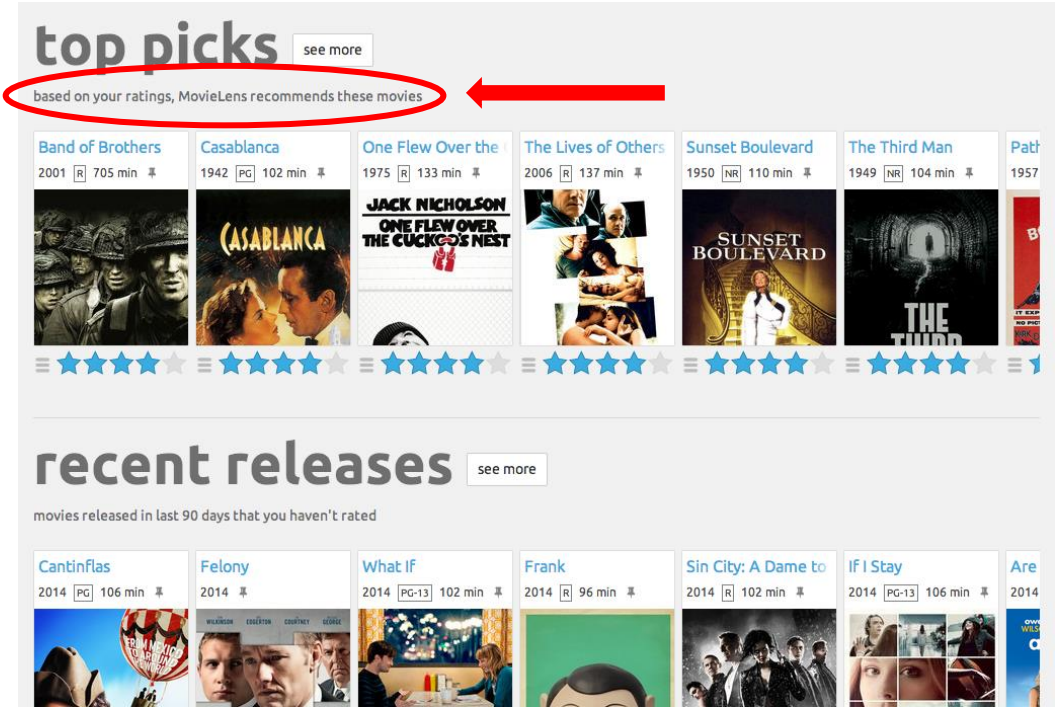

Figura 4.1 Homepage Movielens.org

In più, oltre alla facoltà di avere maggiori informazioni sul film, come *trailer*, immagini, *cast* e anno di *release*, all'utente viene data la possibilità di cercare i titoli per *tag*, ovvero etichette descrittive del suo contenuto, e di aggiungerne dei propri (Figura 4.2).

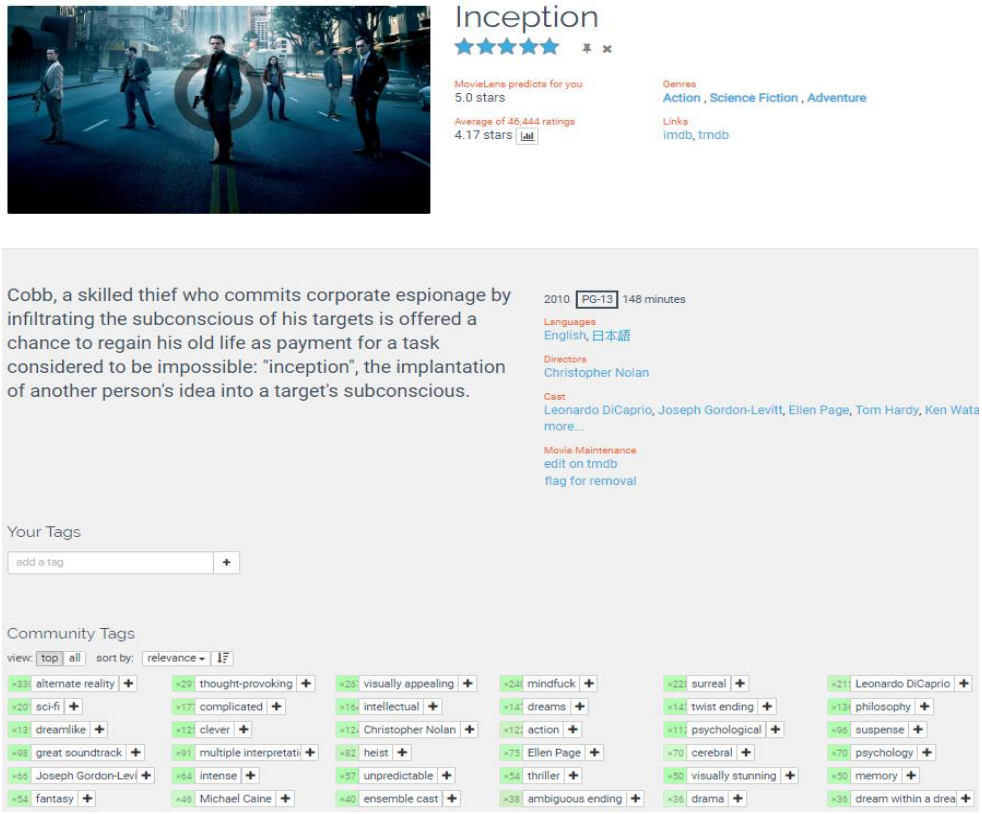

Figura 4.2. Caratteristiche Movielens.org

Il *dataset* oggetto di analisi, *"Movielens"*, è tratto dalle interazioni sopracitate tra la community di utenti ed il sito *www.movielens.org* ed è stato messo a disposizione da *Grouplens* proprio per scopi di ricerca.

Esso contiene nel dettaglio:

- 100.836 rating
- 9.742 film
- 610 utenti
- 3.683 tag

Questi dati sono stati raccolti tra il 29 Marzo del 1996 e il 24 Settembre del 2018, ed il *dataset* è stato generato il 26 Settembre del 2018.

Gli utenti sono stati selezionati in modo casuale tra quelli che avevano valutato almeno 20 titoli. Sono state inoltre rimosse tutte le informazioni demografiche, infatti, gli utenti non sono rappresentati da nessun'altra informazione eccetto un ID identificativo. Esso è scaricabile dal seguente sito: *http://grouplens.org/datasets/*

e contiene i seguenti file in formato *.cvs:*

- **rating.csv**: file contenente tutti i rating; ogni riga rappresenta un rating di un utente per uno specifico film ed ha la seguente struttura:

  userId, movieId, rating, timestamp

  Il file è ordinato prima per userID, poi per movieID. I rating sono effettuati su scala da 1 a 5 con passo 0.5. Timestamp rappresenta il momento temporale esatto in cui l'utente ha espresso una valutazione.

- **movies.cvs:** file contenente le informazioni riguardo i film; dunque, ogni riga rappresenta un film, il titolo e il corrispondente genere ed ha la seguente struttura:

  movieId,title,genres

  I titoli dei film sono stati importati dal sito *https://www.themoviedb.org/* ed includono l'anno di *release* fra parentesi. I generi sono stati selezionati tra i seguenti: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western, no genres listed.

- **tags.csv:** file contenente tutti i *tag*. Ogni riga rappresenta un *tag* assegnato da un utente per un determinato film e ha la seguente struttura:

  userID, movieID, tag, timestamp

  Il file è ordinato prima per userID, poi per movieID. I *tag* sono tipicamente rappresentati da una singola parola o una breve frase, e sono stati generati dagli utenti per descrivere in modo sintetico il contenuto dei film.

  In tutti i *file*, le variabili userID e movieID rappresentano un codice identificativo dell'utente o del film e sono coerenti tra di loro. È quindi possibile unire le informazioni dei diversi *dataset* usandoli come chiave comune.

4.3 Metodologia

Nei paragrafi successivi sarà illustrata la metodologia seguita per analizzare *Movielens.* Si partirà da un'ispezione dei 3 *data frame rating, movies, tags* e, dove necessario, si procederà ad una eventuale pulizia o manipolazione dei dati. Successivamente, sfruttando l'unione dei *data frame* sarà effettuata un'analisi esplorativa dei dati finalizzata ad individuare i principali trend e *insight* riguardo il settore cinematografico sulla base dei dati a disposizione. A questo punto, sarà implementata una fase di *pre-processin*g dei dati per predispone questi ultimi allo sviluppo dei sistemi di raccomandazione. Obiettivo finale sarà la creazione di una lista di film suggeriti per ciascun utente. Infine, dopo aver generato i modelli ed effettuato le *recommendation list*, ne sarà valutata l'accuratezza e saranno discussi i risultati ottenuti.

4.3.1 Ispezione e pulizia/manipolazione dati

Una volta caricati correttamente i 3 *data frame* sopracitati in R studio, si è andata a verificarne la struttura, a manipolarne il formato ed il nome di alcune variabili per renderle funzionali all'analisi. È stato necessario modificare il formato delle variabili *ID_utente* e *ID_film*, da numerico a fattoriale e della variabile *genere* da stringa di testo a fattoriale. Inoltre, poiché la variabile *titolo* all'interno del *data frame "movies"*, conteneva al suo interno oltre che il titolo del film anche l'anno di produzione, è stato creato un quarto data frame *"movies_1"*. Esso è ottenuto dalla manipolazione della suddetta variabile, grazia all'ausilio dei pacchetti *tidyverse* a *dplyr*, in modo da essere sdoppiata in due costrutti, uno soltanto con il titolo e un altro nuovo, la variabile *anno*, con l'informazione della data di rilascio del film. Quest'ultimo sarà approfondito durante l'analisi esplorativa.
Di seguito le strutture finali dei 3 *data frame* principali oggetto di analisi:

1) Data frame ***rating***: 100.836 osservazioni per 3 variabili:
    - ***ID_utente***: variabile *factor* con 610 livelli, corrispondenti ai diversi utenti
    - ***ID_film***: variabile *factor* con 9.724 livelli, corrispondente alla totalità dei film

- ***rating*:** variabile continua che va da un valore minimo di 0.5 ad un massimo di 5, rappresentante la valutazione di ciascun utente verso un determinato film

Di seguito la visualizzazione delle prime 10 osservazioni:

	ID_utente	ID_film	rating
1	1	1	4
2	1	3	4
3	1	6	4
4	1	47	5
5	1	50	5
6	1	70	3
7	1	101	5
8	1	110	4
9	1	151	5
10	1	157	5

Figura 4.3. Data Frame rating

2) Data frame ***movies***: 9.724 osservazioni per 3 variabili:
   - ***ID_film***: variabile *factor* con 9.724 livelli, corrispondente alla totalità dei film e coerente con il data frame *rating*
   - ***titolo***: variabile *character* contenente per ogni film il titolo e l'anno di rilascio
   - ***genere***: variabile *factor* con 951 livelli rappresentanti i generi di ciscun film; ogni titolo può appartenere ad una combinazione differente di *n* generi

Di seguito la visualizzazione delle prime 10 osservazioni:

	ID_film	titolo	genere
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	2	Jumanji (1995)	Adventure|Children|Fantasy
3	3	Grumpier Old Men (1995)	Comedy|Romance
4	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	5	Father of the Bride Part II (1995)	Comedy
6	6	Heat (1995)	Action|Crime|Thriller
7	7	Sabrina (1995)	Comedy|Romance
8	8	Tom and Huck (1995)	Adventure|Children
9	9	Sudden Death (1995)	Action
10	10	GoldenEye (1995)	Action|Adventure|Thriller

Figura 4.4. Data Frame movies

3)Data frame ***tags***: 3.683 osservazioni per 3 variabili

- ***ID_utente***: variabile *factor* con 58 livelli, corrispondenti agli utenti che hanno assegnato una descrizione ai film e coerente con il data frame *rating*
- ***ID_film***: variabile *factor* con 1.572 livelli, corrispondenti ai diversi film che hanno avuto almeno una descrizione
- ***tag***: variabile *character* contenente le descrizioni degli utenti per un determinato film

Di seguito la visualizzazione delle prime 10 osservazioni:

	ID_utente	ID_film	tag
1	2	60756	funny
2	2	60756	Highly quotable
3	2	60756	will ferrell
4	2	89774	Boxing story
5	2	89774	MMA
6	2	89774	Tom Hardy
7	2	106782	drugs
8	2	106782	Leonardo DiCaprio
9	2	106782	Martin Scorsese
10	7	48516	way too long

Figura 4.5. Data Frame tags

4.3.2 Analisi esplorativa

Questo paragrafo sarà dedicato ad una analisi esplorativa del *dataset* con lo scopo di fornire *insight* rilevanti riguardo il settore cinematografico sulla base delle informazioni già presenti nei *data frame* ma soprattutto grazie alla combinazione di essi, utilizzando le variabili in comune come chiave di unione.
Si risponderà alle seguenti domande di ricerca:

- Qual è il comportamento di rating degli utenti
- Quali risultano essere i titoli più popolari
- Quali titoli presentano il rating medio più alto
- Qual è la distribuzione dei film negli anni
- Quali sono i generi di film predominanti
- Quali *tag* forniti dagli utenti descrivono meglio ciascun genere di film

Comportamento di rating

L'obiettivo di questa sezione è capire il comportamento di rating dei 610 utenti coinvolti. Nella totalità sono stati effettuate 100.836 valutazioni; il punteggio minimo assegnato è stato 0,5, il punteggio massimo corrisponde a 5 mentre il rating medio è di 3,5. Considerando le valutazioni da 0,5 a 2,5 come negative, quelle di 3 come neutrali e da 3,5 a 5 come positive, è possibile fare un bilancio rispetto alla polarità dei rating. Come si evince dalla Figura 4.6, complessivamente le buone valutazioni risultano maggiori, infatti, il 19 % sono negative, il 20% neutre e il 61% risultano positive.

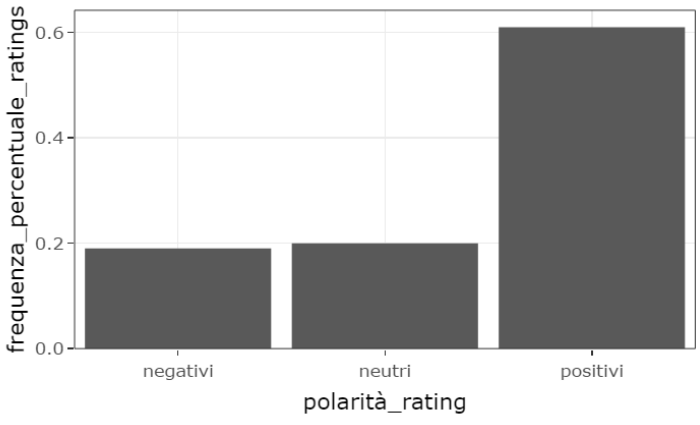

Figura 4.6. Polarità rating

Creando un nuovo *data frame* soltanto con i rating e trasformandoli da variabile continua a fattoriale è possibile calcolare le frequenze assolute e percentuali dei punteggi, con lo scopo di comprendere nello specifico quale siano state le valutazioni più frequenti che gli utenti hanno effettuato. Dal grafico si evince immediatamente che il punteggio più frequente risulta essere 4, che è stato utilizzato 26.818 volte ed occupa il 27% delle valutazioni.

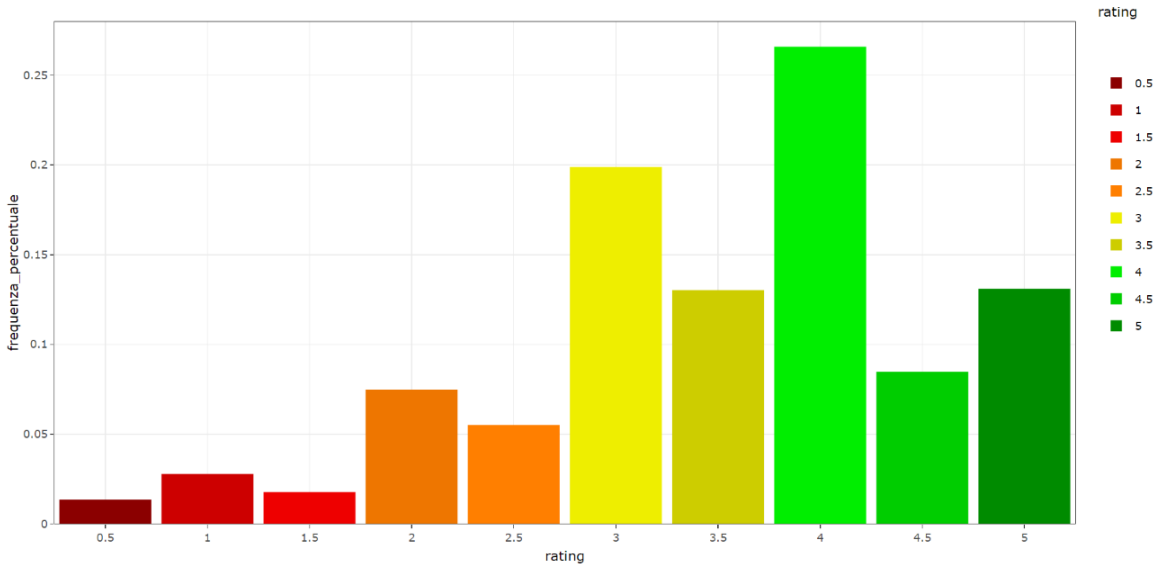

Figura 4.7. Distribuzione rating

Titoli più popolari e più apprezzati

Scendendo ancor di più nel dettaglio, risulta interessante scoprire, sotto forma di ranking, tra la totalità dei film disponibili quali sono stati i titoli più popolari, quindi quelli che hanno avuto più valutazioni da parte degli utenti e quali invece sono stati i titoli che hanno riscosso maggiore successo, ovvero quelli con il rating medio più elevato. Per effettuare questa operazione è stato necessario prima di tutto combinare i due *data frame rating* e *movies*, utilizzando come chiave comune la variabile *ID_film.* A questo punto per ogni ID_film, avremo il corrispondente titolo del film e il conseguente rating assegnatogli.

Per fornire il ranking di popolarità, *in primis* è stato creato un nuovo *data frame* con il calcolo del numero assoluto di valutazioni per ciascun titolo, in secondo luogo è stato ordinato in modo decrescente, estraendo da esso le prime 10 osservazioni che rappresentano la TOP 10 dei film più popolari all'interno del nostro *dataset*.

titolo	frequenza
Forrest Gump (1994)	329
Shawshank Redemption, The (1994)	317
Pulp Fiction (1994)	307
Silence of the Lambs, The (1991)	279
Matrix, The (1999)	278
Star Wars: Episode IV - A New Hope (1977)	251
Jurassic Park (1993)	238
Braveheart (1995)	237
Terminator 2: Judgment Day (1991)	224
Schindler's List (1993)	220

Figura 4.8. Top 10 titoli più popolari

Per fornire il ranking dei titoli che hanno riscosso un maggiore successo è stato attuato un procedimento simile, con la differenza che nel nuovo *data frame* è stata inserita una nuova variabile, *rating_medio*, ottenuta effettuando la media di tutti i rating per ciascun film. Prima di visualizzare la TOP 10, con lo scopo di preservare l'attendibilità delle valutazioni, si è ritenuto opportuno considerare soltanto quei film valutati almeno 50 volte dagli utenti, sfruttando le informazioni ricavate dal ranking di popolarità.
Di seguito i titoli con il rating medio più elevato:

titolo	rating_medio
Shawshank Redemption, The (1994)	4.429022
Godfather, The (1972)	4.289062
Fight Club (1999)	4.272936
Cool Hand Luke (1967)	4.271930
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	4.268041
Rear Window (1954)	4.261905
Godfather: Part II, The (1974)	4.259690
Departed, The (2006)	4.252336
Goodfellas (1990)	4.250000
Casablanca (1942)	4.240000

Figura 4.9. Top 10 titoli più apprezzati

Evoluzione della produzione di titoli negli anni

Dal momento che nei due ranking appena stilati si evidenzia che la maggior parte dei titoli presenti siano stati rilasciati verso la fine degli anni ‘90, risulta interessante comprendere quale sia l’evoluzione dei film negli anni all’interno del *dataset* oggetto di analisi. Per arrivare a rappresentarne la distribuzione si è partiti dalla manipolazione della variabile *titolo* presente nel data frame *movies.* Avendo come contenuto il titolo di ciascun film accompagnato dall’anno di *release* tra parentesi (es: Forrest Gump, (1994)) ed appartenendo ad un formato stringa di testo, è stato possibile scomporre le due informazioni in due variabili differenti: la prima soltanto con il titolo e la seconda con l’anno di produzione. A questo punto, dopo aver generato un nuovo *data frame* con la frequenza assoluta dei film associata a ciascun anno di produzione, è stato possibile visualizzarne l’evoluzione con l’ausilio di un grafico a linee.

Dal grafico (Figura 4.10) risulta evidente che dal 1975 c’è stato un grande incremento di film prodotti e che la maggior parte dei titoli all’interno del *dataset* siano stati rilasciati tra il 1980 e i primi anni del 2000. Ciò, va a giustificare anche i risultati dei ranking stilati in precedenza.

