UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica Musicale

Mooncloud recommendation system

Relatore:
Claudio Agostino Ardagna
Correlatore:
Valerio Bellandi

Tesi di Laurea di:
Andrea Michele Albonico
Matricola: 886667

Anno Accademico 2018/2019

Ringraziamenti

Andrea Michele Albonico

Prefazione

I sistemi di raccomandazione (*Recommendation System*) hanno avuto un forte sviluppo negli ultimi decenni e nascono proprio con lo scopo di identificare quegli oggetti (detti generalmente *item*) all'interno di un vasto mondo di informazioni che possono essere di nostro interesse e tanto maggiore è il grado di conoscenza dell'individuo e tanto più vengono ritenuti affidabili.
Il motivo di questo successo risiede nella riuscita integrazione di tali sistemi in applicazioni commerciali, soprattutto nel mondo dell'E-commerce e nel fatto che sono in grado di aiutare un utente a prendere una decisione, che sia la scelta di un film per l'uscita con gli amici il sabato sera, di una playlist da ascoltare durante un viaggio in auto o in un momento di lettura, e via discorrendo.
Moon Cloud è una piattaforma erogata come servizio che fornisce un meccanismo di *Security Governance* centralizzato. Garantisce il controllo della sicurezza informatica in modo semplice e intuitivo, attraverso attività di test e monitoraggio periodiche e programmate (*Security Assurance*). L'obbiettivo di questa tesi è stato quello di aggiungere, al già presente sistema per la scelta dei controlli all'interno delle attività di test, un sistema di raccomandazione che possa consigliare all'utente delle possibili *Evaluation* rispetto al target indicato; in questo modo anche l'utente meno esperto può usufruire dei servizi offerti da Moon Cloud in modo semplice e intuitivo.

La tesi è organizzata come segue:

Capitolo 1 – Introduzione a Moon Cloud in questo capitolo viene descritta la piattaforma Moon Cloud e il suo funzionamento in ambito di Security Assurance.

Capitolo 2 – Tecnologie utilizzate in questo capitolo vengono presentati gli studi e le analisi di soluzioni esistenti, studi delle tecnologie utilizzate per la realizzazione del progetto.

Capitolo 3 – Recommendation systems in questo capitolo viene descritto in modo più approfondita degli studi compiuti che hanno portato al-

la realizzazione dei sistemi di raccomandaizone proposti nella soluzione implementata per la piattaforma Moon Cloud.

Capitolo 4 – Descrizione della soluzione in questo capitolo viene descritta in maniera dettagliata la realizzazione dell'applicativo, analizzando quali sono state le difficoltà maggiori, i risultati ottenuti e l'uso che se ne è fatto.

Capitolo 5 – Conclusioni in questo capitolo vengono esposte le conclusioni e i possibili sviluppi futuri delle attività svolte e del sistema realizzato.

Indice

Elenco delle figure

Capitolo 1

Introduzione a Moon Cloud

In questo capitolo verrà descritto in modo più approfondito il funzionamento della piattaforma Moon Cloud e unitamente al motivo dell'implementazione della soluzione proposta.

1.1 Moon Cloud overview

La diffusione di sistemi *Information and Communications Technology* (acronimo ICT) ha avuto luogo nella maggiorparte degli ambienti lavorativi e privati in termini di servizi offerti, automazione di processi e incremento delle performance. L'uso di questa tecnologia ha assunto importanza a partire dagli anni novanta come effetto del boom di Internet e al giorno d'oggi le professionalità legate al mondo dell'ICT crescono in numero e si evolvono per specificità, per operare in ambienti fortemente eterogenei ma sempre più interconnessi fra di loro come il Cloud Computing, i Social Newtwork, il Marketing Digitale, i Sistemi IoT, la Realtà Virtuale, etc.

Il Cloud Computing ha portato un rivoluzionario paradigma nella creazione di un nuovo business virtuale accessibile in qualunque momento e in qualunque luogo; esso sfrutta le tecnologie messe a disposizione dai sistemi ICT come le operazioni di virtualized computing, internet e distributed computing, provvedendo un sistema integrato molto potente. Google, Microsoft, Amazon sono un esempio di aziende che forniscono servizi di Cloud Computing in business ICT. Si può definire il Cloud Computing come l'abilità di accedere a risorse (come database o applicazioni) in tutto il mondo attraverso una rete in poco tempo.