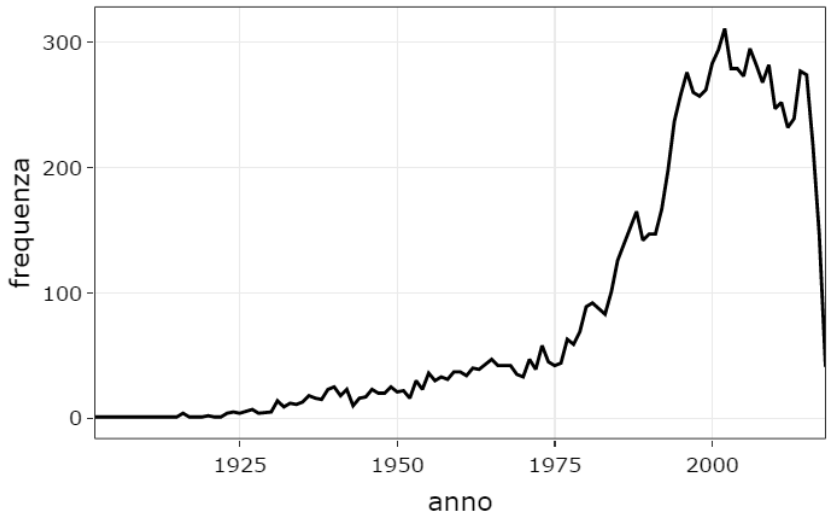

Figura 4.10. Evoluzione titoli negli anni

Generi più popolari

Altrettanto rilevante è la comprensione di quali siano i generi più popolari ai quali appartengono i titoli all’interno del *dataset*. È stato possibile ricavare questa informazione partendo dalla variabile *genere* del *data frame movies.* Tuttavia, il contenuto originario della variabile non era funzionale all’obiettivo di analisi, in

quanto per ciascun titolo non vi era assegnato un solo genere, piuttosto una combinazione di essi nel caso in cui il film fosse considerato un misto di generi differenti (es: Adventure | Children | Fantasy), dando vita a 951 combinazioni diverse. È stato necessario quindi manipolare suddetta variabile, separando i generi combinati e contando singolarmente, ed in modo distinto, la frequenza con cui ogni genere era rappresentativo di un film. Di conseguenza, se un film fosse appartenuto a tre tipologie di generi differenti, gli sarebbe stata associata la frequenza di uno a tutti e tre i generi. Dopo aver creato quindi un nuovo *data frame* con i generi e le rispettive frequenze assolute, è stato stilato un ranking in ordine decrescente dei TOP 10 generi più popolari. (Figura 4.11)

Genere	Frequenza
Drama	4361
Comedy	3756
Thriller	1894
Action	1828
Romance	1596
Adventure	1263
Crime	1199
Sci-Fi	980
Horror	978
Fantasy	779

Figura 4.11. Top 10 generi più popolari

Evoluzione generi negli anni

Questa sezione è dedicata alla verifica dell'esistenza di un legame tra la tipologia di generi al quale appartiene un film e l'anno di produzione di quest'ultimo. Utilizzando il *data frame movies* opportunamente manipolato è stato possibile visualizzare la distribuzione di ciascun genere negli anni. Il grafico sotto stante risulta filtrato soltanto per 3 generi, per una visualizzazione più chiara; sono stati scelti quelli più popolari, cioè *Drama*, *Comedy* e *Thriller*. I risultati tuttavia sono molto coerenti con quelli delle analisi precedenti (Figura 4.12), infatti essendo stati prodotti in misura maggiore tra la fine degli anni '90 e l'inizio del 2000, sono principalmente loro i responsabili dell'incremento generale della produzione di titoli all'interno del *dataset*. Conseguenza diretta è quindi anche la loro maggiore popolarità rispetto agli altri generi.

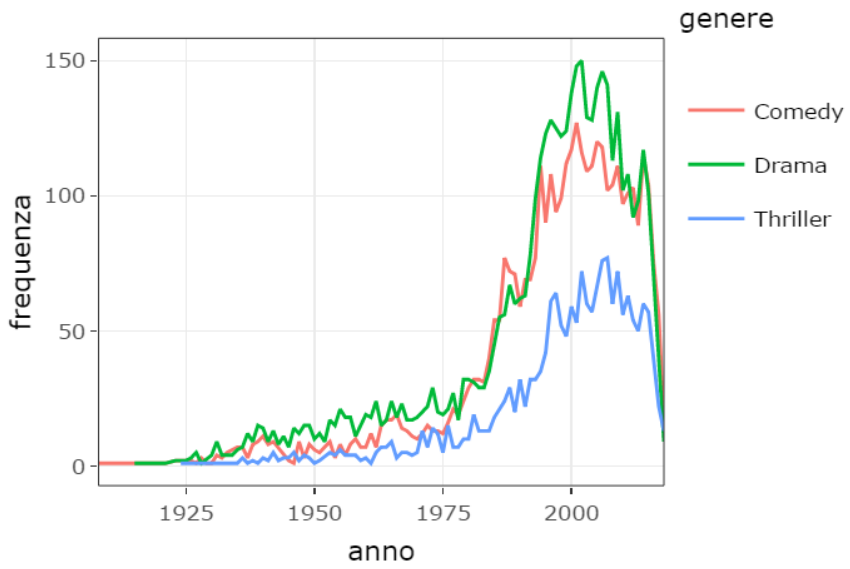

Figura 4.12. Evoluzione generi negli anni

Tag che descrivono meglio i generi di film

Dopo aver fornito il ranking dei generi più popolari, risulta interessante comprendere quali siano le parole (*tag*) che gli utenti attribuiscono con maggiore frequenza a ciascun genere di film, sintetizzandone la descrizione. Il punto di partenza per questa informazione risiede nel contenuto della variabile *tag* all'interno del *data frame tags.* Tuttavia, essa è associata all' ID dell'utente che l'ha assegnata ed all'ID del film a cui è riferita. Per ricollegarla a ciascun genere è stato necessario, in primo luogo considerare i generi singolarmente e non in formato combinato, in modo simile a quanto fatto nell'analisi precedente, ed in secondo luogo unire i *data frame tag* e *movies* per mezzo della variabile chiave *ID_film.* Il risultato finale è un nuovo *data frame* che per ciascun genere, associa un elenco di tutte le parole assegnate dagli utenti per descrivere i film appartenenti ad esso.

Prima di calcolare le frequenze delle parole, indice di importanza nel descrivere l'argomento dei generi, ed evidenziarne le più utilizzate, si è cercato di standardizzare il formato delle parole, con l'ausilio del pacchetto *tm.* Infine, sono stati stilati i ranking dei *tag* più rappresentativi per ciascun genere.

Dall'interpretazione esemplificativa del seguente grafico (Figura 4.13), è possibile intuire che i film *Adventure* trattino spesso argomenti di super eroi, viaggi nello spazio e nel tempo e siano prodotti in maggior numero dalla Disney con un'ottima grafica.

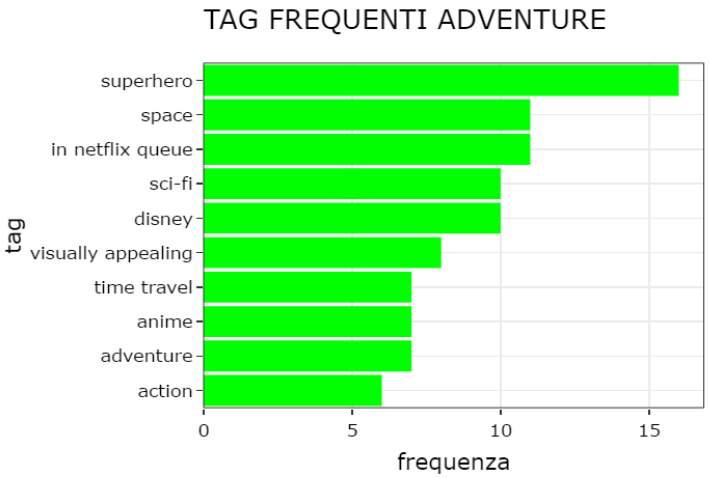

Figura 4.13. Tags più frequenti genere *Adventure*

I film *Drama* invece (Figura 4.14), sono contraddistinti da un contesto atmosferico/surreale ricco di suspance, trattano argomenti religiosi o a sfondo psicologico, presentano colpi di scena e spesso provocano riflessioni nello spettatore

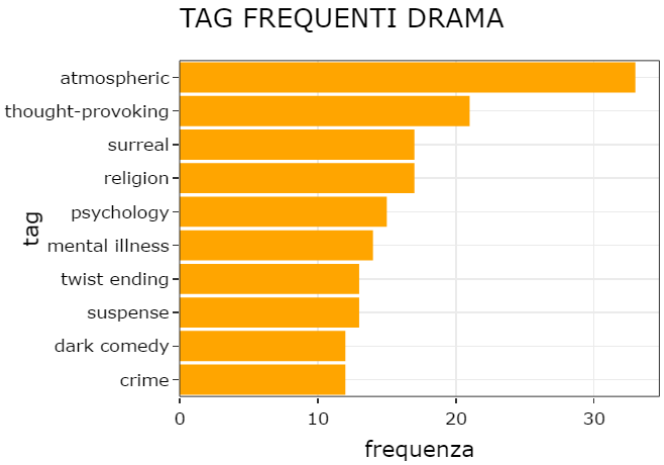

Figura 4.14. Tags più frequenti genere *Drama*

I titoli *Crime* (Figura 4.15) risultano avere qualche affinità con i *Drama*, come i colpi di scena, le riflessioni o la natura psicologia, ma tra gli argomenti più gettonati abbiamo quelli di mafia, commedia dark, droga e polizia.

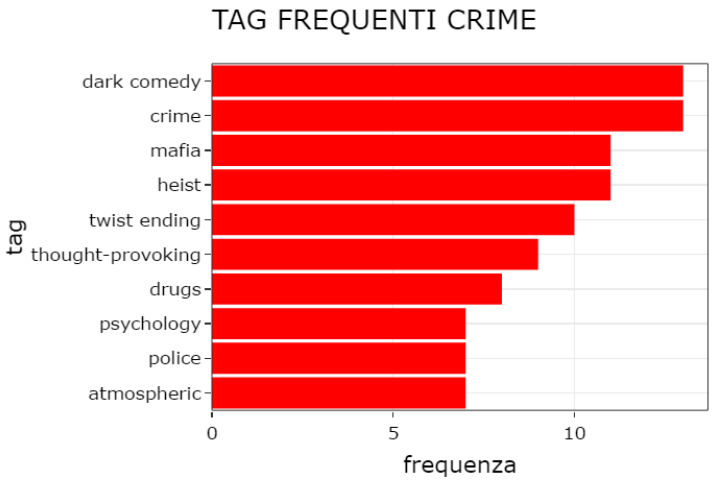

Figura 4.15. Tag più frequenti genere *Crime*

4.4 Implementazione dei sistemi di raccomandazione su Movielens

Questa sezione sarà dedicata all'implementazione di sistemi di raccomandazione applicati al *dataset* oggetto di analisi "*Movielens*". L'analisi sarà effettuata con l'ausilio del software statistico R studio e nello specifico del pacchetto *recomenderlab*. Obiettivo finale sarà avere una lista di film/serie tv da raccomandare ad ogni utente sulla base dei loro comportamenti passati.

Nello specifico saranno implementati racommandation system appartenenti alla categoria degli approcci collaborativi. Verranno messi a confronto i due principali algoritmi collaborativi che utilizzano come input essenziale la matrice *user-item-rating* e si basano sull'assunzione che le preferenze degli utenti siano coerenti e stabili nel tempo:

- ***User-Based collaborative filtering***
- ***Item-Based collaborative filtering***

Entrambi pongono le loro fondamenta sulla misura della similarità tra due o più elementi, tuttavia, mentre l'algoritmo *User-Based* prende in considerazione la similarità dei comportamenti di rating tra gli utenti, l'*Item-Based* pone l'attenzione sugli item, che in questo caso sono film, e la similarità nelle loro valutazioni.

Essi dipendono dalla scelta di 3 parametri principali, che ne condizionano il funzionamento e di conseguenza i risultati finali:

1) ***K nearest-Neighbor*:** Il numero di utenti da tenere in considerazione nel calcolo delle similarità tra utenti o item. Un numero di *neighbor* troppo piccolo comporta ad una riduzione di copertura e quindi una bassa capacità di consigliare un numero consistente di item. Una eccessiva grandezza del *neighborhood*, invece, aumenta il rischio di aggiungere *"noise"* alle predizioni.
2) **Metodologia di normalizzazione dei dati:** Dal momento che i dati derivano da interazioni reali degli utenti con il sito *www.movielens.com*, bisogna tenere conto degli eventuali *bias* a cui essi vanno incontro nell'espressione della loro valutazione riguardo gli item. È possibile infatti che qualche utente attribuisca tendenzialmente rating estremi o al contrario dia voti sempre intermedi. Questo comportamento rischia di pesare negativamente sulla qualità delle predizioni ed è dunque necessario normalizzare i dati. Saranno messi a confronto due tra i metodi più utilizzati: lo *Zero-mean centering* sottrae ai *vector* rating di ciascun utente la loro media, rendendola uguale a zero; lo *Z-score* invece, divide ciascun rating degli utenti per la sua deviazione standard.
3) **La misura di similarità:** tra le differenti metodologie di misurazione della similarità fra utenti o item ne saranno confrontate tre. L'indice di correlazione di Pearson, la *Cosine similarity* e l'indice di Jaccard.

Con lo scopo di trovare il *mix* di parametri che garantisca il margine di errore minimo nella predizione finale, per ciascuno dei due approcci saranno generati diversi sistemi di raccomandazione risultanti dalla combinazione dei suddetti parametri. Una volta implementati i due recommandation system ottimali, saranno poi messi a confronto e ne verranno discussi le peculiarità e i limiti di ciascuno.

4.4.1 Preprocessing: data frame in realRatingMatrix

La prima operazione da effettuare per rendere il *data frame rating* funzionale all'analisi è la trasformazione di quest'ultimo in un formato matrice, le cui righe rappresentano gli utenti, le colonne i film e infine le celle corrispondono alla valutazione che ogni utente ha assegnato a ciascun film. Inoltre, per poter processare tale matrice con il pacchetto *recommenderlab* ai fini dell'implementazione dei sistemi di raccomandazione è necessario renderla un oggetto *"realRatingMatrix"*.

Di seguito (Figura 4.16) la visualizzazione della matrice le cui dimensioni corrispondono a 610 righe date dal numero degli utenti, 9724 colonne, date dal numero di film, e 5.931.640 celle derivanti dall'incrocio di ciascun utente con la totalità dei film e rappresentate i rating.

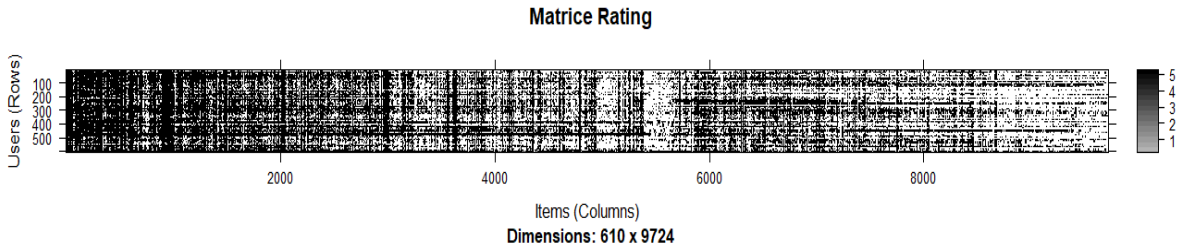

Figura 4.16. Matrice dei rating

Visto il numero elevato di celle, per una migliore visualizzazione della matrice nonché per una maggiore comprensione del suo contenuto, di seguito viene fornita una prospettiva differente di essa, circoscritta alla rappresentazione dei primi 30 utenti e 30 film. Dal grafico sottostante (Figura 4.17) si evince un elevato livello di sparsità della matrice dato da un numero elevato di celle vuote; in corrispondenza di esse l'utente in questione non ha effettuato alcuna valutazione per il corrispondente film. Il numero di rating infatti, 100.836, risulta nettamente inferiore al numero di celle presenti nella matrice.

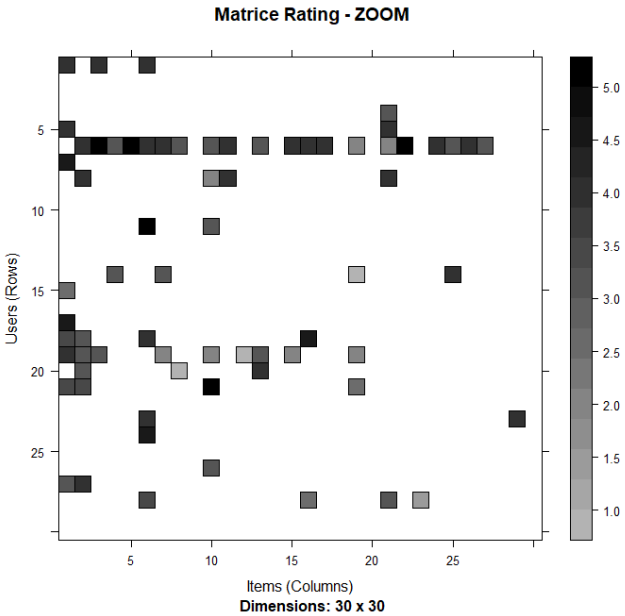

Figura 4.17. Matrice dei rating zoomata

Prima di implementare i sistemi di raccomandazione, al fine di massimizzarne l'attendibilità e l'accuratezza è necessario escludere, pur essendo presenti nella matrice, sia quegli utenti che hanno votato un numero molto basso di film, sia quei film che sono stati valutati raramente. Questa tipologia di utenti e item infatti, non solo risulta poco rilevante per lo scopo dell'analisi, ma addirittura rischia di inficiarne i risultati. Ispezionando la matrice oggetto di analisi, si nota che, sono presenti tutti quegli utenti che hanno espresso almeno 20 rating; per quanto riguarda gli item invece, sono presenti anche film per cui è stato effettuato una sola valutazione. Per quest'ultima ragione l'analisi verrà effettuata su una matrice ridotta, infatti, verranno presi in considerazione soltanto i film per cui sono state espresse almeno 25 valutazioni. Di conseguenza la nuova matrice oggetto di analisi sarà composta da 498 utenti, 860 film e 55360 rating.
Prima dell'implementazione dei sistemi di raccomandazione, di seguito saranno descritti tutti gli strumenti e le funzioni necessarie allo sviluppo e la valutazione dei modelli.

4.4.2 Impostazione evaluationSchema: Train e Test

La creazione di uno schema di valutazione è l'ultimo passaggio funzionale all'impostazione dei sistemi di raccomandazione e allo stesso tempo ne permette anche la successiva comparazione e valutazione. La funzione ***evaluationScheme***, presente nel pacchetto *recomenderlab*, richiede il settaggio dei seguenti parametri fondamentali:

- **La matrice dei dati**: input essenziale per il calcolo delle similitudini tra utenti/item e per la predizione dei rating inespressi.
  In questo caso è stata selezionata la matrice sopracitata decurtata dagli utenti e item poco rilevanti
- **Il metodo di valutazione:** dopo aver effettuato i calcoli necessari, sovviene l'esigenza di verificare che il modello di raccomandazione sia valido non solo per il set di dati utilizzato per l'implementazione ma anche per ulteriori dati, nuovi, che potrebbero aggiungersi nel tempo, rendendolo quindi generalizzabile. In questo caso, il metodo utilizzato è lo *split*, ovvero la divisione casuale del *dataset* in due entità: il *train* utilizzato per l'allenamento

degli algoritmi, e su cui saranno implementati i diversi modelli; il *test*, utilizzato per la valutazione delle raccomandazioni.

- **Percentuale di divisione del dataset in *train* e *test*:** è necessario impostare il la proporzione che andrà a comporre i due set di dati. Nell'analisi seguente, l'80% dei dati saranno dedicati all'implementazione dei modelli (*train*) ed il restante 20% alla valutazione dei risultati (*test*).
- **Given:** in questo campo bisogna inserire il numero di item che, scelti in modo casuale tra quelli appartenenti al *test*, saranno utilizzati per generare le raccomandazioni; i restanti verranno invece utilizzati per valutarne l'accuratezza. Per garantire una valutazione attendibile, il parametro deve essere necessariamente minore o uguale del numero minimo di rating espressi da un utente all'interno della matrice. Nell'analisi esso è stato impostato uguale a 9
- **GoodRating:** ultimo passaggio è il settaggio di un valore soglia che definisca il punteggio minimo affinché una valutazione sia considerata positiva. In questo caso è stato impostato uguale a 4. Di conseguenza tutti i rating che sono maggiori o uguali a 4 verranno considerati positivi.

4.4.3 Implementazione recommender systems

Per l'implementazione dei modelli è stata utilizzata la funzione ***Recommender*** appartenente al pacchetto *recommenderlab*, con la necessità di specificare 3 elementi:

- **GetData:** in questo campo bisogna inserire lo schema di valutazione di riferimento precedentemente creato e la base dati utilizzata per l'implementazione del modello, ovvero il *train*
- **Method:** in cui deve essere specificata la tipologia di algoritmo da impiegare; ai fini dell'analisi saranno utilizzati i due approcci collaborativi più celebri in letteratura, lo *user-based collaborative filtering* (UBCF) e l'*item-based collaborative filtering* (IBCF)
- **Parameter:** include in formato lista l'insieme dei parametri che condizioneranno il funzionamento e i risultati dell'algoritmo. In particolare, bisogna specificare il numero di *K-nearest neighbor*, la metodologia di normalizzazione dei dati, il metodo di calcolo delle similarità tra utenti/item.