Gli immensi benefici del Cloud in termini di flessibilità, consumo delle risorse e gestione semplificata, la rende la prima scelta per utenti e industrie per il deploy dei loro sistemi IT. Tuttavia il Cloud Computing solleva diverse

problematiche legate alla mancanza di fiducia e trasparenza dove i clienti necessitano di avere delle garanzie sui servizi Cloud ai quali si affidano; spesso i fornitori di questi servizi non fornisco ai clienti le specifiche riguardanti le misure di sicurezza messe in atto.

Negli ultimi anni, sono state sviluppate tecniche e modi per rendere sicuri questi sistemi e proteggere i dati degli utenti, portando alla diffusione di approcci eterogenei che incrementaro la confusione negli utenti. Tecniche tradizionali di verifica della sicurezza basati su metodi di analisi statistica non sono più sufficenti e devono essere integrati con processi di raccolta di prove (in inglese evidence) da sistemi Cloud in produzione e funzionanti. In generale il *Cloud Security* definisce i modi, come criptazione e controllo degli accessi, per proteggere attivamente gli asset da minaccie interne ed esterne, e fornire un ambiente in cui i clienti possano affidarsi e interagire in totale sicurezza. Tutto questo non basta a rendere il Cloud degno di fiducia e trasparente, per questo sono state introdotte tecniche di *Security Assurance*, delle garanzie che permettono di ottenere la fiducia necessaria nelle infrastrutture e/o nelle applicazioni di dimostrare il rispetto di certe proprietà di sicurezza, e che operino normalmente anche se subiscono attacchi; grazie alla raccolta e allo studio di evidence è possibile che venga accertata la validità e l'efficenza delle proprietà di sicurezza messe in atto.

Il prezzo che si paga per i benefici di questa tecnologia è dato dall'incremento di violazioni di sicurezza, che oggigiorno preoccupa tutte le aziende e di conseguenza anche i loro clienti, con l'incremento del rischio di fallimento per i servizi più importanti dovuti a violazioni della privacy e al furto di dati.
Il mercato sta lentamente notando che non è l'inadequamento tecnologico dei sistemi di sicurezza che incrementa il rischio delle violazioni di sicurezza; piuttosto, la mal configurazione e l'errata integrazione di questi sistemi nei processi di business. [2] L'utilizzo di sistemi di sicurezza e di controllo migliori non garantisce in modo assoluto la sicurezza dell'infrastruttura; per garantire ciò è necessario implementare un processo continuo di diagnostica e verifica della corretta configurazione dei controlli, supervisionando il loro comportamento, accertandosi che sia quello aspettato.

Il *Security Assessment* diventa allora un aspetto importante specialmente negli ambienti Cloud e IoT. Questo proccesso, costituito da un insieme di attività mirate alla valutazione del rischio in sistemi IT, deve essere portato avanti in modo continuo e olistico, per correlare le evidence raccolte da sempre maggiori meccanismi di protezione. [1]
Moon Cloud è una soluzione PaaS (acronimo inglese di Platform as a Service) che fornice una piattaforma B2B (Business To Business) innovativa per verifiche, diagnostiche e monitoraggio dell'adeguatezza dei sistemi ICT rispetto

alle politiche di sicurezza, in modo continuo e su larga scala. Moon Cloud supporta una semplice ed efficente *ICT Security Governance*, dove le politiche di sicurezza possono essere definite dalle compagnie stesse (a partire da un semplice controllo sulle vulnerabilità a linee guida di sicurezza interna), da entità esterne, imposte da standard oppure da regolamentazioni nazionali o internazionali.
La sicurezza di un sistema o di un insieme di asset dipende solo parzialmente dalla forza dei singoli meccanismi di protezione isolati l'uno dall'altro; infatti, dipende anche dall'abilità di questi meccanismi di lavorare continuamente in sinergia per provvedere una protezione olistica. In più, quando i sistemi Cloud e i servizi IoT sono coinvolti, le dinamiche di questi servizi e la loro rapida evoluzione rende il controllo dei processi all'interno dell'azienda e le politiche di sicurezza più complesse e prone ad errori.
I requisiti ad alto livello fondamentali per poter garantire le Security Assurance sono:

sistema olistico è richiesta una visione globale e pulita dello status dei sistemi di sicurezza; inoltre è cruciale distribuire lo sforzo degli specialisti in sicurezza per migliorare il processo e le politiche messe in atto. Si parte da delle valutazioni fatte manualmente a quella semi-automatiche che vengono usate per ispezionare i meccanismi di sicurezza.

monitoraggio continuo ed efficiente è necessario un controllo continuo che valuti l'efficenza dei sistemi di sicurezza per ridurre l'impatto dell'errore umano, soprattutto dal punto di vista organizzativo. La coesistenza di componenenti in conflitto o la mancata configurazione dovuta al cambiamento dell'ambiente possono essere scenari che richiedono un monitoraggio e un aggiornamento continuo.

singolo punto di management avere un solo punto d'accesso in cui poter gestire tutti gli aspetti relativi alla sicurezza, permette di avere sotto controllo le politiche di sicurezza. Inoltre disporre di un inventario degli asset da proteggere permette di poter conoscere quali meccanismi di protezioni applicare.

reazioni rapide a incidenti di sicurezza spesso la reazione ad incidenti di sicurezza è ritardata da due fattori: il tempo richiesto per rilevare l'incidente e il tempo per analizzare il motivo dell'accaduto; e avere un sistema che implementa un monitoraggio continuo permette di venire a conoscenza di questi problemi in breve tempo e agire di conseguenza.

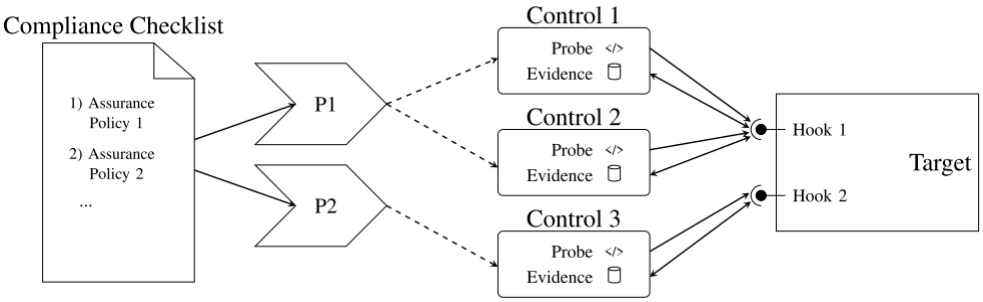

Figura 1.1: Security Compliance Evaluation

Moon Cloud è basato su una tecnica di Security Assurance garantendo che tutte le attività aziendali si compiano seguendo i requisiti prestabiliti da appropriate politiche e procedure precedentemente definite.

Viene definita una *Security Compliance Evaluation*, un processo di verifica a cui un target viene sottoposto e il cui risultato deve soddisfare i requisiti richiesti da standard e politiche. A partire da questi processi di controllo, che devono a loro volta essere affidabili, si ottendo delle evidence; queste ultime possono essere raccolte monitorando l'attività del target oppure, come già menzionato, sottoponendo il target a scenari critici o di testing. In particolare, una Security Compliance Evaluation è un processo di verifica dell'uniformità di un certo target a una o più politiche attraverso una serie di controlli che a seconda del valore booleano (successo pari a 1 o insuccesso pari a 0) associato ad ogni controllo viene prodotto un valore booleano per le politiche; se un target supera tutti i controlli, quindi tutti i valori booleani associati ad ogni controllo è pari al successo, a cui è sottoposto allora significa che rispetta la politica scelta.

1.2 Processo di Evaluation

Moon Cloud implementa il processo di Security Compliance Evaluation in Figura 1.1 usando controlli di monitoraggio o di test personalizzabili. Inoltre garantisce, oltre a tutti i requisisti ad alto livello elencati precedentemente 1.1, anche i seguenti:

Moon Cloud è una piattaforma Cloud centralizzata presentando una visione olistica dello stato di sicurezza di un dato sistema.

Moon Cloud implementa un sistema di Security Assurance Evidence-based continuo, implementato come processo di Compliance, basato su politiche custom o standard.

Moon Cloud è offerto come un servizio (PaaS), dove le attività di Evaluation possono essere facilmente ed efficentemente configurate su un target asset, senza l'intervento dell'uomo.

Moon Cloud permette di schedulare delle ispezioni automatiche, grazie all'inventario di asset protetto.

Moon Cloud Evaluation Engine può ispezionare dall'interno un sistema, gestendo così delle minaccie interne; permettendo anche reazioni rapide a incidenti di sicurezza e veloci rimedi, grazie alla raccolta continua di evidence.