4.4.4 Ottenere le predizioni

Passo successivo alla creazione del modello è l'utilizzo della funzione ***predict.***
Essa rende applicabile il sistema di raccomandazione appena implementato su un nuovo set di dati, fornendo due possibili tipologie di output a seconda delle esigenze: una nuova matrice con la totalità dei *predicted rating* oppure una lista di *Top N* item raccomandati per ciascun utente. Nello specifico la funzione richiede la compilazione dei seguenti campi:

- **Modello:** il sistema di raccomandazione precedentemente implementato sul *train*
- **getData:** lo schema di valutazione di riferimento, che dovrà essere lo stesso utilizzato per implementare il modello, più il nuovo set di dati su cui effettuare le predizioni appartenente al *test*.
- **type:** come preannunciato, questo parametro può essere completato in modo differente a seconda della tipologia di output desiderato. È possibile inserire *"rating"* per fornire una nuova matrice dei dati con la totalità dei *predicted rating*, oppure *"n"*, seguito da un numero specifico, al fine di stilare una lista di *n* item raccomandati per ciascun utente.

4.4.5 Valutazione sistema di raccomandazione

Dopo aver stilato le predizioni sul nuovo set di dati è necessario valutarne l'accuratezza. Per effettuare questa operazione viene utilizzata la funzione ***calcPredictionAccuracy***. Essa fornisce le tre principali unità di misura di accuratezza dei sistemi di raccomandazione, precedentemente descritte: RMSE, MSE, MAE.
Per il corretto funzionamento è necessario specificare i seguenti parametri:

- ***predicted rating:*** matrice con la totalità dei *predicted rating* generata nel passaggio precedente
- ***GetData:*** lo stesso schema di valutazione utilizzato per la creazione del modello e le predizioni più il set di dati su cui confrontare i rating predetti con quelli attuali.

Nonostante le unità si misura di errore appena descritte siano le più utilizzate in letteratura, esse sono spesso affiancate da un'altra tipologia di metrica, ovvero la matrice di confusione e ROC Curve. Mentre le prime risultano piuttosto quantitative e sono utilizzate per valutare la totalità degli scostamenti tra i rating predetti e quelli attuali, questi ultimi si concentrano sulla qualità della *raccomandation list*. Per il calcolo dei seguenti indicatori viene utilizzata la funzione ***evaluate*** con il dettaglio di tali parametri:

- **Schema di valutazione:** necessariamente uguale a quello utilizzato per l'implementazione
- **Metodo:** descrizione nominativa dell'algoritmo utilizzato
- **Numero raccomandazioni:** la *recommendation list* può avere una ampiezza differente a seconda delle proprie esigenze. Tuttavia, è buona norma inserire una lista con ampiezza varia e crescente, in modo da valutare quale sia il numero di item ottimale che garantisca una lista di raccomandazione più accurata.
- **progress:** campo che non influisce sulla valutazione dei risultati ma che, se impostato, permette di visualizzare la performance dell'algoritmo in termini di velocità, sia per l'implementazione del modello che nello sviluppo delle predizioni.

Una volta calcolato il risultato delle valutazioni, è possibile inserirli nella funzione ***getConfusionMatrix,*** per avere come output la matrice di confusione, oppure è possibile visualizzarli graficamente sotto forma di ROC Curve, con l'impostazione di diversi valori soglia corrispondenti a *reco list* di differenti grandezze.

4.5 Procedura per la selezione del modello ottimale

Gran parte dell'analisi è stata dedicata alla ricerca di quel *mix* di parametri in grado di assicurare una accuratezza maggiore ai risultati delle raccomandazioni, essendo l'obiettivo finale la selezione e la comparazione dei due migliori modelli *user-based* e *item-based*. Per ciascuno dei due approcci è stata utilizzata una procedura che ha permesso il confronto tra tutte le combinazioni di parametri. Mentre la metodologia di normalizzazione (*Z-score* e *Zero-mean centering*) e le misure di similarità (Pearson,

Cosine e Jaccard) sono variabili prestabilite, il numero di utenti/item *Neighbor* risulta essere un numero arbitrario. Partendo dunque dal valore default consigliato dal software, sono stati provati altri 4 valori differenti di *neighbor*. Ne consegue che per ciascuno dei due approcci sono stati messi a confronto 30 modelli, derivanti dall'incrocio delle 2 variabili di normalizzazione, 3 misure di similarità e 5 diversi valori di *neighbor*; tra essi è stato poi selezionato il più accurato.
Per una più chiara comparazione e una migliore visualizzazione della ROC Curve, piuttosto che mettere a confronto i 30 modelli contemporaneamente, la procedura eseguita è stata scomposta in diversi passaggi.
Quest'ultima è stata utilizzata in primo luogo per selezionare il miglior modello *User-based* ed in seguito replicata per l'approccio *Item-based*.
Essa si articola nelle seguenti fasi:

1. Per ogni livello di *neighbor* sono stati implementati 6 modelli (per un totale di 30 modelli) con tutte le combinazioni dei restanti due parametri, con lo scopo di comprendere per ognuno dei cinque livelli *neighbor* quale fosse la misura di similarità e la metodologia di normalizzazione più adatta.
2. Verificando le misure di errore e la ROC Curve, sono stati estratti i migliori modelli per ciascun livello di *neighbor* per un totale di 5 modelli
3. Questi ultimi sono stati messi a confronto per poter selezionare il modello che presenti il mix di parametri ottimale.

4.5.1 UBCF recommendation system: ottimizzazione parametri

Di seguito, per una migliore comprensione, viene fornita parte della procedura al punto 1 e 2 seguita per l'approccio UBCF. In particolare, la metodologia con cui, scelto uno dei cinque livelli di *neighbor*, in questo caso 25, è stato possibile confrontare tutte le combinazioni degli altri due parametri, con l'obiettivo di selezionare il più accurato tra essi.
In primis, è stata creata una lista di elementi "compara_modelli_25" il cui contenuto presenta i 6 modelli corrispondenti al valore di *Neighbor* 25 ma con ogni combinazione delle altre due variabili (Misura di similarità e metodo di normalizzazione)

```
#6 modelli con NN 25 e tutte le combinazioni degli altri due parametri

 compara_modelli_25 <-list(
 "25N_cos_c" = list(name="UBCF", param=list(normalize = "center",
                                            method="Cosine",
                                            nn=25)),
 "25N_p_c" = list(name="UBCF", param=list(normalize = "center",
                                          method="pearson",
                                          nn=25)),
 "25N_j_c" = list(name="UBCF", param=list(normalize = "center",
                                          method="jaccard",
                                          nn=25)),
 "25N_cos_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                            method="Cosine",
                                            nn=25)),
 "25N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                          method="pearson",
                                          nn=25)),
 "25N_j_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                          method="jaccard",
                                          nn=25)))
```

Grazie alla funzione *evaluate* è stato possibile implementare i 6 modelli ed effettuare il calcolo di nuove matrici di rating al cui interno sono state generati *predicted rating* per ciascuna coppia utente-item, salvate nell'oggetto "risult_modelli_25".

```
risult_modelli_25<- evaluate(schema_valutazione, compara_modelli_25,
n = recs, progress = TRUE)
```

Per un confronto veloce ed intuitivo tra i modelli risulta utile la visualizzazione della Curva ROC ottenibile plottando i risultati del modello appena calcolati.

```
plot (x = risult_modelli_25, y = "ROC", annotate = 5,
legend="topleft")
```

Il contenuto del grafico sottostante contiene la ROC Curve dei 6 modelli UBCF con livello di *Neighbor* 25 e la combinazione dei restanti due parametri variabili. Nello specifico per "cos", "p", "j" si intendono rispettivamente le distanze *Cosine,* Pearson e Jaccard. Per distinguere il metodo di normalizzazione, con "c" intendiamo *Zero-mean centering* e per "z" ci riferiamo allo *Z-score*. I livelli soglia che permettono di

tracciare la curva corrispondono al numero di film raccomandabili per ciascuna lista; esso varia da 1, quindi una lista contenente un solo film a 30.

Dal grafico (Figura 4.18) si evince chiaramente che i 2 modelli che utilizzano la distanza di Pearson risultano più accurati degli altri, ma tra essi abbastanza simili.

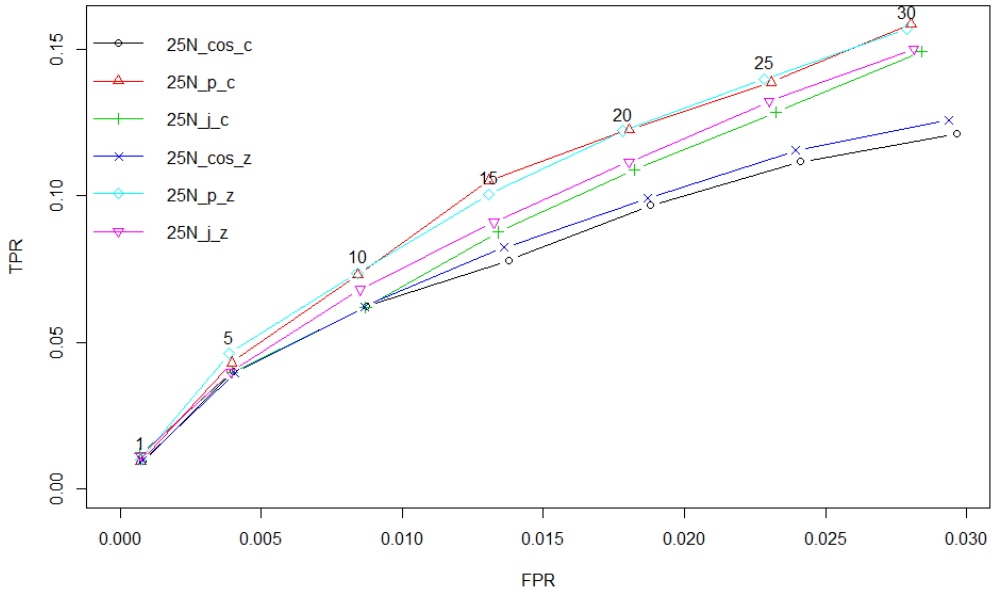

Figura 4.18. ROC Curve UBCF Neighbor 25

Per avere una valutazione quantitativa ancor più oggettiva, risulta necessario il calcolo delle misure di errore. Esse vanno calcolate singolarmente per ciascun modello e richiedono in primo luogo l'utilizzo della funzione *Recommender* per implemetarli, in secondo luogo la funzione *Predict* per effettuare le predizioni ed in terzo luogo la funzione *calcPredictionAccuracy* per verificarne l'accuratezza. Infine, grazie alla funzione *rbind* è possibile visualizzare contemporaneamente tutti i risultati in modo da poterli confrontare.

```
#calcola singolarmente predizione ed errori
UBCF_25N_cos_c<-  Recommender(getData(schema_valutazione,  "train"),
method = "UBCF",
                              param=list(normalize     =     "center",
method="Cosine", nn=25))
pred_UBCF_25N_cos_c<-                          predict(UBCF_25N_cos_c,
getData(schema_valutazione, "known"), type="rating") #predizioni sul
test
```

```
valut_UBCF_25N_cos_c <- calcPredictionAccuracy(pred_UBCF_25N_cos_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_cos_c

UBCF_25N_p_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "center",
method="pearson", nn=25))
pred_UBCF_25N_p_c<- predict(UBCF_25N_p_c,
getData(schema_valutazione, "known"), type="rating") #predizioni sul
test
valut_UBCF_25N_p_c <- calcPredictionAccuracy(pred_UBCF_25N_p_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_p_c

UBCF_25N_j_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "center",
method="jaccard", nn=25))
pred_UBCF_25N_j_c<- predict(UBCF_25N_j_c,
getData(schema_valutazione, "known"), type="rating") #predizioni sul
test
valut_UBCF_25N_j_c <- calcPredictionAccuracy(pred_UBCF_25N_j_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_j_c

UBCF_25N_cos_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                             param=list(normalize = "Z-score",
method="Cosine", nn=25))
pred_UBCF_25N_cos_z<- predict(UBCF_25N_cos_z,
getData(schema_valutazione, "known"), type="rating") #predizioni sul
test
valut_UBCF_25N_cos_z <- calcPredictionAccuracy(pred_UBCF_25N_cos_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_cos_z

UBCF_25N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="pearson", nn=25))
pred_UBCF_25N_p_z<- predict(UBCF_25N_p_z,
getData(schema_valutazione, "known"), type="rating") #predizioni sul
test
valut_UBCF_25N_p_z <- calcPredictionAccuracy(pred_UBCF_25N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
```

```
valut_UBCF_25N_p_z

UBCF_25N_j_z<-   Recommender(getData(schema_valutazione,   "train"),
method = "UBCF",
                            param=list(normalize     =     "Z-score",
method="jaccard", nn=25))
pred_UBCF_25N_j_z<-                                predict(UBCF_25N_j_z,
getData(schema_valutazione, "known"), type="rating") #predizioni sul
test
valut_UBCF_25N_j_z    <-    calcPredictionAccuracy(pred_UBCF_25N_j_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_j_z

#riuniamo tutti gli errori per confrontarli
error_UBFC_25NN<-
rbind(valut_UBCF_25N_cos_c,valut_UBCF_25N_cos_z,valut_UBCF_25N_j_c,v
alut_UBCF_25N_j_z,valut_UBCF_25N_p_c,valut_UBCF_25N_p_z)
error_UBFC_25NN
```

	RMSE	MSE	MAE
valut_UBCF_25N_cos_c	0.9307556	0.8663060	0.7068339
valut_UBCF_25N_cos_z	0.9305769	0.8659734	0.7063564
valut_UBCF_25N_j_c	0.9237834	0.8533758	0.6994586
valut_UBCF_25N_j_z	0.9230885	0.8520923	0.6982376
valut_UBCF_25N_p_c	**0.9172808**	**0.8414041**	**0.6941683**
valut_UBCF_25N_p_z	**0.9153380**	**0.8378437**	**0.6922265**

Figura 4.19. Misure di errore UBCF Neighbor 25

Gli indicatori RMSE, MSE, MAE (Figura 4.19) confermano i risultati ottenuti dalla ROC Curve. Sono infatti i due modelli che utilizzano la distanza di Pearson ad avere un margine di errore inferiore. In particolare, il migliore risulta essere il modello UBCF con livello di *Neighbor* 25, distanza di Pearson e metodo di normalizzazione *Z-score*, che quindi viene selezionato per il confronto finale.

La fase 1 e 2 della procedura appena descritta è stata poi replicata per i restanti 4 valori arbitrari neighbor (45, 65, 85, 100). Il confronto di tutti i modelli per ciascun livello neighbor va a confermare che, a prescindere da quest'ultimo, i sistemi di raccomandazione UBCF implementati utilizzando la distanza di Pearson e il metodo di normalizzazione *Z-score* risultano più accurati e sono stati quindi selezionati per la fase 3 della procedura.

In quest'ultima passaggio, sono stati messi a confronto i 5 modelli precedentemente selezionati aventi le seguenti caratteristiche:

- Valore *Neighbor* variabile: 25, 45, 65, 85, 100
- Misura di similarità: distanza di Pearson
- Metodo di normalizzazione: *Z-score*

È stata quindi creata una lista di elementi "compara_modelli_UB_NN_p_z" al cui interno ritroviamo i 5 modelli aventi misure di similarità e metodo di normalizzazione fissi ma *Neighbor* variabile. Lo scopo infatti è comprendere quale sia il modello migliore al variare dei valori neighbor ma tenendo fissi i restanti due parametri, già ottimizzati.

```
#confrontiamo tutti le combinazioni di Neighbor con distanza di
Pearson e Z-score
compara_modelli_UB_NN_p_z <-list(

  "25N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=25)),
  "45N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=45)),
  "65N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=65)),
  "85N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=85)),
  "100N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                            method="pearson",
                                            nn=100)))
```

Una volta implementati i modelli si procede alla comparazione degli stessi, *in primis* tracciando la ROC Curve e in secondo luogo stimando le misure di errore.
A conferma di quanto riportato in letteratura, dai grafici sottostanti si evidenzia che, a parità dei restanti due parametri, i sistemi di raccomandazione aventi *Neighbor* molto bassi (25) o troppo alti (100) tendono ad avere una prestazione non ottimale se pur per ragioni differenti: nel primo caso gli utenti su cui si sviluppa il modello non sono un

numero sufficiente per elaborare predizioni attendibili, nel secondo caso il numero è eccessivo e si aggiunge "*noise*" ai risultati.

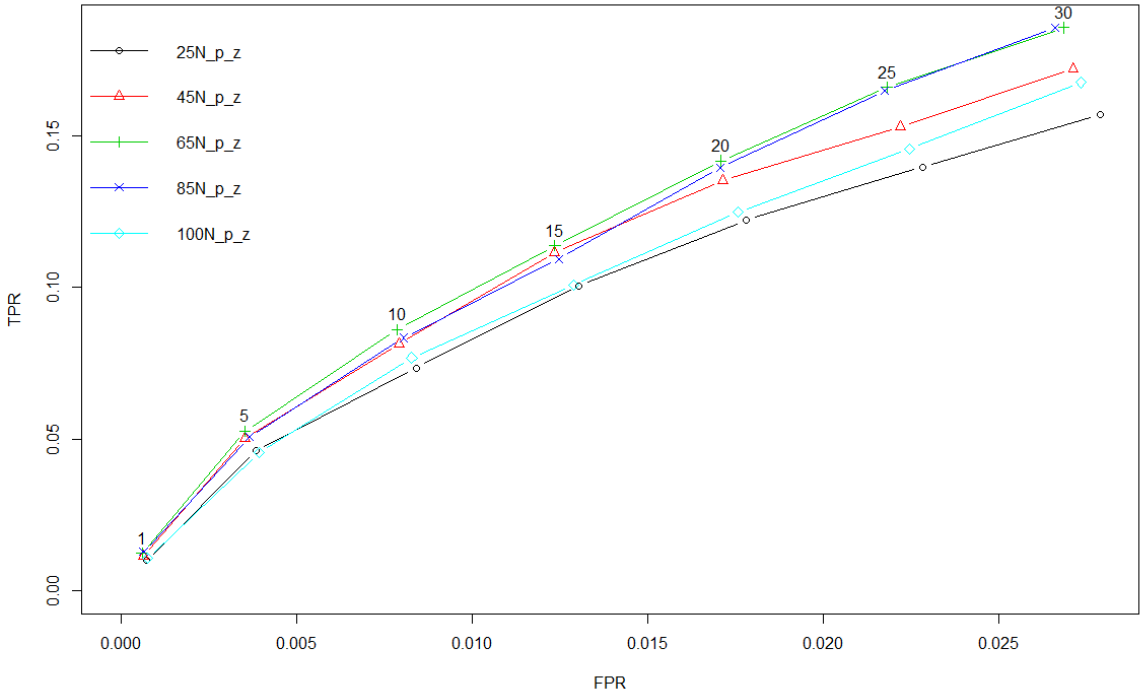

Figura 4.20. ROC Curve UBCF

	RMSE	MSE	MAE
valut_UBCF_25N_p_z	0.9153380	0.8378437	0.6922265
valut_UBCF_45N_p_z	0.9067368	0.8221716	0.6845025
valut_UBCF_65N_p_z	**0.9013977**	**0.8125177**	**0.6800774**
valut_UBCF_85N_p_z	0.9020748	0.8137390	0.6810550
valut_UBCF_100N_p_z	0.9007569	0.8113630	0.6823207

Figura 4.21. Misure di errore UBCF

I due modelli più accurati risultano essere rispettivamente quelli con 65 e 85 *Neighbor*, come si evince dalla ROC (Figura 4.20) e dalle misure di errore (Figura 4.21). Tuttavia, le 3 misure RMSE, MSE, MAE non presentano risultati univoci per i due modelli sopracitati; per questa ragione, al fine di effettuare la scelta tra i due sistemi sopracitati si è scelto di focalizzarsi sulla misura di errore più impegata in letteratura, il MAE. Il modello UBCF più accurato risulta essere dunque quello con *Neighbor* 65, distanza di Pearson e metodo di normalizzazione *Z-score* con un MAE di 0,68007.

4.5.2 IBCF recommendation system: ottimizzazione parametri

Dopo aver selezionato il sistema di raccomandazione UBCF ottimale, è necessario iterare la stessa procedura utilizzata precedentemente per trovare il modello IBCF più accurato. Obiettivo finale sarà, tramite la comparazione dei due approcci la comprensione di quale tra essi sia quello più adatto al *dataset* oggetto di analisi.
Anche in questo caso quindi, sono stati implementati e valutati 30 modelli derivanti dalla combinazione dei 3 parametri: *Neighbor*, metodo di normalizzazione e misura di similarità. Allo stesso modo di quando accaduto per i sistemi *User-based*, anche per IBCF i modelli che utilizzano distanza di Pearson e *Z-score* risultano avere prestazioni migliori, motivo per cui sono stati selezionati per l'ultimo passaggio della procedura.
Nell'identificazione del modello *Item-based* ottimale quindi abbiamo 5 modelli con le seguenti caratteristiche:

- Valore *Neighbor* variabile: 30, 45,65,85,100
- Misura di similarità: distanza di Pearson
- Metodo di normalizzazione: *Z-score*

I modelli sopracitati sono stati inseriti all'interno della lista "compara_modelli_IB_NN_p_z" dopodiché sono stati implementati e successivamente comparati.

```
#confronto tra i diversi NN dei modelli IBCF con pearson e z-score
compara_modelli_IB_NN_p_z <-list(
  "30N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           k=30)),
  "45N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           k=45)),
  "65N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           k=65)),
  "85N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           k=85)),
  "100N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                            method="pearson",
                                            k=100)))
```

Dagli indicatori di errore (Figura 4.22) e dalla visualizzazione della ROC (Figura 4.23) si evince immediatamente che, nonostante l'ottimizzazione dei parametri, l'accuratezza dei sistemi *Item-based* risulta particolarmente bassa. Inoltre, i risultati della Curva ROC non sono coerenti con quelli delle misure di errore, infatti, mentre per quest'ultima il migliore modello risulta essere quello con il livello di *Neighbor* 45, per le misure di errore è il *Neighbor* 100. Scegliamo tuttavia di selezionare il modello IBCF con *Neighbor* 45, misura di similarità Pearson e metodo di normalizzazione *Z-score* come modello IBCF di riferimento per il confronto finale.

	RMSE	MSE	MAE
valut_IBCF_30N_p_z	1.278177	1.633737	0.9045636
valut_IBCF_45N_p_z	1.248636	1.559093	0.9017524
valut_IBCF_65N_p_z	1.215027	1.476291	0.8797132
valut_IBCF_85N_p_z	1.204057	1.449753	0.8809299
valut_IBCF_100N_p_z	**1.179377**	**1.390931**	**0.8638960**

Figura 4.22. Misure di errore IBCF

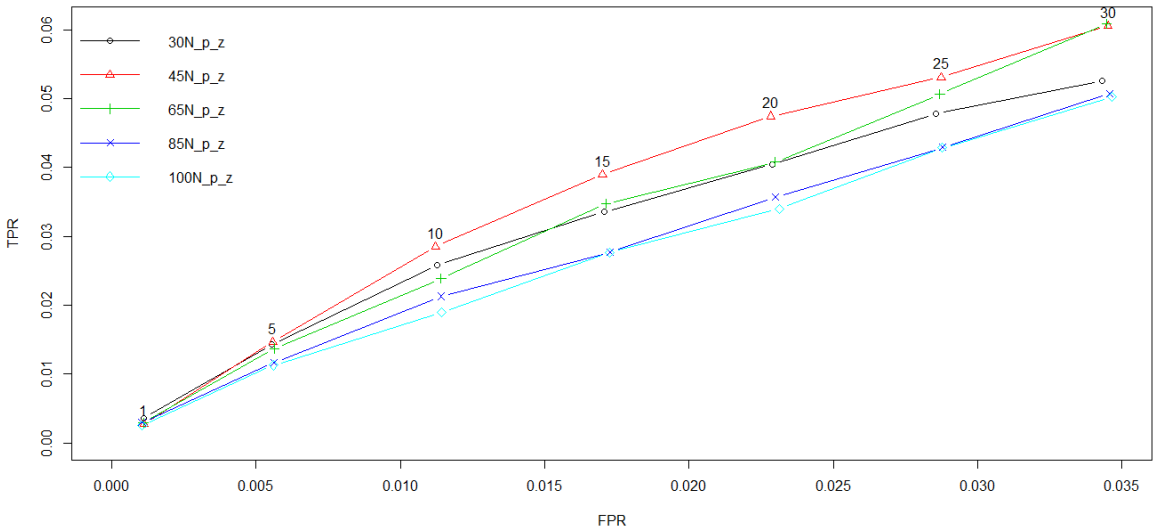

Figura 4.23. ROC Curve IBCF

4.5.3 UBCF e IBCF a confronto

Una volta selezionati i due modelli *User-based* e *Item-based* ottimizzati, questo paragrafo sarà dedicato al confronto tra essi e la conseguente scelta dell'approccio più adatto per la base dati oggetto di analisi. Oltre ai due modelli sopracitati, sarà inserito anche un ulteriore modello *benchmark*, ovvero un sistema di raccomandazione *"Random"*. Esso andrà a simulare un recommendation system completamente casuale, che raccomanda gli item in modo *random* senza l'implementazione di alcun algoritmo. In sintesi, avremo il confronto tra i seguenti modelli:

- **UBCF** con *Neighbor* 65, distanza di Pearson e metodo di normalizzazione *Z-score*
- **IBCF** con *Neighbor* 45, distanza di Pearson e metodo di normalizzazione *Z-score*
- Recommendation system "***Random***", utile come *benchmark*, che andrà a simulare raccomandazioni casuali in assenza di alcun algoritmo implementato

Primo passaggio è stato l'aggregazione dei 3 modelli all'interno di una lista nominata "compara_modelli_finale"

```
compara_modelli_finale <-list(

  "UBCF 65N_p_z" = list(name="UBCF", param=list(normalize = "Z-
score",
                                                method="pearson",
                                                nn=30)),
  "IBCF 45N_p_z" = list(name="IBCF", param=list(normalize = "Z-
score",
                                                method="pearson",
                                                k=45)),
  "RANDOM" = list(name="RANDOM"))
```

Ai fini dello sviluppo dei modelli ed il calcolo dei predicted rating è stata nuovamente utilizzata la funzione *evaluate*:

```
risult_modelli_finale<-evaluate(schema_valutazione,
compara_modelli_finale, n = recs, progress = TRUE)
```

Una prima importante osservazione riguarda la differente efficienza dei modelli in termini di velocità (Figura 4.24). Il sistema di raccomandazione *User-Based* ha impiegato 2 centesimi di secondo per l'implementazione del modello ed un quarto di secondo per effettuare le predizioni. Al contrario il sistema *Item-based* ha impiegato 2,73 secondi per la prima operazione, risultando 137 volte più lento e 0,04 secondi per stimare le predizioni. Il sistema *Random*, non avendo alcun modello sottostante ha impiegato semplicemente 0,06 secondi per calcolare i *predicted rating*.

```
UBCF run fold/sample [model time/prediction time]
                1  [0.02sec/0.25sec]

IBCF run fold/sample [model time/prediction time]
                1  [2.73sec/0.04sec]

RANDOM run fold/sample [model time/prediction time]
                1  [0sec/0.06sec]
```

Figura 4.24. Velocità Recommendation system implementati

Plottando i risultati dei modelli sulla Curva ROC, è possibile comprendere la differente efficacia dei modelli in termini qualitativi, sfruttando come soglie arbitrarie differenti *recommendation list* che, come nelle fasi precedenti, si distinguono per la quantità di item da raccomandare. Dal grafico sottostante (Figura 4.25), il modello *User-Based* risulta avere una prestazione nettamente superiore rispetto a quello basato sugli Item che si presenta particolarmente deludente. Quest'ultimo infatti, oltre a fornire raccomandazioni inaccurate è praticamente assimilabile al sistema *Random* che è stato sviluppato senza l'ausilio di alcun algoritmo.

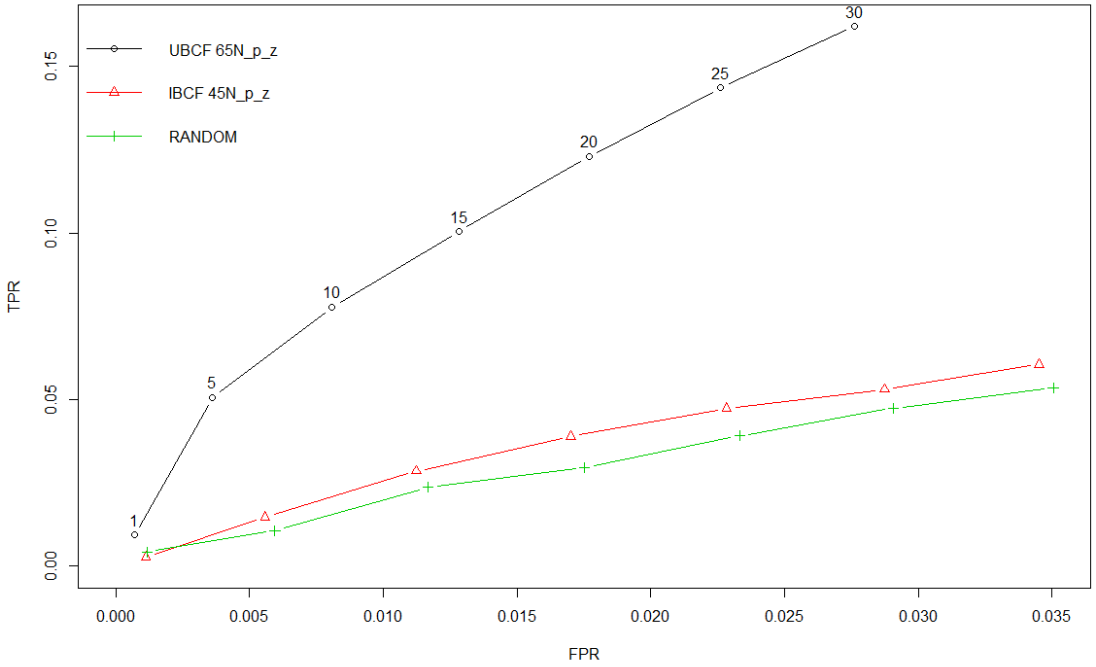

Figura 4.25. ROC Curve UBCF vs IBCF vs Random

Gli indicatori di errore (Figura 4.26) confermano in termini quantitativi quanto visto graficamente dalla ROC. Il sistema di raccomandazione *Item-Based* presenta un errore medio sulle predizioni molto alto, avvicinandosi addirittura ai risultati del modello casuale, motivo per cui non si dimostra funzionale ai fini dell'analisi.

La prestazione del modello *User Based* con *Neighbor* 65, distanza di Pearson e metodo di normalizzazione *Z-score* risulta essere senza dubbio la migliore tra tutti i modelli sperimentati (30 *item-based* e 30 *user-based*). Il margine di errore infatti è stato ridotto notevolmente fino a raggiungere un **MAE = 0,68.**

La scelta finale del modello da utilizzare al fine di raccomandare una lista di film per ogni utente presente nel *dataset* oggetto di analisi ricade dunque su quest'ultimo che sarà quindi utilizzato come riferimento per l'estrazione e la visualizzazione delle *recommendation list*.

	RMSE	MSE	MAE
valut_RANDOM	1.2565642	1.5789536	0.9632421
valut_UBCF_65N_p_z	**0.9013977**	**0.8125177**	**0.6800774**
valut_IBCF_45N_p_z	1.2486364	1.5590928	0.9017524

Figura 4.26. Misure di errore UBCF vs IBCF vs Random

4.5.4 Recommendation list: estrazione e visualizzazione

Dopo aver dedicato la sezione precedente alla scoperta del modello più adatto ai fini dell'analisi, quest'ultimo paragrafo si focalizzerà sulla presentazione dell'output finale. L'obiettivo ultimo infatti, risiede nell'estrazione e la seguente visualizzazione di una lista di 15 film da raccomandare a ciascun utente presente nel *test set.*
Quest'ultima è stata estratta grazie alla seguente funzione e salvata nell'oggetto "TOP15_UBCF_65N_p_z":

```
TOP15_UBCF_65N_p_z<-predict(UBCF_65N_p_z,getData(schema_valutazione,
"known"), n=15)
```

All'interno di esso sono memorizzate 100 *recommendation list* corrispondenti al numero di utenti presenti nel *test set* e contenenti l'ID di 15 film ciascuna con i corrispettivi *predicted rating*. Tuttavia, per una migliore comprensione, è stato necessario trasformare l'oggetto precedente in una matrice nominata "UBFC_matrice_raccomandazione" che per ogni utente restituisce i codici identificativi dei film raccomandati:

```
 UBCF_matrice_raccomandazione<-sapply(TOP15_UBCF_65N_p_z@item,
function(x){colnames(matrice1)[x] })
```

Nello specifico, ciascuna colonna della matrice corrisponde ad un utente, in riga invece ritroviamo l'elenco degli item che vanno a comporre la lista di raccomandazione.
In questo modo è possibile estrarre la *recommendation list* per qualsiasi utente appartenente al *test set.*
Di seguito verranno estratte a scopo esemplare due liste per i primi due utenti:

```
UBCF_matrice_raccomandazione [,1] #item raccomandati al primo utente

 "318"  "356"  "1196" "296"  "260"  "858"  "1136" "2571" "2858" "593"
"912"  "527"  "4306" "50"   "1198"
```

```
UBCF_matrice_raccomandazione[,2]#item raccomandati al secondo utente

"1748"  "3275"  "57669" "1148"  "1199"  "1921"  "1261"  "38061" "1090"
"1266"  "34405" "2019"  "2951"  "1274"  "5971"
```

In ultimo, per una visualizzazione più inuitiva, le *recommendation list* sono state inserite in grafici a griglia ed infine a ciascun codice identificativo dei film raccomandati vi è stato associato il corrispondente titolo.

```
UBCF_raccomandazione_utente1 <- matrix (0,15)
for (i in 1:15){UBCF_raccomandazione_utente1[i]<-
as.character(subset(movies,movies$ID_film==as.integer(lista_UBCF_65N
_p_z[[1]][i]))$titolo)}colnames(UBCF_raccomandazione_utente1)<-
c("Recommendation list utente 1")

grid.table(UBCF_raccomandazione_utente1)
```

Di seguito (Figura 4.27) la *recommandation list* finale per l'utente 1:

Recommendation list utente 1
Shawshank Redemption, The (1994)
Forrest Gump (1994)
Star Wars: Episode V - The Empire Strikes Back (1980)
Pulp Fiction (1994)
Star Wars: Episode IV - A New Hope (1977)
Godfather, The (1972)
Monty Python and the Holy Grail (1975)
Matrix, The (1999)
American Beauty (1999)
Silence of the Lambs, The (1991)
Casablanca (1942)
Schindler's List (1993)
Shrek (2001)
Usual Suspects, The (1995)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)

Figura 4.27. Recommendation List utente 1

Risulta interessante approfondire le caratteristiche delle *recommendation list* nella loro totalità. Nel complesso sono stati suggeriti 126 film sui 171 disponibili nel *test set*. Il grafico sottostante (Figura 4.28) descrive la distribuzione dei film all'interno delle liste; sull'asse delle ascisse troviamo il numero di volte che un film è stato suggerito e sull'asse delle ordinate la quantità di film che corrispondono a tale valore. Ad esempio, un valore dell'ascissa = 89 e di ordinata = 1 vorrà dire che un solo film è stato raccomandato 89 volte. Di conseguenza nella coda della distribuzione ritroviamo quei film che sono stati raccomandati un numero elevato di volte e praticamente quasi sempre presenti in tutte le liste di raccomandazione, viceversa, nella testa ritroviamo quei film che sono stati suggeriti in un numero di occasioni sostanzialmente inferiore.

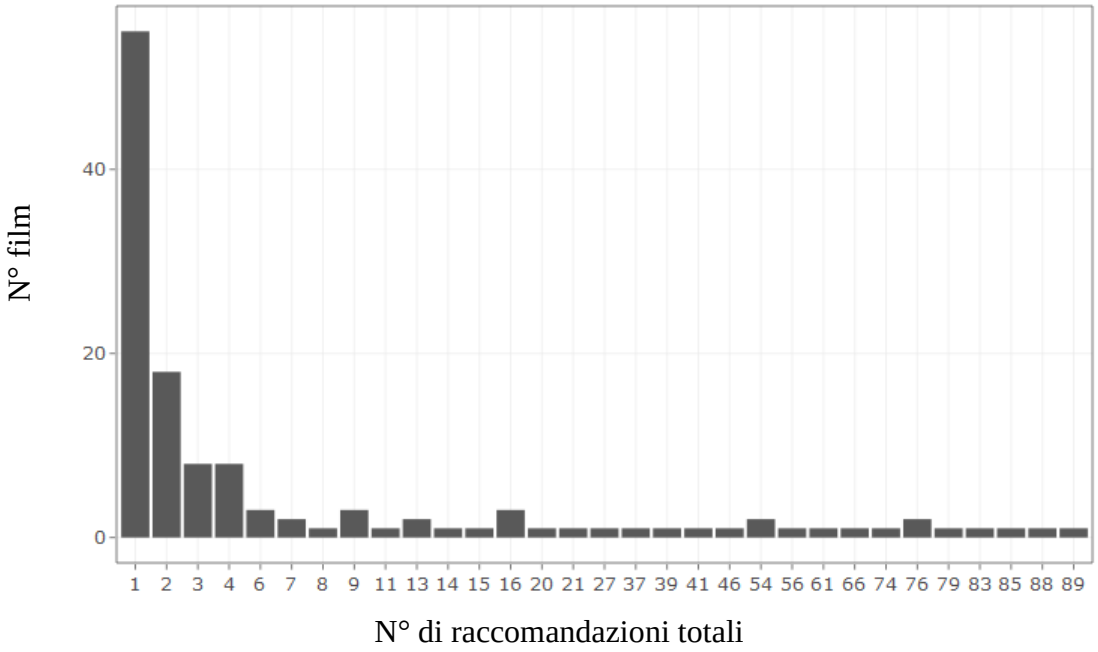

Figura 4.28. Distribuzione film Recommendation List

È proprio su questi ultimi che è importante focalizzare l'attenzione, in quanto, rispetto ai film più gettonati presenti nella coda che stravincono in termini di valutazioni medie e popolarità, essi risultano essere rari. La loro raccomandazione dunque, non essendo scontata rispecchia le caratteristiche di originalità e serendipità fondamentali per dare utilità e valore maggiore ad un sistema di raccomandazione.

Riproponendo l'esempio precedente, nella *Recommendation list* di 15 film dell'utente 1 è bene evidenziare nel grafico sottostante (Figura 4.29) quei film sono stati raccomandati pur non essendo estremamente popolari all'interno del *dataset* o non avendo una valutazione media palesemente positiva. Essi quindi, sono frutto di una raccomandazione personalizzata, sviluppata sulla base del comportamento di utenti simili grazie all'ausilio dell'algoritmo collaborativo *User-based*.

Recommendation list utente 1
Shawshank Redemption, The (1994)
Forrest Gump (1994)
Star Wars: Episode V - The Empire Strikes Back (1980)
Pulp Fiction (1994)
Star Wars: Episode IV - A New Hope (1977)
Godfather, The (1972)
Monty Python and the Holy Grail (1975)
Matrix, The (1999)
American Beauty (1999)
Silence of the Lambs, The (1991)
Casablanca (1942)
Schindler's List (1993)
Shrek (2001)
Usual Suspects, The (1995)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)

Figura 4.29. Recommendation List evidenziata

Conclusioni

Il lavoro di tesi ha contribuito *in primis*, a fornire una spiegazione teorica sull'origine, la rilevanza e la classificazione dei sistemi di raccomandazione, andando ad approfondire i differenti algoritmi alla base del loro funzionamento. In secondo luogo, ha descritto la metodologia con cui, a partire da un determinato *dataset*, è possibile costruire empiricamente un vero e proprio sistema di raccomandazione; in particolare, l'elaborato si è concentrato sull'applicazione di questi ultimi al mondo dei servizi video-streaming on demand (VOD). Esso è uno dei settori in cui i recommendation systems, a causa dell'enorme vastità dei cataloghi e la conseguente difficoltà di scelta da parte dell'utente, risultano una risorsa chiave per offrire una piattaforma che suggerisca o metta in primo piano soltanto quei titoli coerenti con i propri interessi, garantendo così un'esperienza di navigazione funzionale e piacevole.

Il cuore dell'analisi si è focalizzato sull'implementazione e sul confronto dei due algoritmi più celebri in letteratura: lo *User-Based Collaborative Filtering* (UBCF) e l'*Item-Based Collavorative Filtering* (IBCF). Prima dello sviluppo e la seguente comparazione dei risultati ottenuti dai due approcci, è stata eseguita, per entrambi, una prima fase di ottimizzazione dei parametri finalizzata alla massimizzazione delle prestazioni dei due modelli di riferimento. Ambedue gli algoritmi hanno dimostrato di fornire prestazioni migliori con l'utilizzo del metodo *Z-score* per quanto riguarda la normalizzazione dei dati e della distanza di Pearson per il calcolo delle similarità tra utenti e item.

Una volta selezionati i due modelli UBCF e IBCF ottimizzati si è proseguito con il confronto dei risultati e la conseguente scelta del modello di riferimento per l'output finale. Dalla comparazione è emerso che l'algoritmo *Item-Based* risulta particolarmente inadatto alla tipologia di *dataset* oggetto di analisi. Verificando l'accuratezza delle sue raccomandazioni è risultato che esse sono equiparabili a raccomandazioni *random*, sviluppate in modo casuale e senza l'ausilio di alcun algoritmo. Al contrario, L'approccio *User-based* ottimizzato, ha fornito una prestazione soddisfacente sia in termini di accuratezza (MAE = 0,68), che in termini qualitativi. Infatti, le *recommendation list* ottenute da esso, sono costituite non solo da suggerimenti prevedibili di film celebri e comunemente apprezzati, bensì anche titoli

più rari, frutto di suggerimenti personalizzati ritagliati su misura per gli utenti, che difficilmente sarebbero stati suggeriti in assenza di un sistema di raccomandazione.
L'analisi empirica si è dunque focalizzata su tecniche appartenenti all'approccio collaborativo che per lo sviluppo delle raccomandazioni prendono in considerazione i comportamenti degli utenti, ma non sfruttano alcuna informazione riguardante le caratteristiche descrittive dei titoli. In ottica futura sarebbe interessante estrarre informazioni rilevanti per ciascun titolo presente nel *dataset* (regista, trama, attori, argomenti trattati, anno di rilascio etc.), condurre uno studio che si concentri su tecniche *Content-based* ed infine metterne a confronto il livello di accuratezza e la qualità delle differenti liste di raccomandazione generate.