In generale, l'architettura di Moon Cloud è costituita da un'Assurance Manager che gestisce i processi di Evaluation attraverso un set di *Execution Cluster*; ognuno dei quali gestisce ed esegue un set di probe che collezionano le evidence necessarie per effettuare i processi di valutazione. Tutte le attività di collezione sono eseguite dal probe, ognuno dei quali è uno script Python fornito come una singola immagine di Docker, che viene inizializzata quando viene triggerata una Evaluation ed è distrutta quando il processo di Evaluation è terminato.
Accedendo alla piattaforma di Moon Cloud, l'utente può definire le proprie politiche di sicurezza e attività di Evaluation come espressioni booleane di controlli di sicurezza e altre politiche predefinite. Una volta che una politica viene definita, l'utente può decidere quando schedulare l'Evaluation; e nel momento in cui un processo di Evaluation viene inizializzato, tutti i controlli vengono eseguiti e i risultati dell'espressioni booleane vengono memorizzati e restituiti all'utente. A questo punto l'utente può accedere a questi risultati a diversi gradi di precisione: una visione sommaria e generale di tutte le politche implentate e dello stato generale del sistema di sicurezza, al risultato di una specifica politica oppure alle evidence raccolte per una Evaluation.

Per poter rendere ancora più intuituivo e semplice da utilizzare un sistema di questa importanza, si è pensato di introdurre un sistema che possa raccomandare agli utenti, in base agli asset che vuole proteggere e monitorare, una serie di Evaluation o politihe da applicare in quei casi; questo permette anche a utenti meno esperti di poter configurare in modo rapido ed efficente dei meccanismi di protezione da minaccie.

Capitolo 2

Tecnologie utilizzate

In questo capitolo verranno descritte le attività preliminari per la realizzazione di questo progetto, le tecnologie utilizzate unitamente alle motivazioni legate all'uso di questi sistemi rispetto ad altri.

2.1 Perché Python e Django

Python è un linguaggio di programmazione ad alto livello, orientato agli oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing; ideato e rilasciato pubblicamente per la prima volta nel 1991 dal suo creatore Guido van Rossum, programmatore olandese.

Python supporta diversi paradigmi di programmazione, come quello object-oriented (con supporto all'ereditarietà multipla), quello imperativo e quello funzionale, ed offre una tipizzazione dinamica forte. È fornito di una libreria built-in estremamente ricca, che unitamente alla gestione automatica della memoria e a robusti costrutti per la gestione delle eccezioni fa di Python uno dei linguaggi più ricchi e comodi da usare.

Inoltre è anche semplice da usare e imparare. Python, nelle intenzioni di Guido van Rossum, è nato per essere un linguaggio immediatamente intuibile. La sua sintassi è pulita e snella così come i suoi costrutti, decisamente chiari e non ambigui. I blocchi logici vengono costruiti semplicemente allineando le righe allo stesso modo, incrementando la leggibilità e l'uniformità del codice anche se vi lavorano diversi autori.

Un aspetto inusuale del Python è il metodo che usa per delimitare i blocchi di programma, che lo rende unico fra i linguaggi più diffusi.

```
1 # Testing if two strings are equals
2 def test(got, expected):
```

```
3   if got == expected:
4     prefix = ' OK '
5   else:
6     prefix = ' X '
7   print('%s got: %s expected: %s' % (prefix, repr(got), repr(
     expected)))
8
9 def main():
10  print('verbing')
11  test('hail', 'hailing')
12  test('swiming', 'swimingly')
13  test('do', 'do')
14
15 if __name__ == '__main__':
16  main()
```

Listing 2.1: Esempio di programma in Python

Nei linguaggi derivati dall'ALGOL come Pascal, C e Perl, i blocchi di codice sono indicati con le parentesi oppure con parole chiave (il C ed il Perl usano ; il Pascal usa begin ed end). In questi linguaggi è solo una convenzione degli sviluppatori il fatto di indentare il codice interno ad un blocco, per metterlo in evidenza rispetto al codice circostante. In Python invece di usare parentesi o parole chiave, usa l'indentazione stessa per indicare i blocchi nidificati in congiunzione col carattere "due punti" (:). Si può usare sia una tabulazione, sia un numero arbitrario di spazi, ma lo standard Python è di 4 spazi.

Python è un linguaggio pseudocompilato: un interprete si occupa di analizzare il codice sorgente (semplici file testuali con estensione .py) e, se sintatticamente corretto, di eseguirlo e non esiste una fase di compilazione separata (come avviene in C, per esempio) che generi un file eseguibile partendo dal sorgente. [7]

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

Complete Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and up-to-date documentation. Versatile Django can be (and has been) used to build al-

most any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). The site you are currently reading is based on Django!