Comandi R

Ispezione, pulizia/manipolazione dei dati e analisi esplorativa

```
library(tidyverse)
library(dplyr)
library(ggplot2)
library(gridExtra)
library(plotly)

str(ratings)
ratings <- ratings[,-4] #elimino timestamp che non è funzionale
#userID e movieID da numero a fattore
ratings$userId <- as.factor(ratings$userId)
ratings$movieId <- as.factor(ratings$movieId)
colnames(ratings)<-c("ID_utente","ID_film", "rating") #rinomino
variabili
summary(ratings) #non ci sono valori nulli e variabili corrette
ratings_1<- ratings[1:10,]
grid.table(ratings_1) #visualizzazione delle prime 10 osservazioni

str(movies)
movies$movieId <- as.factor(movies$movieId)
colnames(movies)<-c("ID_film", "titolo", "genere")
summary(movies)
str(movies)
movies_2<- movies[1:10,]
grid.table(movies_2) #visualizzazione delle prime 10 osservazioni

tags <- tags[,-4]
tags$userId <- as.factor(tags$userId)
tags$movieId <- as.factor(tags$movieId)
colnames(tags)<-c("ID_utente", "ID_film", "tag")
tags$tag<-as.character(tags$tag)
summary(tags)
tags_1<- tags[1:10,]
grid.table(tags_1) #visualizzazione delle prime 10 osservazioni

#Analisi esplorativa
library(ggplot2)
library(plotly)

#comportamenti di rating
#istogrammi distribuzione rating per film e per utente
ratings %>% group_by(ID_film) %>% summarize(n = n()) %>%
ggplot(aes(n)) + geom_histogram(fill = "cadetblue",color= "black",
bins = 10) +
  scale_x_log10() + ggtitle("DISTRIBUZIONE DI RATING PER FILM")+
labs( x= "numero film", y ="numero di ratings") + theme_bw()-
>Distribfilm
ggplotly(Distribfilm)

ratings %>% group_by(ID_utente) %>% summarize(n = n()) %>%
  ggplot(aes(n)) + geom_histogram(fill = "cadetblue",color= "black",
bins = 20) +
```

```
  scale_x_log10() + ggtitle("DISTRIBUZIONE DI RATING PER UTENTE")+
labs( x= "numero utenti", y ="numero di ratings") + theme_bw()-
>Distributenti
ggplotly(Distributenti)


ratings_factor<-as.factor(ratings$rating)
ratings_factor<-data.frame(ratings_factor)
colnames(ratings_factor)<-c("rating")
ratings_factor %>% count(rating) %>% mutate(perc = n /
nrow(ratings_factor)) -> frequenza_ratings
colnames(frequenza_ratings)<-c("rating", "frequeza_assoluta",
"frequenza_percentuale")
comportamento_rating<-ggplot(frequenza_ratings,
aes(rating,frequenza_percentuale)) +
geom_bar(stat="identity")+theme_bw()+ggtitle("COMPORTAMENTO RATING")
comportamento_rating<-comportamento_rating +
scale_fill_manual(values = c("red4","red3","red2",
"darkorange2","darkorange1","yellow2","yellow3","green2",
"green3","green4"))
comportamento_rating<-comportamento_rating+scale_y_continuous(
breaks = c(0.05, 0.1,0.15,0.2,0.25))
ggplotly(comportamento_rating) #!!!bisogna modificare il colore
della scala e la grandezza!!
frequenza_ratings
(1370+2811+1791+7551+5550)/(1370+2811+1791+7551+5550+20047+12136+268
18+8551+13211) #percentuale count negativi
20047/(1370+2811+1791+7551+5550+20047+12136+26818+8551+13211)  #perc
count neutri
(12136+26818+8551+13211)/(1370+2811+1791+7551+5550+20047+12136+26818
+8551+13211)#perc count positivi
polarità_rating<- factor(c(1, 2, 3))
levels(polarità_rating) <- c("negativi","neutri","positivi")
frequenza_percentuale_ratings<-(c(0.19, 0.2, 0.61))
bilancio_rating <- data.frame(polarità_rating,
frequenza_percentuale_ratings)
bil_rating<-ggplot(bilancio_rating,
aes(polarità_rating,frequenza_percentuale_ratings)) +
geom_bar(stat="identity")+theme_bw()+ggtitle("POLARITÀ RATING")
bil_rating+scale_fill_manual(values = c("red4","red3"))

bil_rating<-ggplot(bilancio_rating,
aes(polarità_rating,frequenza_percentuale_ratings)) +
geom_bar(stat="identity")+theme_bw()+ggtitle("POLARITÀ RATING")
ggplotly(bil_rating) #!!!bisogna modificare il colore


str(rating_film)
#unione dataset ratings e film
rating_film<-merge(x=ratings,y=movies, by="ID_film")

#media rating per film
rating_film %>% group_by(titolo) %>% summarise(mean(rating))-
>rating_titoli
#frequenza film
frequenza_titoli<-rating_film %>%
group_by(titolo)%>%count(titolo,sort=TRUE)

#titoli più votati(elimati quelli meno di 50)
frequenza_titoli%>% filter(n > 49)->frequenza_titoli_famosi
colnames(frequenza_titoli_famosi)<-c("titolo","frequenza")
frequenza_top10 <- frequenza_titoli_famosi[1:10,]
frequenza_titoli_famosi %>% top_n(10, frequenza)->frequenza_top10
library(gridExtra)
```

```
grid.table(frequenza_top10)
ggplotly(ggplot(frequenza_top10, aes(x = reorder(titolo,frequenza),
y = frequenza)) + geom_bar(stat =
"identity",fill="BLACK")+theme_bw()+theme(axis.title.y
=element_blank())+coord_flip()+ labs(x="titolo")+ggtitle("10 TITOLI
PIÙ VISTI")+theme(axis.text = element_text(colour="black")))


#unione dataset titoli famosi e rating titoli per sapere rating
medio tra quelli visti almeno 50 volte
rating_famosi<-merge(x=frequenza_titoli_famosi,y=rating_titoli,
by="titolo")
rating_famosi <- rating_famosi[,-2]
colnames(rating_famosi)<-c("titolo","rating_medio")
rating_famosi %>% top_n(10, rating_medio)->rating_top10
rating_top10[order(-rating_top10$rating_medio),]->rating_top10
grid.table(rating_top10)

#separare anno dal titolo e creare un dataset nuovo con la variabile
anno (preso spunto da un blog)
movies$titolo<-as.character(movies$titolo)
movies_1 <- movies %>%
  # trim whitespaces
  mutate(title = str_trim(titolo)) %>%
  # split title to title, year
  extract(titolo, c("titolo_1", "anno"), regex = "^(.*) \\(([0-9 \\-
]*)\\)$", remove = F) %>%
  # for series take debut date
  mutate(anno = if_else(str_length(anno) > 4,
as.integer(str_split(anno, "-", simplify = T)[1]),
as.integer(anno))) %>%
  # replace title NA's with original title
  mutate(titolo = if_else(is.na(titolo_1), titolo, titolo_1)) %>%
  # drop title_tmp column
  select(-titolo_1)  %>%
  str(movies_1)
movies_1 <- movies_1[,-5] #elimino titolo vecchio
movies_1$titolo<-as.factor(movies_1$titolo)
str(movies_1)
#numero di film per anno
film_per_anno <- movies_1 %>%
  select(ID_film, anno) %>% # select columns we need
  group_by(anno) %>% # group by year
  summarise(count = n())  %>% # count movies per year
  arrange(anno)
summary(film_per_anno)
film_per_anno <-film_per_anno %>% na.omit()
colnames(film_per_anno)<-c("anno","frequenza")
ggplotly(ggplot(film_per_anno, aes(x = anno, y = frequenza)) +
geom_line()+theme_bw()+theme(axis.text = element_text(colour =
"black")))

#ESPLORARE I GENERI
generi<- movies %>%
  separate_rows(genere, sep = "\\|") %>%
  group_by(genere) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
generi %>% top_n(10, count)->generi_top10
colnames(generi_top10)<-c("Genere","Frequenza")
grid.table(generi_top10)


#come ognuno si evolve negli anni
```

```
generi_per_anno <- movies_1 %>%
  na.omit() %>%
  select(ID_film, anno, genere) %>%
  separate_rows(genere, sep = "\\|") %>%
  mutate(genere = as.factor(genere))%>%
  group_by(anno, genere) %>%
  summarise(count = n())

colnames(generi_per_anno)<-c("anno","genere","frequenza")
#selezionare solo alcuni generi
ggplotly(ggplot(generi_per_anno, aes(x = anno, y = frequenza))
+theme_bw()+
           geom_line(aes(color=genere)))
#visualizziamone alcuni in modo singolo per una migliore
visualizzazione
generi_per_anno %>%
  filter(genere %in% c("Drama", "Comedy", "Thriller")) %>%
  ggplot(aes(x = anno, y = frequenza))+theme_bw() +
  geom_line(aes(color=genere))->grafico
ggplotly(grafico)

generi_per_anno %>%
  filter(genere %in% c("Sci-Fi", "War")) %>%
  ggplot(aes(x = anno, y = frequenza))+theme_bw() +
  geom_line(aes(color=genere))->grafico2
ggplotly(grafico2)


# Tags per generi
tag_generi <- movies %>%
  na.omit() %>%
  select(ID_film,genere) %>%
  separate_rows(genere, sep = "\\|") %>%
  inner_join(tags, by = "ID_film") %>%
  select(genere, tag) %>%
  group_by(genere) %>%
  nest()
str(tag_generi)
tag_generi$genere<-as.factor(tag_generi$genere)

library("tm")
library("SnowballC")

Adventure<-tag_generi[1,2]%>%unnest()
Adventure$tag<-tolower(Adventure$tag)
freq_adventure_tag<-Adventure%>% group_by(tag) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
colnames(freq_adventure_tag)<-c("tag","frequenza")
ggplotly(ggplot(fre_top10_adv, aes(x = reorder(tag,frequenza), y =
frequenza)) + geom_bar(stat =
"identity",fill="green")+theme_bw()+coord_flip()+
labs(x="tag")+ggtitle("TAG FREQUENTI ADVENTURE")+theme(axis.text =
element_text(colour = "black")))

Drama<-tag_generi[7,2]%>%unnest()
Drama$tag<-tolower(Drama$tag)
freq_drama_tag<-Drama%>% group_by(tag) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
colnames(freq_drama_tag)<-c("tag","frequenza")
ggplotly(ggplot(fre_top10_drama, aes(x = reorder(tag,frequenza), y =
frequenza)) + geom_bar(stat =
"identity",fill="orange")+theme_bw()+coord_flip()+
```

```
labs(x="tag")+ggtitle("TAG FREQUENTI DRAMA")+theme(axis.text =
element_text(colour = "black")))

Thriller<-tag_generi[9,2]%>%unnest()
Thriller$tag<-tolower(Thriller$tag)
freq_thriller_tag<-Thriller%>% group_by(tag) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
colnames(freq_thriller_tag)<-c("tag","frequenza")
ggplotly(ggplot(fre_top10_thr, aes(x = reorder(tag,frequenza), y =
frequenza)) + geom_bar(stat = "identity")+theme_bw()+coord_flip()+
labs(x="tag")+ggtitle("TAG FREQUENTI THRILLER"))

Crime<-tag_generi[8,2]%>%unnest()
Crime$tag<-tolower(Crime$tag)
freq_crime_tag<-Crime%>% group_by(tag) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
colnames(freq_crime_tag)<-c("tag","frequenza")
ggplotly(ggplot(fre_top10_crim, aes(x = reorder(tag,frequenza), y =
frequenza)) + geom_bar(stat =
"identity",fill="red")+theme_bw()+coord_flip()+
labs(x="tag")+ggtitle("TAG FREQUENTI CRIME")+theme(axis.text =
element_text(colour="black")))
```