Internally, while it provides choices for almost any functionality you might want (e.g. several popular databases, templating engines, etc.), it can also be extended to use other components if needed. Secure Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash. Scalable Django uses a component-based "shared-nothing" architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands (e.g. Instagram and Disqus, to name just two). Maintainable Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern). Portable Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavours of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

What does Django code look like?

In a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser

to display by inserting the retrieved data into placeholders in an HTML template.

Django web applications typically group the code that handles each of these steps into separate files:

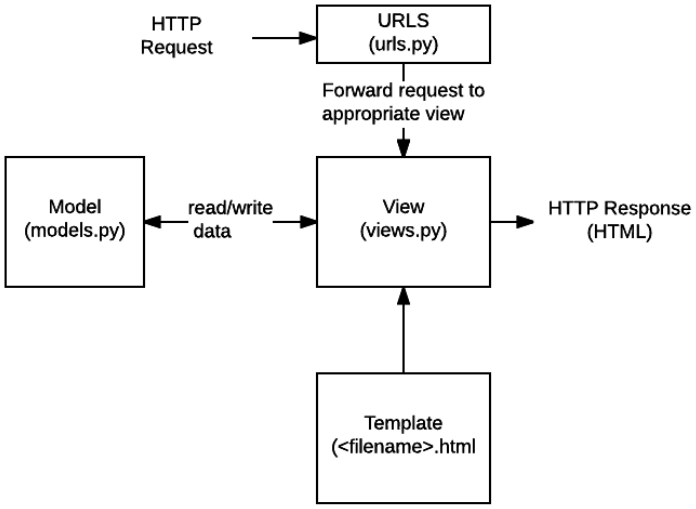

Figura 2.1: Schema di funzionamento generico di un applicativo web sviluppato con Django

URLs: While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in an URL, and pass these to a view function as data. View: A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via models, and delegate the formatting of the response to templates. Models: Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database. Templates: A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A view can dynamically create an HTML page using an HTML template, populating it with data from a model. A template can be used to define the structure of any type of file; it doesn't have to be HTML! [5] [3]

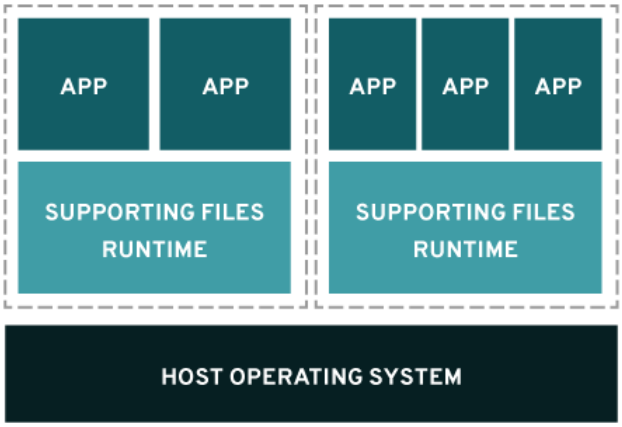

Figura 2.2: Schematizzazione del contenuto di un Container di Docker

2.2 Docker

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Docker raccoglie il software in unità standardizzate chiamate container che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito.

La tecnologia Docker utilizza il kernel di Linux e le sue funzionalità, come Cgroups e namespace, per isolare i processi in modo da poterli eseguire in maniera indipendente. Questa indipendenza è l'obiettivo dei container: la capacità di eseguire più processi e applicazioni in modo separato per sfruttare al meglio l'infrastruttura esistente pur conservando il livello di sicurezza che sarebbe garantito dalla presenza di sistemi separati.

Gli strumenti per la creazione di container, come Docker, consentono il deployment a partire da un'immagine. Ciò semplifica la condivisione di un'applicazione o di un insieme di servizi, con tutte le loro dipendenze, nei vari ambienti.

Docker, considera i container come macchine virtuali modulari estremamente leggere, offrendo la flessibilità di creare, distribuire, copiare e spostare i container da un ambiente all'altro, ottimizzando così le app per il cloud.

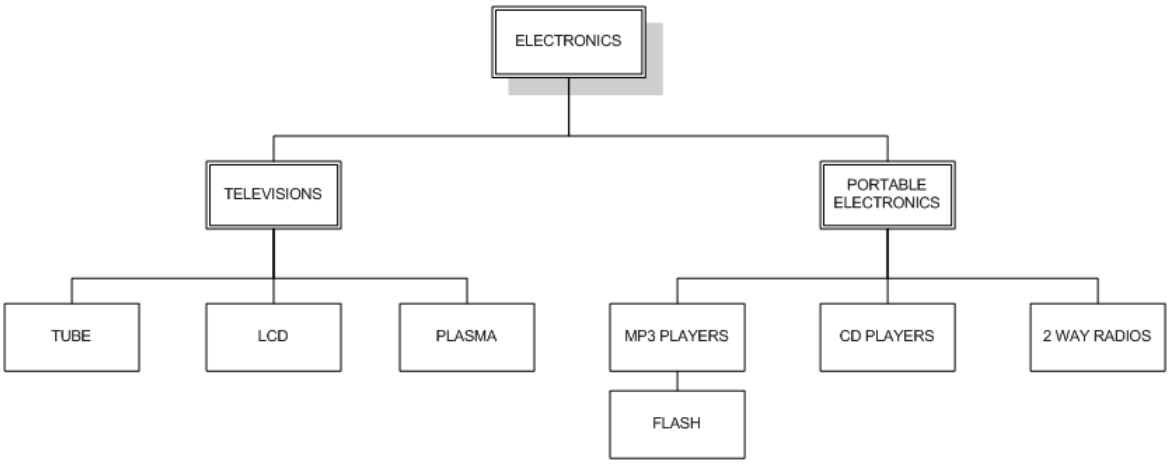

Figura 2.3: Esempio di una gestione di dati in modo gerarchico

I contenitori forniscono una modalità standard per impacchettare il codice della tua applicazione, le configurazioni e le dipendenze, in un oggetto singolo. I contenitori condividono un sistema operativo installato sul server e operano come processi con risorse isolate, assicurando velocità, affidabilità e distribuzioni coerenti, indipendentemente dall'ambiente.

2.3 Strutture dati gerarchiche

Le tabelle di un database relazione non sono gerarchiche (come nel XML), ma sono delle semplici liste piatte. I dati gerarchici sono constituiti da relazioni padre-figlio che non possono essere rappresentate in modo naturale nelle tabelle dei database relazionali. In questo caso, i dati gerarchici sono una collezione di informazioni dove ogni item ha un solo padre e nessuno o più figli (ad eccezione del nodo radice che non ha un nodo padre); questo genere di rappresentazione delle informazioni può essere trovato in diversi ambiti di applicazione di un database, incluse discussioni su forum e mailing list, grafici di organizzazione di un business, categorie per gestire contenuti e categorie di prodotti.

Ci sono differenti modelli per poter gestire dati in modo gerarchico, i più importanti che sono stati presi in considerazione sono i seguenti:

2.3.1 The adjacency list model

Il primo approccio, e quello di più semplice implementazione, qui descritto è chiamato *'adjacency list model* o metodo ricorsivo; è definito tale perchè per funzionare necessita solo di una funzione che itera per tutto l'albero.
In questo modello, ogni item (nodo dell'albero) nella tablla contiene un puntatore al suo item padre; invece il nodo radice avrà un puntatore a un valore NULL per l'item padre.

```
CREATE TABLE category(
        category_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(20) NOT NULL,
        parent INT DEFAULT NULL
);

INSERT INTO category VALUES(1,'ELECTRONICS',NULL),(2,'TELEVISIONS',1),(3,'TUBE',2),
        (4,'LCD',2),(5,'PLASMA',2),(6,'PORTABLE ELECTRONICS',1),(7,'MP3 PLAYERS',6),
(8,'FLASH',7),
        (9,'CD PLAYERS',6),(10,'2 WAY RADIOS',6);

SELECT * FROM category ORDER BY category_id;
+-------------+----------------------+--------+
| category_id | name                 | parent |
+-------------+----------------------+--------+
|           1 | ELECTRONICS          |   NULL |
|           2 | TELEVISIONS          |      1 |
|           3 | TUBE                 |      2 |
|           4 | LCD                  |      2 |
|           5 | PLASMA               |      2 |
|           6 | PORTABLE ELECTRONICS |      1 |
|           7 | MP3 PLAYERS          |      6 |
|           8 | FLASH                |      7 |
|           9 | CD PLAYERS           |      6 |
|          10 | 2 WAY RADIOS         |      6 |
+-------------+----------------------+--------+
```

Figura 2.4: Esempio di una tabella per gestire dati in modo gerarchico secondo l'adjacency list model