Implementazione Recommendation system

```
library(reshape2)
library(recommenderlab)
library(ggplot2)
library(plotly)
library(gridExtra)
ratings <- ratings[,-4] #elimino timestamp
#userID e movieID da numero a fattore
ratings$userId <- as.factor(ratings$userId)
ratings$movieId <- as.factor(ratings$movieId)
colnames(ratings)<-c("ID_utente","ID_film", "rating")
summary(ratings) #non ci sono valori nulli e variabili corrette

movies$movieId <- as.factor(movies$movieId)
colnames(movies)<-c("ID_film", "titolo", "genere")
summary(movies)

#unione dataset ratings e film
rating_film<-merge(x=ratings,y=movies, by="ID_film")
str(rating_film)

#creazione matrice rating(user-item, cella=rating)
matrice <- dcast(rating_film, ID_utente~ID_film, value.var =
"rating", na.rm=FALSE)
matrice <- as.matrix(matrice[,-1]) #rimozione id utente per poterla
trasformare

#trasformazione in un ogetto utilizzabile per l'implementazione di
recommenderlab (risparmio di memoria di 9 volte)
matrice <- as(matrice, "realRatingMatrix") #descriverla:
min(rowCounts(matrice)) #capire il minimo numero di film che ogni
utente ha valutato
```

```
min(colCounts(matrice)) #capire il minimo delle volte che un item è
stato valutato

#ridurre la matrice inserendo solo utenti e item rilevanti (che
hanno partecipato di più) per evitare noise
matrice1 <- matrice[rowCounts(matrice) > 30,  colCounts(matrice) >
30]
matrice1 #nuovo dataset  498 utenti, 860 film, 55360 ratings


#elenco di tutti i modelli utilizzabili con le caratteristiche
descrizione_modelli<-recommenderRegistry$get_entries(dataType
="realRatingMatrix")
#comprendiamo user based e item based e rispettivi parametri
standard
descrizione_modelli
descrizione_modelli[["UBCF_realRatingMatrix"]]
descrizione_modelli[["IBCF_realRatingMatrix"]]

#divisione in train e test + creazione dello schema di valutazione
set.seed(1) #per poter reiterare una procedura casuale
schema_valutazione <- evaluationScheme(matrice1, method = "split",
train = .8, given = 9, goodRating = 4)
schema_valutazione

#Implementazione UBCF con parametri default
UBCF<- Recommender(getData(schema_valutazione, "train"), method =
"UBCF") #modello sul train
pred_UBCF<- predict(UBCF, getData(schema_valutazione, "known"),
type="ratings") #predizioni sul test
valut_UBCF<-
calcPredictionAccuracy(pred_UBCF,getData(schema_valutazione,
"unknown")) #calcolo accuratezza
valut_UBCF #RMSE, MSE, MAE

recs <- c(1,5,10, 15, 20,25,30) #impostazioni di diverse reco list
come soglie per disegnare ROC
valut2_UBCF<-evaluate(schema_valutazione, method = "UBCF", n=recs,
progress= TRUE)
plot(x = valut2_UBCF, y = "ROC", annotate=4,legend="topleft")

#Sebbene le misure di accuratezza sono accettabili la ROC non è
soddisfacente; quindi si prova ad ottimizzare i parametri
#Fissiamo 5 livelli di Neighboor (25, 45, 65, 85, 100) e per ognuno
proviamo 6 modelli derivanti dalla combinazione degli altri due
parametri (3 tipi di distanze X 2 tipi di normalizzazione) per un
totale di 30 modelli


#iniziamo con Neighboor 25
compara_modelli_25 <-list(

  "25N_cos_c" = list(name="UBCF", param=list(normalize = "center",
                                             method="Cosine",
                                             nn=25)),
  "25N_p_c" = list(name="UBCF", param=list(normalize = "center",
                                           method="pearson",
                                           nn=25)),
  "25N_j_c" = list(name="UBCF", param=list(normalize = "center",
                                           method="jaccard",
                                           nn=25)),
  "25N_cos_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                             method="Cosine",
                                             nn=25)),
```

```
  "25N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=25)),
  "25N_j_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="jaccard",
                                           nn=25)))
risult_modelli_25 <- evaluate(schema_valutazione,
compara_modelli_25, n = recs, progress = TRUE)
plot(x = risult_modelli_25, y = "ROC", annotate = 5,
legend="topleft")

#errori NN 25
#calcola singolarmente predizione ed errori
UBCF_25N_cos_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                      param=list(normalize = "center",
method="Cosine", nn=25))
pred_UBCF_25N_cos_c<- predict(UBCF_25N_cos_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_25N_cos_c <- calcPredictionAccuracy(pred_UBCF_25N_cos_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_cos_c

UBCF_25N_p_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                      param=list(normalize = "center",
method="pearson", nn=25))
pred_UBCF_25N_p_c<- predict(UBCF_25N_p_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_25N_p_c <- calcPredictionAccuracy(pred_UBCF_25N_p_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_p_c

UBCF_25N_j_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                      param=list(normalize = "center",
method="jaccard", nn=25))
pred_UBCF_25N_j_c<- predict(UBCF_25N_j_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_25N_j_c <- calcPredictionAccuracy(pred_UBCF_25N_j_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_j_c

UBCF_25N_cos_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                       param=list(normalize = "Z-score",
method="Cosine", nn=25))
pred_UBCF_25N_cos_z<- predict(UBCF_25N_cos_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_25N_cos_z <- calcPredictionAccuracy(pred_UBCF_25N_cos_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_cos_z

UBCF_25N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="pearson", nn=25))
pred_UBCF_25N_p_z<- predict(UBCF_25N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
```

```
valut_UBCF_25N_p_z <- calcPredictionAccuracy(pred_UBCF_25N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_p_z

UBCF_25N_j_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="jaccard", nn=25))
pred_UBCF_25N_j_z<- predict(UBCF_25N_j_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_25N_j_z <- calcPredictionAccuracy(pred_UBCF_25N_j_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_j_z

error_UBFC_25NN<-
rbind(valut_UBCF_25N_cos_c,valut_UBCF_25N_cos_z,valut_UBCF_25N_j_c,v
alut_UBCF_25N_j_z,valut_UBCF_25N_p_c,valut_UBCF_25N_p_z)
error_UBFC_25NN

#ROC ed errori dicono che il migliore tra i 25 è pearson + z-score.
adesso verifichiamo nn 45
compara_modelli_45 <-list(

  "45N_cos_c" = list(name="UBCF", param=list(normalize = "center",
                                             method="Cosine",
                                             nn=45)),
  "45N_p_c" = list(name="UBCF", param=list(normalize = "center",
                                           method="pearson",
                                           nn=45)),
  "45N_j_c" = list(name="UBCF", param=list(normalize = "center",
                                           method="jaccard",
                                           nn=45)),
  "45N_cos_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                             method="Cosine",
                                             nn=45)),
  "45N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=45)),
  "45N_j_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="jaccard",
                                           nn=45)))
risult_modelli_45 <- evaluate(schema_valutazione,
compara_modelli_45, n = recs, progress = TRUE)
plot(x = risult_modelli_45, y = "ROC", annotate = 5,
legend="topleft")


#errori 45
#calcola singolarmente predizione ed errori
UBCF_45N_cos_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                             param=list(normalize = "center",
method="Cosine", nn=45))
pred_UBCF_45N_cos_c<- predict(UBCF_45N_cos_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_45N_cos_c <- calcPredictionAccuracy(pred_UBCF_45N_cos_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_45N_cos_c

UBCF_45N_p_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
```

```
                            param=list(normalize = "center",
method="pearson", nn=25))
pred_UBCF_45N_p_c<- predict(UBCF_45N_p_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_45N_p_c <- calcPredictionAccuracy(pred_UBCF_45N_p_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_45N_p_c

UBCF_45N_j_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                            param=list(normalize = "center",
method="jaccard", nn=45))
pred_UBCF_45N_j_c<- predict(UBCF_45N_j_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_45N_j_c <- calcPredictionAccuracy(pred_UBCF_45N_j_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_45N_j_c

UBCF_45N_cos_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                              param=list(normalize = "Z-score",
method="Cosine", nn=45))
pred_UBCF_45N_cos_z<- predict(UBCF_45N_cos_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_45N_cos_z <- calcPredictionAccuracy(pred_UBCF_45N_cos_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_45N_cos_z

UBCF_45N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                            param=list(normalize = "Z-score",
method="pearson", nn=45))
pred_UBCF_45N_p_z<- predict(UBCF_45N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_45N_p_z <- calcPredictionAccuracy(pred_UBCF_45N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_45N_p_z

UBCF_45N_j_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                            param=list(normalize = "Z-score",
method="jaccard", nn=45))
pred_UBCF_45N_j_z<- predict(UBCF_45N_j_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_45N_j_z <- calcPredictionAccuracy(pred_UBCF_45N_j_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_45N_j_z

error_UBFC_45NN<-
rbind(valut_UBCF_45N_cos_c,valut_UBCF_45N_cos_z,valut_UBCF_45N_j_c,v
alut_UBCF_45N_j_z,valut_UBCF_45N_p_c,valut_UBCF_45N_p_z)
error_UBFC_45NN
#tra 45 migliore è pearson con z-score. verifichiamo NN 65

compara_modelli_65 <-list(

  "65N_cos_c" = list(name="UBCF", param=list(normalize = "center",
                                             method="Cosine",
                                             nn=65)),
```

```
  "65N_p_c" = list(name="UBCF", param=list(normalize = "center",
                                           method="pearson",
                                           nn=65)),
  "65N_j_c" = list(name="UBCF", param=list(normalize = "center",
                                           method="jaccard",
                                           nn=65)),
  "65N_cos_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                             method="Cosine",
                                             nn=65)),
  "65N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=65)),
  "65N_j_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="jaccard",
                                           nn=65)))
risult_modelli_65 <- evaluate(schema_valutazione,
compara_modelli_65, n = recs, progress = TRUE)
plot(x = risult_modelli_65, y = "ROC", annotate = 5,
legend="topleft")

#errori 65
#calcola singolarmente predizione ed errori
UBCF_65N_cos_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                             param=list(normalize = "center",
method="Cosine", nn=65))
pred_UBCF_65N_cos_c<- predict(UBCF_65N_cos_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_65N_cos_c <- calcPredictionAccuracy(pred_UBCF_65N_cos_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_65N_cos_c

UBCF_65N_p_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "center",
method="pearson", nn=65))
pred_UBCF_65N_p_c<- predict(UBCF_65N_p_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_65N_p_c <- calcPredictionAccuracy(pred_UBCF_65N_p_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_65N_p_c

UBCF_65N_j_c<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "center",
method="jaccard", nn=65))
pred_UBCF_65N_j_c<- predict(UBCF_65N_j_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_65N_j_c <- calcPredictionAccuracy(pred_UBCF_65N_j_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_65N_j_c

UBCF_65N_cos_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                             param=list(normalize = "Z-score",
method="Cosine", nn=65))
pred_UBCF_65N_cos_z<- predict(UBCF_65N_cos_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_65N_cos_z <- calcPredictionAccuracy(pred_UBCF_65N_cos_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
```

```
valut_UBCF_65N_cos_z

UBCF_65N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="pearson", nn=65))
pred_UBCF_65N_p_z<- predict(UBCF_65N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_65N_p_z <- calcPredictionAccuracy(pred_UBCF_65N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_65N_p_z

UBCF_65N_j_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="jaccard", nn=65))
pred_UBCF_65N_j_z<- predict(UBCF_65N_j_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_65N_j_z <- calcPredictionAccuracy(pred_UBCF_65N_j_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_65N_j_z

error_UBFC_65NN<-
rbind(valut_UBCF_65N_cos_c,valut_UBCF_65N_cos_z,valut_UBCF_65N_j_c,v
alut_UBCF_65N_j_z,valut_UBCF_65N_p_c,valut_UBCF_65N_p_z)
error_UBFC_65NN
#ANCHE NEL 65 MEGLIO PERSON CON Z-SCORE. SEMBRA CHE LO USERBASED
FUNZIONI MEGLIO COSI! A QUESTO PUNTO PROVIAMO A CONFRONTARE TUTTI I
MIGLIORI PER CIASCUN NN

#CONFRONTO TUTTI I NN p_z
compara_modelli_UB_NN_p_z <-list(

  "25N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=25)),
  "45N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=45)),
  "65N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=65)),
  "85N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=85)),
  "100N_p_z" = list(name="UBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           nn=100)))

risult_modelli_UB_NN_p_z <- evaluate(schema_valutazione,
compara_modelli_UB_NN_p_z, n = recs, progress = TRUE)
plot(x = risult_modelli_UB_NN_p_z, y = "ROC", annotate = 3,
legend="topleft")

#errori
UBCF_25N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="pearson", nn=25))
pred_UBCF_25N_p_z<- predict(UBCF_25N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
```

```
valut_UBCF_25N_p_z <- calcPredictionAccuracy(pred_UBCF_25N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_25N_p_z

UBCF_45N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="pearson", nn=45))
pred_UBCF_45N_p_z<- predict(UBCF_45N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_45N_p_z <- calcPredictionAccuracy(pred_UBCF_45N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_45N_p_z

UBCF_65N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="pearson", nn=65))
pred_UBCF_65N_p_z<- predict(UBCF_65N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_65N_p_z <- calcPredictionAccuracy(pred_UBCF_65N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_65N_p_z

UBCF_85N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="pearson", nn=85))
pred_UBCF_85N_p_z<- predict(UBCF_85N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_85N_p_z <- calcPredictionAccuracy(pred_UBCF_85N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_85N_p_z

UBCF_100N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "UBCF",
                           param=list(normalize = "Z-score",
method="pearson", nn=100))
pred_UBCF_100N_p_z<- predict(UBCF_100N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_UBCF_100N_p_z <- calcPredictionAccuracy(pred_UBCF_100N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_UBCF_100N_p_z



error_UBFC_NN_p_z<-
rbind(valut_UBCF_25N_p_z,valut_UBCF_45N_p_z,valut_UBCF_65N_p_z,valut
_UBCF_85N_p_z,valut_UBCF_100N_p_z)
error_UBFC_NN_p_z


#user based migliore considerando l'indicatore MAE è 65NN CON
PEARSON e Z-SCORE.

#Implementiamo ITEM BASED

IBCF<- Recommender(getData(schema_valutazione, "train"), method =
"IBCF") #modello sul train
```

```
pred_IBCF<- predict(IBCF, getData(schema_valutazione, "known"),
type="ratings") #predizioni sul test
valut_IBCF<-
calcPredictionAccuracy(pred_IBCF,getData(schema_valutazione,
"unknown")) #calcolo accuratezza
valut_IBCF

valut2_IBCF<-evaluate(schema_valutazione, method = "IBCF", n=recs,
progress= TRUE)
getConfusionMatrix(valut2_IBCF) #matrice di confusione
plot(x = valut2_IBCF, y = "ROC", annotate=4,legend="topleft")

#molto impreciso. proviamo ad ottimizzare i parametri con la stessa
procedura utilizzata in precedenza
descrizione_modelli[["IBCF_realRatingMatrix"]]

compara_modelli_IB_30<-list(

  "30N_cos_c" = list(name="IBCF", param=list(normalize = "center",
                                             method="Cosine",
                                             k=30)),
  "30N_p_c" = list(name="IBCF", param=list(normalize = "center",
                                           method="pearson",
                                           k=30)),
  "30N_j_c" = list(name="IBCF", param=list(normalize = "center",
                                           method="jaccard",
                                           k=30)),
  "30N_cos_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                             method="Cosine",
                                             k=30)),
  "30N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           k=30)),
  "30N_j_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="jaccard",
                                           k=30)))
risult_modelli_IB_30 <- evaluate(schema_valutazione,
compara_modelli_IB_30, n = recs, progress = TRUE)
plot(x = risult_modelli_IB_30, y = "ROC", annotate = 5,
legend="topleft")

#errori 30
#calcola singolarmente predizione ed errori
IBCF_30N_cos_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                             param=list(normalize = "center",
method="Cosine", k=30))
pred_IBCF_30N_cos_c<- predict(IBCF_30N_cos_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_30N_cos_c <- calcPredictionAccuracy(pred_IBCF_30N_cos_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_30N_cos_c

IBCF_30N_p_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "center",
method="pearson", k=30))
pred_IBCF_30N_p_c<- predict(IBCF_30N_p_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_30N_p_c <- calcPredictionAccuracy(pred_IBCF_30N_p_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_30N_p_c
```

```
IBCF_30N_j_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "center",
method="jaccard", k=30))
pred_IBCF_30N_j_c<- predict(IBCF_30N_j_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_30N_j_c <- calcPredictionAccuracy(pred_IBCF_30N_j_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_30N_j_c

IBCF_30N_cos_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                             param=list(normalize = "Z-score",
method="Cosine", k=30))
pred_IBCF_30N_cos_z<- predict(IBCF_30N_cos_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_30N_cos_z <- calcPredictionAccuracy(pred_IBCF_30N_cos_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_30N_cos_z

IBCF_30N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="pearson", k=30))
pred_IBCF_30N_p_z<- predict(IBCF_30N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_30N_p_z <- calcPredictionAccuracy(pred_IBCF_30N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_30N_p_z

IBCF_30N_j_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="jaccard", k=30))
pred_IBCF_30N_j_z<- predict(IBCF_30N_j_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_30N_j_z <- calcPredictionAccuracy(pred_IBCF_30N_j_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_30N_j_z

error_IBFC_30NN<-
rbind(valut_IBCF_30N_cos_c,valut_IBCF_30N_cos_z,valut_IBCF_30N_j_c,v
alut_IBCF_30N_j_z,valut_IBCF_30N_p_c,valut_IBCF_30N_p_z)
error_IBFC_30NN
#più accurato risulta 30n con pearson e center; vediamo 45
compara_modelli_IB_45<-list(

  "45N_cos_c" = list(name="IBCF", param=list(normalize = "center",
                                             method="Cosine",
                                             k=45)),
  "45N_p_c" = list(name="IBCF", param=list(normalize = "center",
                                           method="pearson",
                                           k=45)),
  "45N_j_c" = list(name="IBCF", param=list(normalize = "center",
                                           method="jaccard",
                                           k=45)),
  "45N_cos_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                             method="Cosine",
                                             k=45)),
```

```
  "45N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           k=45)),
  "45N_j_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="jaccard",
                                           k=45)))
risult_modelli_IB_45<- evaluate(schema_valutazione,
compara_modelli_IB_45, n = recs, progress = TRUE)
plot(x = risult_modelli_IB_45, y = "ROC", annotate = 5,
legend="topleft")

#errori 45
#calcola singolarmente predizione ed errori
IBCF_45N_cos_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                             param=list(normalize = "center",
method="Cosine", k=45))
pred_IBCF_45N_cos_c<- predict(IBCF_45N_cos_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_45N_cos_c <- calcPredictionAccuracy(pred_IBCF_45N_cos_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_45N_cos_c

IBCF_45N_p_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "center",
method="pearson", k=45))
pred_IBCF_45N_p_c<- predict(IBCF_45N_p_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_45N_p_c <- calcPredictionAccuracy(pred_IBCF_45N_p_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_45N_p_c

IBCF_45N_j_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "center",
method="jaccard", k=45))
pred_IBCF_45N_j_c<- predict(IBCF_45N_j_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_45N_j_c <- calcPredictionAccuracy(pred_IBCF_45N_j_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_45N_j_c

IBCF_45N_cos_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                             param=list(normalize = "Z-score",
method="Cosine", k=45))
pred_IBCF_45N_cos_z<- predict(IBCF_45N_cos_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_45N_cos_z <- calcPredictionAccuracy(pred_IBCF_45N_cos_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_45N_cos_z

IBCF_45N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="pearson", k=45))
pred_IBCF_45N_p_z<- predict(IBCF_45N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
```

```
valut_IBCF_45N_p_z <- calcPredictionAccuracy(pred_IBCF_45N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_45N_p_z

IBCF_45N_j_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="jaccard", k=45))
pred_IBCF_45N_j_z<- predict(IBCF_45N_j_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_45N_j_z <- calcPredictionAccuracy(pred_IBCF_45N_j_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_45N_j_z

error_IBFC_45NN<-
rbind(valut_IBCF_45N_cos_c,valut_IBCF_45N_cos_z,valut_IBCF_45N_j_c,v
alut_IBCF_45N_j_z,valut_IBCF_45N_p_c,valut_IBCF_45N_p_z)
error_IBFC_45NN
#meglio 45 pearson z score nella roc; vediamo 65

compara_modelli_IB_65<-list(

  "65N_cos_c" = list(name="IBCF", param=list(normalize = "center",
                                             method="Cosine",
                                             k=65)),
  "65N_p_c" = list(name="IBCF", param=list(normalize = "center",
                                           method="pearson",
                                           k=65)),
  "65N_j_c" = list(name="IBCF", param=list(normalize = "center",
                                           method="jaccard",
                                           k=65)),
  "65N_cos_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                             method="Cosine",
                                             k=65)),
  "65N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="pearson",
                                           k=65)),
  "65N_j_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="jaccard",
                                           k=65)))
risult_modelli_IB_65<- evaluate(schema_valutazione,
compara_modelli_IB_65, n = recs, progress = TRUE)
plot(x = risult_modelli_IB_65, y = "ROC", annotate = 5,
legend="topleft")

#errori 65
#calcola singolarmente predizione ed errori
IBCF_65N_cos_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                             param=list(normalize = "center",
method="Cosine", k=65))
pred_IBCF_65N_cos_c<- predict(IBCF_65N_cos_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_65N_cos_c <- calcPredictionAccuracy(pred_IBCF_65N_cos_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_65N_cos_c

IBCF_65N_p_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "center",
method="pearson", k=65))
```

```
pred_IBCF_65N_p_c<- predict(IBCF_65N_p_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_65N_p_c <- calcPredictionAccuracy(pred_IBCF_65N_p_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_65N_p_c

IBCF_65N_j_c<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "center",
method="jaccard", k=65))
pred_IBCF_65N_j_c<- predict(IBCF_65N_j_c,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_65N_j_c <- calcPredictionAccuracy(pred_IBCF_65N_j_c,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_65N_j_c

IBCF_65N_cos_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                             param=list(normalize = "Z-score",
method="Cosine", k=65))
pred_IBCF_65N_cos_z<- predict(IBCF_65N_cos_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_65N_cos_z <- calcPredictionAccuracy(pred_IBCF_65N_cos_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_65N_cos_z

IBCF_65N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="pearson", k=65))
pred_IBCF_65N_p_z<- predict(IBCF_65N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_65N_p_z <- calcPredictionAccuracy(pred_IBCF_65N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_65N_p_z

IBCF_65N_j_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="jaccard", k=65))
pred_IBCF_65N_j_z<- predict(IBCF_65N_j_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_65N_j_z <- calcPredictionAccuracy(pred_IBCF_65N_j_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_65N_j_z

error_IBFC_65NN<-
rbind(valut_IBCF_65N_cos_c,valut_IBCF_65N_cos_z,valut_IBCF_65N_j_c,v
alut_IBCF_65N_j_z,valut_IBCF_65N_p_c,valut_IBCF_65N_p_z)
error_IBFC_65NN

#anche per item based è person e zscore (evitiamo 85 e 10 e ansdiamo
a confrontare i migliori pearson e z score)

#CONFRONTO TUTTI I NN p_z per item based
compara_modelli_IB_NN_p_z <-list(

  "30N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                           method="pearson",
```

```
                                              k=30)),
  "45N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                              method="pearson",
                                              k=45)),
  "65N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                              method="pearson",
                                              k=65)),
  "85N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                              method="pearson",
                                              k=85)),
  "100N_p_z" = list(name="IBCF", param=list(normalize = "Z-score",
                                               method="pearson",
                                               k=100)))

risult_modelli_IB_NN_p_z <- evaluate(schema_valutazione,
compara_modelli_IB_NN_p_z, n = recs, progress = TRUE)
plot(x = risult_modelli_IB_NN_p_z, y = "ROC", annotate = 2,
legend="topleft")

#errori
IBCF_30N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="pearson", k=30))
pred_IBCF_30N_p_z<- predict(IBCF_30N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_30N_p_z <- calcPredictionAccuracy(pred_IBCF_30N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_30N_p_z

IBCF_45N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="pearson", k=45))
pred_IBCF_45N_p_z<- predict(IBCF_45N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_45N_p_z <- calcPredictionAccuracy(pred_IBCF_45N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_45N_p_z

IBCF_65N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="pearson", k=65))
pred_IBCF_65N_p_z<- predict(IBCF_65N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_65N_p_z <- calcPredictionAccuracy(pred_IBCF_65N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_65N_p_z

IBCF_85N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                           param=list(normalize = "Z-score",
method="pearson", k=85))
pred_IBCF_85N_p_z<- predict(IBCF_85N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_85N_p_z <- calcPredictionAccuracy(pred_IBCF_85N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_85N_p_z
```

```
IBCF_100N_p_z<- Recommender(getData(schema_valutazione, "train"),
method = "IBCF",
                            param=list(normalize = "Z-score",
method="pearson", k=100))
pred_IBCF_100N_p_z<- predict(IBCF_100N_p_z,
getData(schema_valutazione, "known"), type="ratings") #predizioni
sul test
valut_IBCF_100N_p_z <- calcPredictionAccuracy(pred_IBCF_100N_p_z,
getData(schema_valutazione, "unknown")) #calcolo accuratezza
valut_IBCF_100N_p_z



error_IBFC_NN_p_z<-
rbind(valut_IBCF_30N_p_z,valut_IBCF_45N_p_z,valut_IBCF_65N_p_z,valut
_IBCF_85N_p_z,valut_IBCF_100N_p_z)
error_IBFC_NN_p_z

#A questo punto selezionare il migliore user based, il migliore item
based e un sistema "random"
compara_modelli_finale <-list(

  "UBCF 65N_p_z" = list(name="UBCF", param=list(normalize = "Z-
score",
                                                method="pearson",
                                                nn=30)),
  "IBCF 45N_p_z" = list(name="IBCF", param=list(normalize = "Z-
score",
                                                method="pearson",
                                                k=45)),
  "RANDOM" = list(name="RANDOM"),
  "POPULAR"=list(name="POPULAR"))


risult_modelli_finale<- evaluate(schema_valutazione,
compara_modelli_finale, n = recs, progress = TRUE)
plot(x = risult_modelli_finale, y = "ROC", annotate = 1,
legend="topleft")

#ERRORI FINALE

RANDOM<- Recommender(getData(schema_valutazione, "train"), method =
"RANDOM") #modello sul train
pred_RANDOM<- predict(RANDOM, getData(schema_valutazione, "known"),
type="ratings") #predizioni sul test
valut_RANDOM<-
calcPredictionAccuracy(pred_RANDOM,getData(schema_valutazione,
"unknown")) #calcolo accuratezza
valut_RANDOM


error_finale<-
rbind(valut_RANDOM,valut_UBCF_65N_p_z,valut_IBCF_45N_p_z)
error_finale


#fornire la lista considerando il modello migliore: USER BASED
TOP15_UBCF_65N_p_z <- predict(UBCF_65N_p_z,
getData(schema_valutazione, "known"), n=15)
#trasforma predizioni in una matrice che per ogni utente del test
set restituisce gli items
UBCF_matrice_raccomandazione <- sapply(TOP15_UBCF_65N_p_z@items,
function(x){   colnames(matrice1)[x] })
dim(UBCF_matrice_raccomandazione)
```

```
UBCF_matrice_raccomandazione[, 1] #item raccomandati al primo utente
UBCF_matrice_raccomandazione[, 2] #item raccomandati al secondo
utente


#visualizzare in griglia con titoli
UBCF_raccomandazione_utente1 <- matrix(0,15)
for (i in 1:15){UBCF_raccomandazione_utente1[i] <-
as.character(subset(movies, movies$ID_film ==
as.integer(lista_UBCF_65N_p_z[[1]][i]))$titolo)}
colnames(UBCF_raccomandazione_utente1)<-c("Recommendation list
utente 1")
grid.table(UBCF_raccomandazione_utente1)


#caratteristiche lista
UBCF_n_film_raccomandati<-
factor(table(UBCF_matrice_raccomandazione))
ggplotly(qplot(UBCF_n_film_raccomandati)+theme_bw())
```

BIBLIOGRAFIA

Adomavicius, G & Tuzhilin A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering.

Agrawal. R., & Srikant., R. (1994). Fast algorithms for mining association rules. Proceeding VLDB '94 Proceedings of the 20th International Conference on Very Large Data Bases. pp 487-499

Anand, S. S. & Mobasher, B. (2005). Intelligent techniques in Web Personalization. Eds. LNAI 3169. Springer-Verlag, 1-37.

Anderson, C. (2006). The Long Tail: Why the Future of Business is Selling Less of More, Hyperion Books, New York.

Balabanovic, M., & Shoham, Y. (1997). Fab: Content-based, Collaborative Recommendation. Communications of the ACM40(3), 66–72.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning, Springer.

Billsus, D., & Pazzani M.J. (1999). A hybrid user model for news story classification. In: Proceedings of the seventh international conference on user modeling, Banff, Canada. Springer-Verlag, New York; 1999. p. 99-108

Breese, J., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence.

Burke., R. (2002). Hybrid Recommender Systems: Survey and Experiments, User Modeling and User-Adapted Interaction 12(4)

Charu, C. (2016). Recommender Systems: The Textbook.

Chaturvedi, A.K. (2017). Recommender System for News Articles using Supervised Learning.

Claypool, M., Gokhale. A., Miranda, T., Murnikov, P., Netes, D., & Sartin M. (1999). Combining content-based and collaborative based filters in an online newspaper.

Cotter, P., & Smyth, B., (2000). IPTV: Intelligent personalized TV guides. In: Twelfith conference on innovative applications of artificial intelligence.

Deerwester, S., Dumais, S.T., Furnas, G. W., Landauer T.K., & R. Harshman. (1990). Indexing by latent semantic analysis, Journal of the American Society for Information Science 41, 391–407.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. Mach Learn.

Gomez-Uribe, C. A., & Hunt, N. (2015). The Netflix Recommender System. ACM Transactions on Management Information Systems. 6 (4): 1–19.

Herlocker, J., Konstan, J., Terveen, G., & Riedl, T. (2004). Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems 22, 25–53.

Iyengar, S., & Lepper, M. (2000). When choice is demotivating: Can one desire too much of a good thing?. Journal of Personality and Social Psychology, 75,pp 995--100.

Isinkaye, F., Folajimi, Y., & Ojokoh, B. (2015). Recommendation systems: Principles, methods and evaluation. Egyptian Informatics Journal 16.

Hahsler, M. (2015). recommenderlab: A Framework for Developing and Testing Recommendation Algorithms.

Harper, F.M., Konstan, J.A. (2015). The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015).

Herlocker, L., Konstan, J.A. et al. (1999). An Algorithmic Framework for Performing Collaborative Filtering, Proceedings of the 22nd Annual International ACM SIGIR Conference, ACM Press, pp. 230–237.

Jannach, D., Zanker, M., Felfernig, A., & Friedrich, G. (2010). Recommender Systems: An Introduction.

Kaisler, S., Armour, F., Espinosa, J.A. & Money, W. (2013). Big Data: Issues and Challenges Moving Forward. Conference: System Sciences (HICSS), 2013 46th Hawaii International Conference.

Leonard, D. (2016). Spotify Is Perfecting the Art of the Playlist. Bloomberg Businessweek.

Linden, G., Smith, B & York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. IEEE Internet Computing, 7, 76-80.

Lops, P., De Gemmis, M., & Semeraro, G. (2011) Chapter 3, Content-based Recommender Systems: State of the Art and Trends.

Manning, C.D., Raghavan, P., & Schtze, H. (2008). Introduction to Information Retrieval. Cambridge

Melville, P., & Sindhawani, V. (2010). Encyclopedia of Machine Learning, Chapter No:00338.

Montaner, M., Lopez, B., & de la Rosa, J. (2003). A taxonomy of recommender agents on the internet. Artificial Intelligence Review 19 (4), 285–330.

Pelanek, R. (2018). Evaluation of Recommender system.

Ricci, F., Rokach, L., & Shapira, B. (2011). Recommender Systems Handbook.

Sarwar, B., Karypis G., Konstan J., & Reidl J. (2001) Item-based collaborative filtering recommendation algorithms. In. Proceedings of the 10th international conference on World Wide Web, ACM, Hong Kong, China, 295.