Il vantaggio di usare questo modello sta nella sua semplicità di costruzione sopratutto a livello di codice client-side, e di restituzione dei figli di un nodo. Mentre diventa problematico se si lavora in puro SQL e nella maggior parte dei linguaggi di programmazione, è lento e poco efficente, perchè è necessaria

una query per ogni nodo dell'albero, e visto che ogni query impiega un certo periodo di tempo, questo rende la funzione molto lente quando si lavora con alberi di grandi dimensioni. Inoltre molti linguaggi non sono ottimizzati per funzioni ricorsive. Per ogni nodo, la funzione inizia una nuova istanza di se stessa, ogni istanza occupa una porzione di memoria e impiega un certo tempo per inizializzarsi, e più grande è l'albero e più questo processo sarà portato a termine in maggior tempo.

2.3.2 The Nested set model

Il secondo approccio che viene proposto è il *Nested set model*, che permette di osservare la gerarchia in un modo diverso, non come nodi e linee, ma come container innestati.

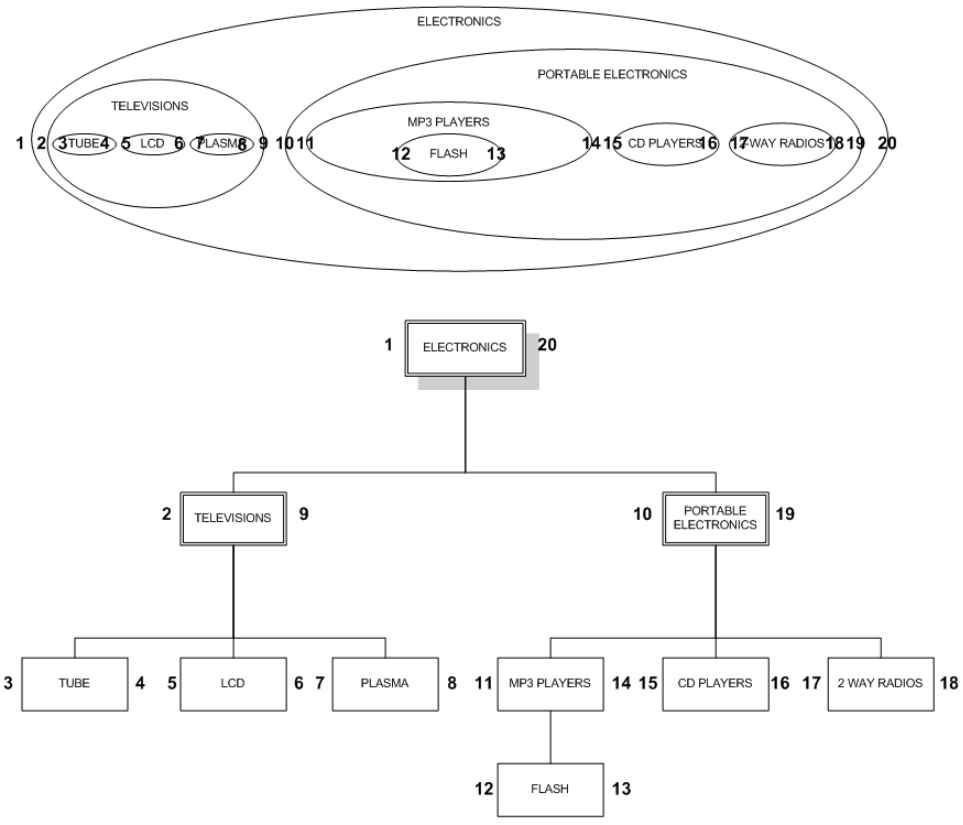

Figura 2.5: Esempio di una gestione di dati in modo gerarchico secondo il Nested set model

La gerarchia dei dati viene rappresentata nella tabella attraverso l'uso degli attributi 'left' e 'right' per rappresentare l'annidamento dei nodi (il nome delle colonne: left e right, hanno significati speciali in SQL; per questo motivo si identificano questi campi con i nomi 'lft' e 'rght'). Ogni nodo dell'albero viene visitato due volte, assegnando i valori in ordine di visita, e in entrambe le visite. Quindi vengono associati ad ogni nodo due numeri, memorizzato come due attributi. I valori di left e right sono determinati come segue: si inizia a numerare a partire dal lato più a sinistra di ogni nodo e si continua verso destra. Lavorando con un albero, si parte da sinistra e si continua verso destra, un livello alla volta, scendendo per ogni nodo i suoi figli, assegnando i valori al campo left, prima di assegnare un valore al campo right, e successivamente si continua verso destra. Questo approccio è chiamato Modified preorder tree traversal algorithm.