Sarwar B., Karypis, G., Konstan J., &Riedl J. (2000). Application of dimensionality reduction in recommender system-a case study. WebKDD-2000 Workshop.

Sarwar, B., Konstan, J.A., Herlocker, J.L., Miller. B., & Riedl, J.T. (1998). Using filtering agents to improve prediction quality in the grouplens research, collaborative filtering system.

Schwab, I., Kobsa, A. and Koychev, I. (2001), Learning User Interests through Positive Examples Using Content Analysis and Collaborative Filtering. User Modeling and User-Adapted Interaction.

Shah et al., (2014) International Journal of Advanced Research in Computer Science and Software Engineering 4(11), November - 2014, pp. 367-371

Sielis, G., Tzanavari, A., & Papadopoulos, G. (2014) A Review of Recommender Systems: Types, Techniques and Applications. In book: Encyclopedia of Information Science and TechnologyEdition: 3rdChapter: 714.

Thi Do, M., Van Nguyen, D., & Nguyen, L. (2010) Model-based approach for collaborative filtering.

Vargas, S., & Castells, P., (2011). Rank and Relevance in Novelty and Diversity Metrics for Recommender Systems.

Ziegler, C., McNee, S.M., Konstan, J.A., & Lausen, G. (2005). Improving recommendation lists through topic diversification. Proceedings of the 14th International Conference on World Wide Web, pp. 22–32. ACM, New York

SITOGRAFIA

[1] “Big Data”. Accessibile da: *https://it.wikipedia.org/wiki/Big_data*

[2] “Le cinque V dei Big Data”. Accessibile da: *https://francescocugurra.wordpress.com/la-banca-liquida/le-cinque-v-dei-big-data/#_ftn1*

[3] “Data Mining”. Accessibile da: *https://it.wikipedia.org/wiki/Data_mining*

[4] “I pilastri dei big data: il data mining e il machine learning, 2018”. Accessibile da: *http://www.consultadibioetica.it/i-pilastri-dei-big-data-il-data-mining-e-il-machine-learning-seconda-parte-di-fiorello-casi/*

[5] “The Age of Tech”. Accessibile da: *https://www.statista.com/chart/5403/most-valuable-companies-2006-vs-2016/*

[6] “E-commerce worldwide- Staistics and Facts”. Accessibile da: *https://www.statista.com/topics/871/online-shopping/*

[7] “Netflix’s number”. Accessibile da: *https://www.businessinsider.com/netflix-movie-catalog-size-has-gone-down-since-2010-2018-2?IR=T*

[8] “75 Amazing Spotify statistics and facts (2019)”. Accessibile da: *https://expandedramblings.com/index.php/spotify-statistics/*

[9] “La coda lunga: da un mercato di massa a una massa di mercati”. Accessibile da: *http://www.corso-informatica-web.com/wp-content/uploads/2012/04/lez_30_03_2012.pdf*

[10] “Troppe possibilità di scelta, nessuna decisione: così il cervello va in sovraccarico”. Accessibile da: *https://www.focus.it/comportamento/psicologia/cervello-choice-overload-scelte*

[11] “How retailers can keep up with consumers”. Accessibile da: *https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers*

[12] “Recommender system for Learning”. Accessibile da *https://www.slideshare.net/Drachsler/recsystel-lecture-at-advanced-siks-course-nl*

[13] “Model based Recommendation system”. Accessibile da: *http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/modelbased.html*

[14] “Matrix Factorization Techniques For Recommender Systems”. Accessibile da: *https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/lehre/SS2011/Collaborative_Filtering/pres1-matrixfactorization.pdf*

[15] “Singular Value Decomposition”. Accessibile da: *https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75*

[16] “Cluster Analysis”. Accessibile da: *https://en.wikipedia.org/wiki/Cluster_analysis*

[17] “K Means Clustering”. Accessibile da: *https://towardsdatascience.com/k-means-clustering-identifying-f-r-i-e-n-d-s-in-the-world-of-strangers-695537505d*

[18] “La Cluster Analysis”. Accessibile da: *http://host.uniroma3.it/facolta/economia/db/materiali/insegnamenti/702_13195.pdf*

[19] “Cluster Analysis gerarchica”. Accessibile da: *http://www.federica.unina.it/economia/analisi-statistica-sociologica/cluster-analysis-gerarchica/*

[20] “Analisi Multivariata:Cluster Analysis”. Accessibile da: *http://www.rescoop.com/multivariata/AnalisiCluster.htm*

[21] “How to build a content-based movie recommender system with Natural Language Processing”. Accessibile da: *https://towardsdatascience.com/how-to-build-from-scratch-a-content-based-movie-recommender-with-natural-language-processing-25ad400eb243*

[22] “Recommender engine: TF-IDF”. Accessibile da: *https://towardsdatascience.com/recommender-engine-under-the-hood-7869d5eab072*

[23] “Text Processing with NLTK and Vector Space Models”. Accessibile da: *http://rmanak.github.io/notes/text-mining-nlp-python/*

1. Definizione del problema: long tail e choice overload

Nell'era della digitalizzazione, grazie alla diffusione del web sono stati abbattuti molti dei limiti strutturali che fisiologicamente caratterizzano da sempre i negozi fisici. Dal momento che i bisogni dei consumatori sono divenuti sempre più complessi, risulta necessario offrire una grande varietà di prodotti e servizi ad una ampia pluralità di nicchie. Anderson (2006) formalizza questa riflessione nella *Long Tail Theory*: la nostra economia si sta spostando gradualmente da quella che era una cultura dei prodotti *mainstream*, verso un'iper-segmentazione del mercato. (Figura 1.3)

Avendo adesso la possibilità di accesso ad una infinita varietà di prodotti, il consumatore è libero di soddisfare i suoi bisogni, se pur specifici, e sostanzialmente divergenti da esigenze più comuni alla maggioranza degli individui. Viene tuttavia a presentarsi la problematica opposta: nella ricerca della soluzione migliore, troppe possibilità di scelta possono indurre il consumatore a vivere la fase decisionale in modo oppressivo e spossante piuttosto che liberatorio, ricadendo così nel cosiddetto *choice overload*. I *Recommendation System* hanno proprio l'obiettivo di andare a risolvere questa tipologia di difficoltà. Essi sono dei sistemi automatici che, sfruttando tecniche di *data mining*, si occupano di assistere gli utenti nelle decisioni online, consigliando l'acquisto o la fruizione di quegli articoli che corrispondono ai loro interessi. Il seguente elaborato fornisce la descrizione di approcci e metodologie per l'implementazione di sistemi di raccomandazione. Il focus si indirizza sull'applicazione di motori di raccomandazione nel campo dei servizi video-streaming on demand (VOD), in particolare sono stati implementati algoritmi collaborativi su un *dataset* idoneo a scopi di ricerca e reperibile online.

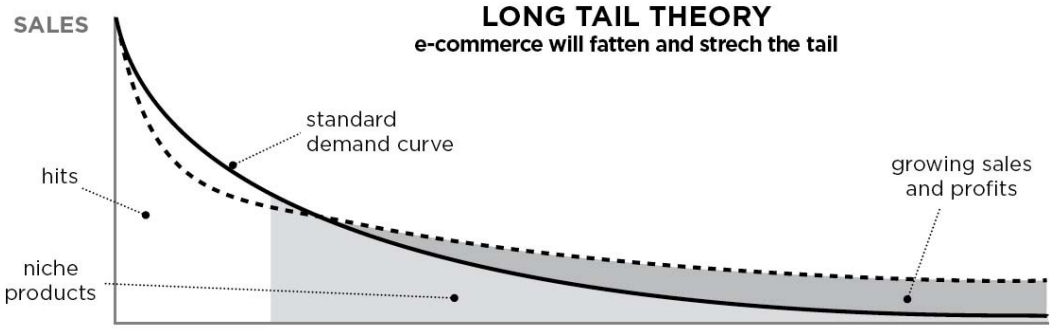

Figura 1.3. Long Tail Theory (Elbers, 2008)

2. Recommendation System: elementi essenziali

Qualsiasi tipologia di sistema di raccomandazione è costituito da 3 elementi essenziali: gli *item*, gli *user* e le *transactions.* (Ricci et al, 2011)

Gli *user* sono i protagonisti del recommandation system, i soggetti ai quali è indirizzato il consiglio di uno o più articoli, ma allo stesso tempo coloro che grazie alle loro caratteristiche (età, sesso, preferenze etc.) e comportamenti alimentano il *dataset* alla base della raccomandazione e quindi di tutto il sistema.

Gli *item* sono l'oggetto della raccomandazione da parte del sistema agli utenti. Essi sono definiti da un proprio set di attributi o caratteristiche che variano a seconda della tipologia dell'articolo (per un libro potrebbero essere l'autore, il genere l'anno di pubblicazione etc.). Ad essi l'utente assocerà una utilità soggettiva che dipende dai propri gusti e bisogni. Affinché il sistema sia in grado di comprendere gli interessi degli utenti, entrano in gioco le *transaction:* con esse possiamo intendere qualsiasi interazione user-computer che sia utile al fine di prevedere le preferenze del consumatore rispetto agli item. Queste ultime possono essere esplicite come l'invio volontario di un rating oppure implicite, ad esempio derivante dal *tracking* del comportamento online riguardo ricerche web o acquisti; in quest'ultimo caso gli interessi dell'utente sono dedotti dalle sue azioni

2.1 Benefici per l'utente

Lato consumatore, Herlocker et al. (2004) ci fornisce una classificazione dei principali vantaggi offerti dal sistema di raccomandazione:

- **Trovare item interessanti**: Il compito chiave dei sistemi di raccomandazione è certamente quello di raccomandare tra tutti i contenuti disponibili quelli che corrispondono agli interessi dell'utente. Ciò apporta vantaggi in termini di rilevanza degli articoli rispetto ai propri gusti, in termini di tempo per la ricerca dei contenuti più appropriati.
- **Migliorare il proprio profilo*:*** gli utenti contribuiscono a rilasciare i loro feedback con la finalità di delineare il loro profilo con maggiore precisione. In questo modo la loro interfaccia risulta personalizzata e di conseguenza anche la propria esperienza di navigazione. Al crescere delle valutazioni e quindi dei dati a disposizione aumenta anche la qualità delle raccomandazioni.

- **Esprimere sé stessi:** alcuni utenti, più che interessati all'accuratezza della raccomandazione, trovano giovamento nel fornire le proprie valutazioni semplicemente per il gusto di esprimere la propria opinione. Tale atteggiamento è stato riscontrato spesso sul sito *Amazon.com* dove gli utenti valutano i prodotti acquistati; questa tipologia di utenti è stata definita *"power user"*.
- **Aiutare gli altri:** in certi casi invece, gli utenti partecipano alle valutazioni con il solo intento di aiutare tutta la *community* di user a trovare i contenuti generalmente più interessanti. Essi provano soddisfazione nel fatto che altri utenti possano beneficiare del proprio contributo.
- **Intrattenimento:** anche se generalmente i sistemi di raccomandazione sono valutati sulla base di quanto essi aiutino gli utenti a prendere decisioni di consumo corrette, talvolta l'intento di questi ultimi è semplicemente quello di navigare sui siti senza un preciso bisogno imminente di acquisto ma al contrario con la sola volontà di intrattenimento. In questi casi piuttosto che l'accuratezza delle raccomandazioni risulta più importante l'interfaccia proposta, la facilità d'uso e la natura delle informazioni fornite.

2.2 Benefici per le aziende

F. Ricci et al., (2011) hanno descritto i principali vantaggi che le aziende digitali traggono dall'utilizzo dei sistemi di raccomandazione. Essi si traducono essenzialmente in un aumento complessivo delle vendite e la costruzione di una relazione solida con il consumatore. Di seguito la descrizione dei principali benefici:

- **Incremento del numero di articoli venduti:** dal momento che gli item sono consigliati al consumatore grazie ad un sistema di raccomandazione è più probabile che gli articoli proposti intercettino i bisogni e gli interessi del consumatore e che quindi vengano acquistati/fruiti.
- **Vendere articoli diversi:** anche prodotti/servizi di nicchia, particolarmente sconosciuti, grazie alla raccomandazione possono avere una visibilità maggiore e di conseguenza essere comprati.
- **Incrementare la soddisfazione dell'utente:** l'accuratezza delle raccomandazioni, combinata ad una interfaccia *easy to use* va a influenzare positivamente l'esperienza dell'utente all'interno del sito o applicazione.

- **Aumentare la loyalty:** quanto più un utente interagisce con il sito/app tanto più il suo profilo sarà ben definito e di conseguenza le raccomandazioni saranno più accurate, fino a *matchare* sempre di più le preferenze del cliente. Ciò alimenta un meccanismo a feedback positivo che spinge l'utente a non cambiare piattaforma/azienda ed essere *loyal*
- **Comprendere meglio i bisogni:** una volta raccolte le informazioni e i comportamenti del consumatore, il *provider* potrebbe decidere di utilizzare questa conoscenza per ottimizzare la gestione dello stock, la produzione, i messaggi promozionali etc.

2.3 Classificazione sintetica

I sistemi di raccomandazioni possono sfruttare diversi approcci attraverso cui è possibile consigliare i contenuti agli utenti. Isinkaye, Folajini e Ojokoh (2015) forniscono quella che è considerata la tradizionale tassonomia delle metodologie più diffuse e principalmente utilizzate (Figura 2.2).

La prima classificazione presenta tre macrocategorie che si differenziano per la logica che è alla base del sistema di raccomandazione:

- ***Collaborative-based filtering***: approccio più utilizzato, talvolta trattato come sinonimo di *Recommendation System*, è basato sulla raccolta di una grande quantità di informazioni su comportamenti, attività e gusti degli utenti. Esso è in grado di predire cosa piacerà agli utenti sulla base della somiglianza delle loro proferenze con quelle degli altri utenti.
- ***Content-based filtering*:** approccio che si fonda sulla descrizione delle caratteristiche degli oggetti da raccomandare e sulle preferenze degli utenti. Esso prova a raccomandare agli utenti gli articoli che hanno caratteristiche somiglianti a quelli che essi hanno già apprezzato in passato.
- ***Hybrid filtering***: metodologia che combina i due sistemi sopra citati.

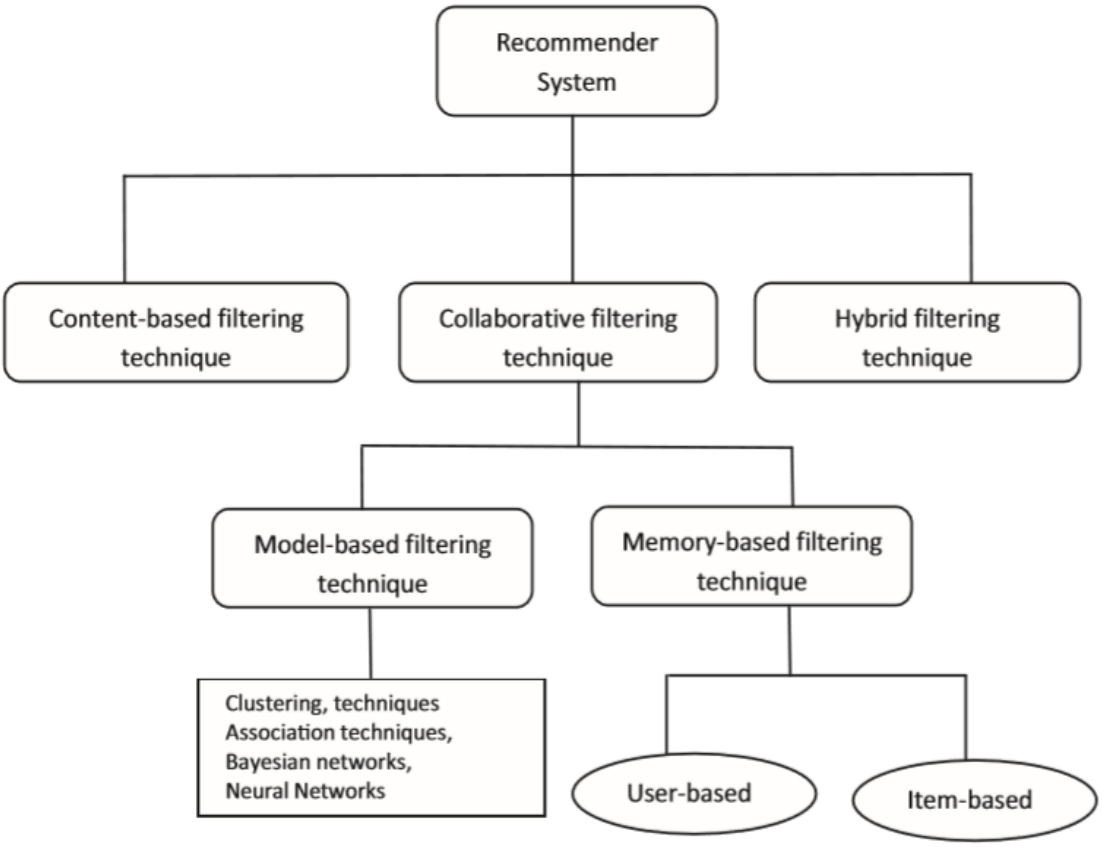

Figura 2.2. Classificazione sistemi di raccomandazione (Isinkaye et al., 2015)

Una seconda tipologia di classificazione riguarda il metodo di raccomandazione collaborativo: esso può essere suddiviso in una ulteriore sottocategoria differenziata a seconda del modo in cui è preso in considerazione il *dataset* e sulla tipologia di algoritmi utilizzati per analizzarlo:

- ***Memory-based***: Di questa categoria fanno parte gli algoritmi che, per predire valutazioni future su articoli non ancora incontrati, si basano sullo storico di tutti i rating (in memoria) degli utenti. Per questo scopo vengono utilizzati due approcci differenti: *User-based filtering* in cui vengono calcolate le similitudini tra le preferenze degli utenti e *Item-based filtering* che invece considera la somiglianza tra i rating dei contenuti.
- ***Model-based***: al contrario, le tecniche di raccomandazione *Model-based* non usano direttamente la storia del sistema per fare previsioni, ma piuttosto se ne avvalgono per apprendere un modello, che poi è utilizzato per generare raccomandazioni. In questa categoria rientrano sistemi che fanno uso di reti *bayesiane*, reti neurali, tecniche di *clustering*, regole di associazione.

3. Approcci ed algoritmi: Collaborative Filtering Memory-based

L'idea alla base dell'approccio collaborativo è sfruttare le informazioni sui comportamenti o opinioni passate di una esistente *community* di utenti per predire quale item è più probabile che interessi agli utenti correnti. Esso non utilizza alcuna informazione sul contenuto degli elementi da raccomandare ma ha come unico input una matrice *user-item* di rating. Il puro approccio collaborativo produce generalmente i seguenti tipi di output:

1. Una predizione numerica che indica il livello di interesse che gli utenti correnti potrebbero esprimere se incontrassero gli item.
2. Una lista di *n* item raccomandati.

È possibile utilizzare due logiche differenti: *user-based* e *item-based*

3.1 User-based e Item-based a confronto

Pur essendo entrambi sistemi *"collaborative"* e quindi basati sul calcolo delle somiglianze, la differenza principale tra l'approccio basato sugli utenti e quello fondato sugli item risiede negli elementi che sono alla base della computazione delle somiglianze e quindi del sistema di raccomandazione. L'approccio *item-based* parte dal calcolo delle similarità tra item e le utilizza per raccomandare i contenuti agli utenti. L'approccio *user-based* invece, parte dagli utenti e calcolando la somiglianza tra essi, fornisce raccomandazioni basate su cosa altri utenti con *pattern* di comportamento simili potrebbero apprezzare. (Figura 3.1)

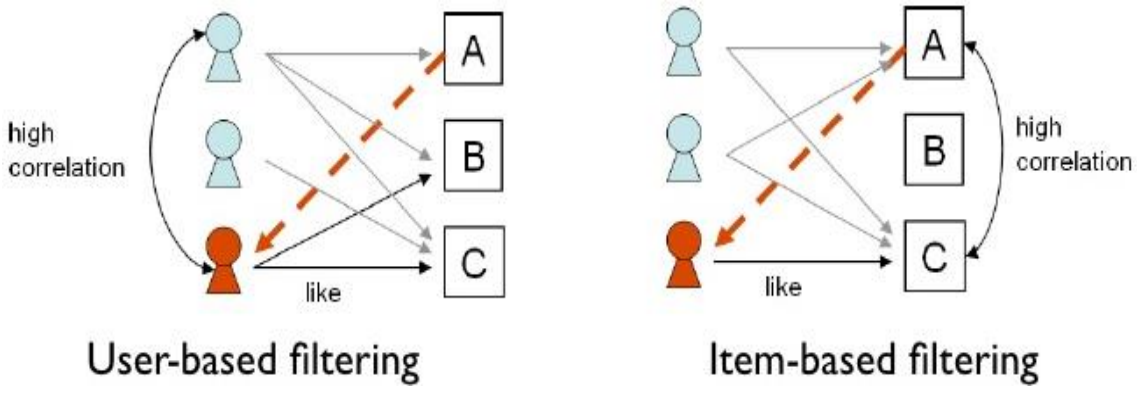

Figura 3.1. User-Based filtering vs Item-Based filtering [11]

Ne risulta che, pur avendo lo stesso obiettivo finale, il procedimento che porta alla stima della raccomandazione è differente. Ciò implica che i due approcci generano una serie di vantaggi e svantaggi differenti, dunque tra i due non esiste uno generalmente preferito all'altro, piuttosto vengono impiegati a seconda della situazione. Per questo è importante elencarne le ripercussioni sostanziali ai fini della raccomandazione (Ricci et al., 2011).

Accuratezza: l'accuratezza dei metodi di raccomandazione *collaborative* dipende in gran misura dal rapporto tra il numero di utenti e item all'interno del sistema. Nel caso in cui il numero di utenti è molto più grande del numero di item, come ad esempio capita in Amazon, l'approccio *item-based* produce generalmente raccomandazioni più accurate. Al contrario, sistemi che hanno un numero di utenti minore rispetto agli item, ad esempio siti che consigliano *paper* scientifici in cui ci sono milioni di utenti per centinaia di articoli, l'approccio *user-based* è tendenzialmente migliore.

Efficienza: l'efficienza computazionale del sistema di raccomandazione e la memoria necessaria al *processing* dipende anch'essa dal rapporto utenti/item. Quando il numero di utenti supera il numero di item, il metodo *item-based* richiede meno memoria e tempo di computazione, rendendolo di conseguenza più scalabile.

Stabilità: la scelta tra i due approcci dipende anche dalla frequenza di aggiornamenti e quindi cambiamenti negli utenti e nei contenuti all'interno del sistema. Se l'elenco degli item disponibili tende ad essere più statico rispetto alla lista degli utenti correnti, il metodo *item-based* è preferibile dal momento che è possibile effettuare raccomandazioni ai nuovi utenti senza la necessità di calcolare nuovamente le similarità tra i contenuti. Viceversa, nelle applicazioni in cui la lista degli item è costantemente in aggiornamento, l'approccio *user-based* risulta più stabile.

Giustificazione: un vantaggio dei metodi basati sugli item è certamente attribuibile alla semplice giustificazione della raccomandazione agli utenti. Infatti, può essere presentata all'utente come motivazione una lista degli item *neighbor* utilizzati nella predizione e il loro grado di similarità con il contenuto raccomandato. Inoltre, dando la possibilità all'utente di modificare la lista, lo si può rendere partecipe al processo in modo interattivo. La giustificazione non può essere fornita con i metodi *user-based*, in quanto gli utenti nuovi generalmente non conoscono gli user simili utilizzati per la predizione e quindi psicologicamente potrebbero non comprendere le motivazioni della raccomandazione.

Serendipity: Nell'approccio *item-based*, il rating stimato che un utente potrebbe fornire ad un item è basato sulle valutazioni effettuate per item simili. Di conseguenza, il sistema di raccomandazione che utilizza questo metodo tende a consigliare agli utenti contenuti simili a quelli già apprezzati in precedenza. Da un lato, ciò tende a generare raccomandazioni più sicure, dall'altro non concede agli utenti la possibilità di scoprire tipologie di contenuti nuove, per cui potrebbe provare un interesse forte ma latente. Al contrario, l'approccio basato sugli utenti, dal momento che origina dal calcolo della similarità tra gli utenti, è più probabile che generi raccomandazioni originali e sorprendenti.

4. Analisi: implementazione di sistemi di raccomandazione per servizi Video Streaming on Demand con R studio

Lo scopo della seguente analisi sarà fornire empiricamente la metodologia con cui, a partire da un determinato *dataset*, è possibile costruire operativamente un sistema di raccomandazione. Dopo aver costruito i motori di raccomandazione ne saranno discussi i risultati ed evidenziati le peculiarità non che i limiti dell'uno e dell'altro, con l'obiettivo di verificare se essi risultino o meno coerenti con lo stato dell'arte.

Il campo di applicazione è circoscritto al settore cinematografico.

Con il diffondersi del web, la produzione di film, serie tv, cortometraggi o anche semplicemente materiali video creati dagli utenti, hanno avuto una crescita esponenziale e le prospettive future sembrano andare ancora in questa direzione. Inoltre, la modalità di visualizzazione dei contenuti ha avuto un radicale cambiamento negli anni; grazie alla diffusione di servizi di video-streaming on demand (VOD), quali Netflix, Amazon Prime, Plus and Hulu, gli utenti tendono a privilegiare sempre di più questi ultimi alle modalità tradizionali di visualizzazione. Il beneficio di poter decidere autonomamente quando vedere un film oppure una serie tv, avendo la possibilità di guardarla tutta di un fiato è mitigato però dalla enorme grandezza del catalogo di titoli a disposizione. La vastità dei titoli disponibili induce nell'utente una sensazione di incertezza che ricade talvolta in un freno all'utilizzo delle piattaforme. Motivo per cui risulta cruciale avere un sistema di raccomandazione che scelga per noi, suggerendo o mettendo in primo piano i film coerenti con i propri interessi; è infatti questo uno dei motivi del successo delle piattaforme Streaming Video On Demand.

Nella seguente analisi si andrà a simulare la situazione appena descritta. Compito del sistema di raccomandazione implementato, sarà fornire una *recommendation list* per ogni utente, in cui gli item da suggerire saranno titoli di film/serie tv non ancora visionati e per i quali gli utenti potrebbero provare interesse.

4.1 Origine e contenuto del dataset *"Movielens"*

Il *dataset* oggetto di analisi si chiama*"Movielens"*, è tratto dalle interazioni tra la *community* di utenti ed il sito web *www.movielens.org* ed è stato messo a disposizione per scopi di ricerca da *Grouplens,* laboratorio di ricerca nel Dipartimento di *Computer Science and Engineering* dell'Università del Minnesota, "Twin Cities". Una volta registrati, agli utenti sono stati proposti casualmente un determinato numero di film e gli è stato chiesto di esprimere una propria valutazione da 0 a 5 per un numero minimo di 15 titoli, tali da poter iniziare a costruire un profilo utente attendibile nella rappresentazione delle sue preferenze

Esso contiene nel dettaglio:

- 100.836 rating
- 9.742 film
- 610 utenti
- 3.683 tag

Questi dati sono stati raccolti tra il 29 Marzo del 1996 e il 24 Settembre del 2018, ed il *dataset* è stato generato il 26 Settembre del 2018.

Gli utenti sono stati selezionati in modo casuale tra quelli che avevano valutato almeno 20 titoli. Sono state inoltre rimosse tutte le informazioni demografiche, infatti, gli utenti non sono rappresentati da nessun'altra informazione eccetto un ID identificativo.

4.2 Implementazione UBCF e IBCF

Il *core* dell'analisi risiede nell'implementatazione di recommandation system basati su un approccio collaborativo, in particolare verranno messi a confronto due algoritmi, lo *User-based Collaborative Filtering* (UBCF) e l'*Item-based collaborative filtering* (IBCF). Essi dipendono dalla scelta di 3 parametri principali, che ne condizionano il funzionamento e di conseguenza i risultati finali: 1) *K-nearest Neighbor* (numero di utenti da tenere in considerazione nel calcolo delle similarità tra utenti o item); 2)

Metodologia di normalizzione dei dati (*Zero-mean centering o Z-score*); 3)Misura di similarità (Pearson, *Cosine*, Jaccard).

Primo step consiste nell'ottimizzazione dei suddetti parametri. Per entrambi gli approcci sono stati implementati e messi a confronto 30 modelli ciascuno, derivanti dall'incrocio delle 2 variabili di normalizzazione, 3 misure di similarità e 5 diversi valori di *neighbor*.

Una volta selezionati i due modelli *User-based* e *Item-based* più accurati, è stata effettuata una comparazione tra essi con la conseguente scelta dell'approccio più adatto per la base dati oggetto di analisi. Oltre ai due modelli sopracitati, è stato inserito anche un ulteriore modello *benchmark*, ovvero un sistema di raccomandazione *"Random"*. Esso andrà a simulare un recommendation system completamente casuale, che raccomanda gli item in modo *random* senza l'implementazione di alcun algoritmo. In sintesi, sarà illustrato il confronto tra i seguenti modelli:

- **UBCF** con Neighbor 65, distanza di Pearson e metodo di normalizzazione *Z-score*
- **IBCF** con Neighbor 45, distanza di Pearson e metodo di normalizzazione *Z-score*
- Recommendation system "***Random***", utile come *benchmark*, che andrà a simulare raccomandazioni casuali in assenza di alcun algoritmo implementato

Una prima importante osservazione riguarda la differente efficienza dei modelli in termini di velocità (Figura 4.24). Il sistema di raccomandazione *User-Based* ha impiegato 2 centesimi di secondo per l'implementazione del modello ed un quarto di secondo per effettuare le predizioni. Al contrario il sistema *Item-based* ha impiegato 2,73 secondi per la prima operazione, risultando 137 volte più lento, e 0,04 secondi per stimare le predizioni. Il sistema Random, non avendo alcun modello sottostante ha impiegato semplicemente 0,06 secondi per calcolare i *predicted rating*.

```
UBCF run fold/sample [model time/prediction time]
                1  [0.02sec/0.25sec]

IBCF run fold/sample [model time/prediction time]
                1  [2.73sec/0.04sec]

RANDOM run fold/sample [model time/prediction time]
                1  [0sec/0.06sec]
```

Figura 4.24. Velocità Recommendation system implementati

Plottando i risultati dei modelli sulla Curva ROC, è possibile comprendere la differente efficacia dei modelli in termini qualitativi, sfruttando come soglie arbitrarie differenti *recommendation list* che, si distinguono per la quantità di item da raccomandare. Dal grafico sottostante (Figura 4.25), il modello *User-Based* risulta avere una prestazione nettamente superiore rispetto a quello *Item-Based* che si presenta particolarmente deludente. Quest'ultimo infatti, oltre a fornire raccomandazioni inaccurate è praticamente assimilabile al sistema *Random* che è stato sviluppato senza l'ausilio di alcun algoritmo.

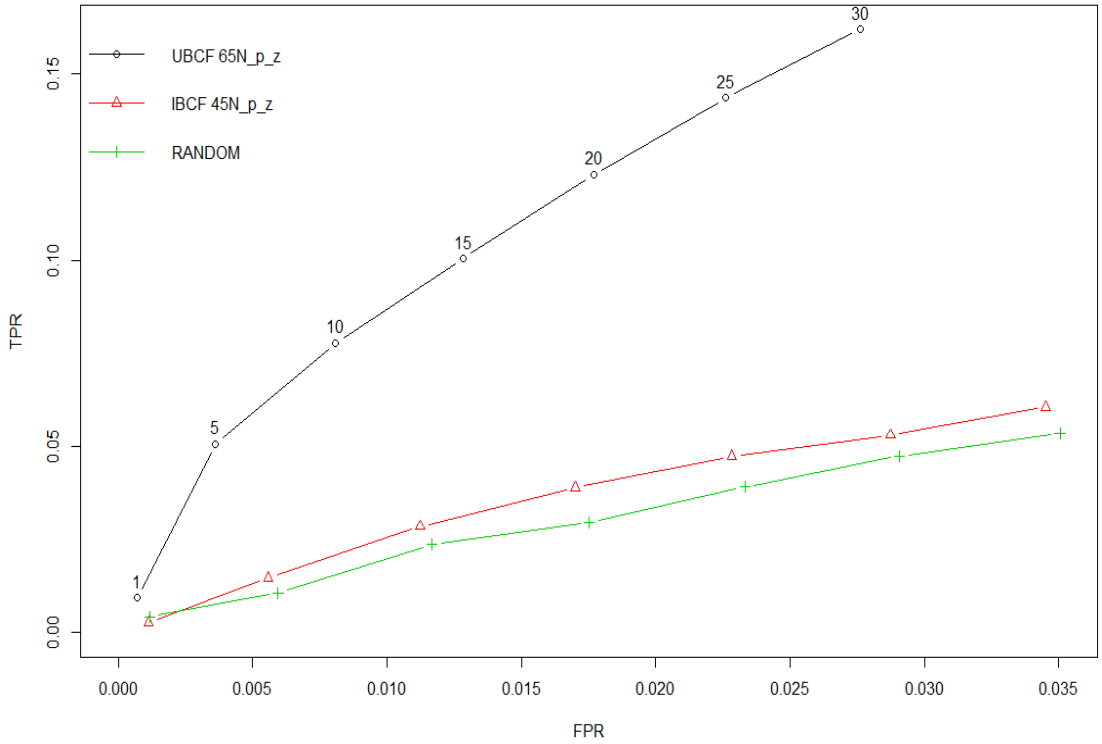

Figura 4.25. ROC Curve UBCF vs IBCF vs Random

Gli indicatori di errore (Figura 4.26) confermano in termini quantitativi quanto visto graficamente dalla ROC. Il sistema di raccomandazione *Item-Based* presenta un errore medio sulle predizioni molto alto, avvicinandosi addirittura ai risultati del modello casuale, motivo per cui non si dimostra funzionale ai fini dell'analisi.

La prestazione del modello *User-Based* con *Neighbor* 65, distanza di Pearson e metodo di normalizzazione *Z-score* risulta essere senza dubbio la migliore tra tutti i modelli sperimentati (30 *item-based* e 30 *user-based*). Il margine di errore infatti è stato ridotto notevolmente fino a raggiungere un **MAE = 0,68.**

La scelta finale del modello da utilizzare al fine di raccomandare una lista di film per ogni utente presente nel *dataset* oggetto di analisi ricade dunque su quest'ultimo che sarà quindi utilizzato come riferimento per l'estrazione e la visualizzazione delle *recommendation list*.

	RMSE	MSE	MAE
valut_RANDOM	1.2565642	1.5789536	0.9632421
valut_UBCF_65N_p_z	**0.9013977**	**0.8125177**	**0.6800774**
valut_IBCF_45N_p_z	1.2486364	1.5590928	0.9017524

Figura 4.26. Misure di errore UBCF vs IBCF vs Random

4.3 Recommendation list: estrazione e visualizzazione

Dopo aver dedicato la sezione precedente alla scoperta del modello più adatto ai fini dell'analisi, quest'ultimo paragrafo si focalizzerà sulla presentazione dell'*output* finale. L'obiettivo ultimo infatti, risiede nell'estrazione e la seguente visualizzazione di una lista di 15 film da raccomandare a ciascun utente presente nel *test set.*

Risulta interessante approfondire le caratteristiche delle *recommendation list* nella loro totalità. Nel complesso sono stati suggeriti 126 film sui 171 disponibili nel *test set*. Il grafico sottostante (Figura 4.28) descrive la distribuzione dei film all'interno delle liste; sull'asse delle ascisse troviamo il numero di volte che un film è stato suggerito e sull'asse delle ordinate la quantità di film che corrispondono a tale valore. Ad esempio, un valore dell'ascissa = 89 e di ordinata = 1 vorrà dire che un solo film è stato raccomandato 89 volte. Di conseguenza nella coda della distribuzione ritroviamo quei film che sono stati raccomandati un numero elevato di volte e praticamente quasi sempre presenti in tutte le liste di raccomandazione, viceversa, nella testa ritroviamo quei film che sono stati suggeriti in un numero di occasioni sostanzialmente inferiore.

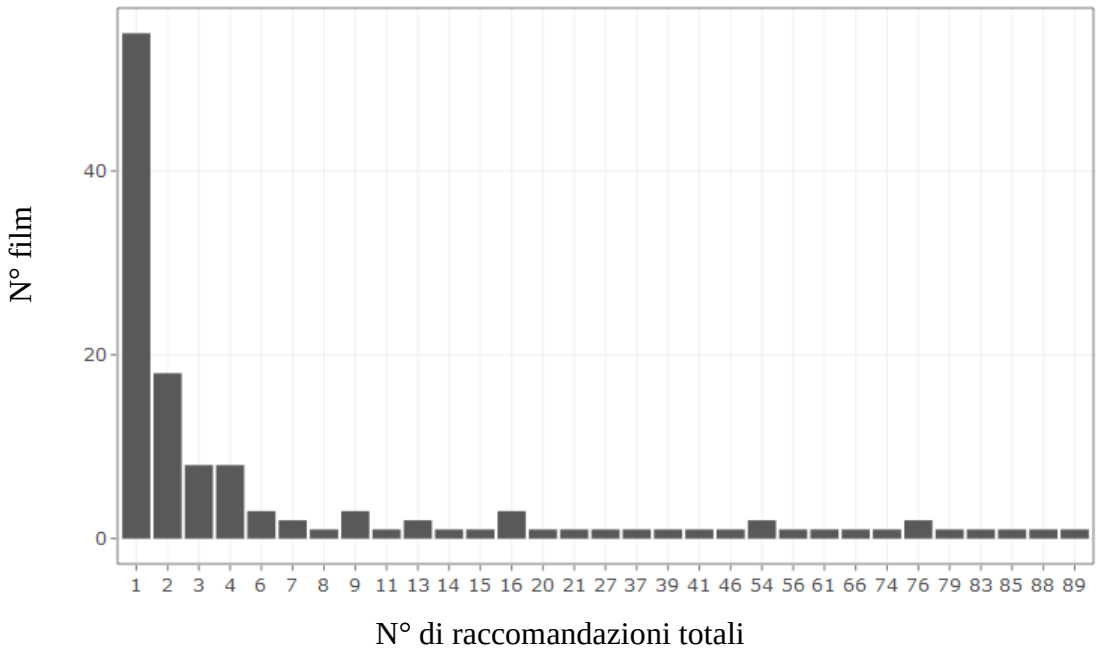

Figura 4.28. Distribuzione film Recommendation List

È proprio su questi ultimi che è importante focalizzare l'attenzione, in quanto, rispetto ai film più gettonati presenti nella coda che stravincono in termini di valutazioni medie e popolarità, essi risultano essere rari. La loro raccomandazione dunque, non essendo scontata rispecchia le caratteristiche di originalità e serendipità fondamentali per dare una utilità ed un valore maggiore ad un sistema di raccomandazione.

Fornendo la *recommendation list* dell'utente 1 (Figura 4.29) è possibile visualizzare tutti i titoli che il sistema di raccomandazione ha raccomandato. Inoltre, è bene evidenziare quei film sono stati raccomandati pur non essendo estremamente popolari all'interno del *dataset* o non avendo una valutazione media palesemente positiva. Essi quindi, sono frutto di una raccomandazione personalizzata, sviluppata sulla base del comportamento di utenti simili grazie all'ausilio dell'algoritmo collaborativo *User-based*.

Recommendation list utente 1
Shawshank Redemption, The (1994)
Forrest Gump (1994)
Star Wars: Episode V - The Empire Strikes Back (1980)
Pulp Fiction (1994)
Star Wars: Episode IV - A New Hope (1977)
Godfather, The (1972)
Monty Python and the Holy Grail (1975)
Matrix, The (1999)
American Beauty (1999)
Silence of the Lambs, The (1991)
Casablanca (1942)
Schindler's List (1993)
Shrek (2001)
Usual Suspects, The (1995)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)

Figura 4.29. Recommendation List evidenziata

Conclusioni

Il lavoro di tesi ha contribuito *in primis*, a fornire una spiegazione teorica sull'origine, la rilevanza e la classificazione dei sistemi di raccomandazione, andando ad approfondire i differenti algoritmi alla base del loro funzionamento. In secondo luogo, ha descritto la metodologia con cui, a partire da un determinato *dataset*, è possibile costruire empiricamente un vero e proprio sistema di raccomandazione; in particolare, l'elaborato si è concentrato sull'applicazione di questi ultimi al mondo dei servizi video-streaming on demand (VOD). Esso è uno dei settori in cui i recommendation system, a causa dell'enorme vastità dei cataloghi e la conseguente difficoltà di scelta da parte dell'utente, risultano una risorsa chiave per offrire all'utente una piattaforma che suggerisca o metta in primo piano soltanto quei titoli coerenti con i propri interessi, garantendo così un'esperienza di navigazione funzionale e piacevole.

Il cuore dell'analisi si è focalizzato sull'implementazione e sul confronto dei due algoritmi più celebri in letteratura: lo *User-Based Collaborative Filtering* (UBCF) e l'*Item-Based Collavorative Filtering* (IBCF). Prima dello sviluppo e la seguente

comparazione dei risultati ottenuti dai due approcci, è stata eseguita, per entrambi, una prima fase di ottimizzazione dei parametri finalizzata alla massimizzazione delle prestazioni dei due modelli di riferimento. Ambedue gli algoritmi hanno dimostrato di fornire prestazioni migliori con l'utilizzo del metodo *Z-score* per quanto riguarda la normalizzazione dei dati e della distanza di Pearson per il calcolo delle similarità tra utenti e item.

Una volta selezionati i due modelli UBCF e IBCF ottimizzati si è proseguito con il confronto dei risultati e la conseguente scelta del modello di riferimento per l'output finale. Dalla comparazione è emerso che l'algoritmo *Item-Based* risulta particolarmente inadatto alla tipologia di *dataset* oggetto di analisi. Verificando l'accuratezza delle sue raccomandazioni è risultato che esse sono equiparabili a raccomandazioni *random*, sviluppate in modo casuale e senza l'ausilio di alcun algoritmo. Al contrario, L'approccio *User-based* ottimizzato, ha fornito una prestazione soddisfacente sia in termini di accuratezza (MAE = 0,68), che in termini qualitativi. Infatti, le *recommendation list* ottenute da esso, sono costituite non solo da suggerimenti prevedibili di film celebri e comunemente apprezzati, bensì anche titoli più rari, frutto di suggerimenti personalizzati ritagliati su misura per gli utenti, che difficilmente sarebbero stati suggeriti in assenza di un sistema di raccomandazione.

L'analisi empirica si è dunque focalizzata su tecniche appartenenti all'approccio collaborativo che per lo sviluppo delle raccomandazioni prendono in considerazione i comportamenti degli utenti, ma non sfruttano alcuna informazione riguardante le caratteristiche descrittive dei titoli. In ottica futura sarebbe interessante estrarre informazioni rilevanti per ciascun titolo presente nel *dataset* (regista, trama, attori, argomenti trattati, anno di rilascio etc.), condurre uno studio che si concentri su tecniche *Content-based* ed infine metterne a confronto il livello di accuratezza e la qualità delle differenti liste di raccomandazione generate.