A prima vista questo approccio può sembrare più complicato da comprendere rispetto all'adjacency list model, ma quest'utlimo metodo è molto più veloce quando si vuole recuperare i nodi, visto che basta una query, mentre più lento per operazioni di aggiornamento e cancellazione dei nodi; in quest ultimo il grado di complicatezza dell'operazione è determinato dal nodo che si vuole cancellare, a partire dal caso più semplice, il nodo foglia (nodo senza figli) fino al caso più complicato, quando si vuole cancellare il nodo radice.

```
CREATE TABLE nested_category (
        category_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(20) NOT NULL,
        lft INT NOT NULL,
        rgt INT NOT NULL
);

INSERT INTO nested_category VALUES(1,'ELECTRONICS',1,20),(2,'TELEVISIONS',2,9),(3,'TUBE',3,4),
 (4,'LCD',5,6),(5,'PLASMA',7,8),(6,'PORTABLE ELECTRONICS',10,19),(7,'MP3 PLAYERS',11,14),
(8,'FLASH',12,13),
 (9,'CD PLAYERS',15,16),(10,'2 WAY RADIOS',17,18);

SELECT * FROM nested_category ORDER BY category_id;

+-------------+----------------------+-----+-----+
| category_id | name                 | lft | rgt |
+-------------+----------------------+-----+-----+
|           1 | ELECTRONICS          |   1 |  20 |
|           2 | TELEVISIONS          |   2 |   9 |
|           3 | TUBE                 |   3 |   4 |
|           4 | LCD                  |   5 |   6 |
|           5 | PLASMA               |   7 |   8 |
|           6 | PORTABLE ELECTRONICS |  10 |  19 |
|           7 | MP3 PLAYERS          |  11 |  14 |
|           8 | FLASH                |  12 |  13 |
|           9 | CD PLAYERS           |  15 |  16 |
|          10 | 2 WAY RADIOS         |  17 |  18 |
```

Figura 2.6: Esempio di una tabelle per la gestione di dati in modo gerarchico secondo il Nested set model

2.4 Sistemi di raccomandazione

Un sistema di raccomandazione (*Recommendation System*) è un sistema che raccomanda item ad utenti tra numerosi item esistenti in database. L'item è qualsiasi cosa che possa piacere agli utenti, come prodotti, libri e giornali. Si generano aspettative che l'item raccomandato possa essere tra quelli che sia di maggiore interesse; in altre parole, questi item particolari sono in accordo con i gusti degli utenti.

Oggigiorno si possono trovare due trend di sistemi di raccomandazione:

Content-based filtering (CBF): un item viene raccomandato ad un utente se esso è simile agli altri item di interesse o piaciuti nel passato (la valutazione per questi item è alta o sono stati molto utilizzati). Da notare che ogni item ha delle informazioni che ne definiscono le caratteristiche e proprietà (spesso questo insieme di dati viene definito metadati), e sono fondamentali per il processo di raccomandazione.

Collaborative filtering (CF): un item viene raccomandato ad un utente se i suoi vicini (gli altri utenti simili) sono interessati a quell'item.

Entrambi gli approcci (CBF e CF) hanno i loro punti di forza e di debolezza. Il primo algoritmo si focalizza sul contenuto degli item e sugli interessi del singolo utente; raccomanda iteme differenti a utenti differenti. Ogni utente può ricevere raccomandazioni uniche; e questo è un vantaggio. Tuttavia CBF non si tende verso la comunità di utenti come il CF; per questo motivo si potrebbero tralasciare item di interesse ad un utente parchè nascosti all'interno della comunità, CBF non ha l'abilità di scoprire item in modo implicito; e questa è la sua più grande limitazione. Se ci sono molti contenuti associati agli item (per esempio, un item ha molte proprietà) allora, il sistema CF consuma molte risorse e tempo per poter analizzare i dati, nel contempo a questo algoritmo non interessano queste informazioni. Una raccomandazione viene fatta sulla base delle valutazioni degli utenti per gli item, o sugli usi che gli utenti fanno degli item; questo è il suo punto di forza perchè non si trova di fronte a dover analizzare item con un ricco contenuto. Allo stesso tempo è anche il suo punto debole, perchè può portare a suggerimenti che potrebbero essere considerati poco adatti sulla base della poca relazione con i profili di alcuni utenti.

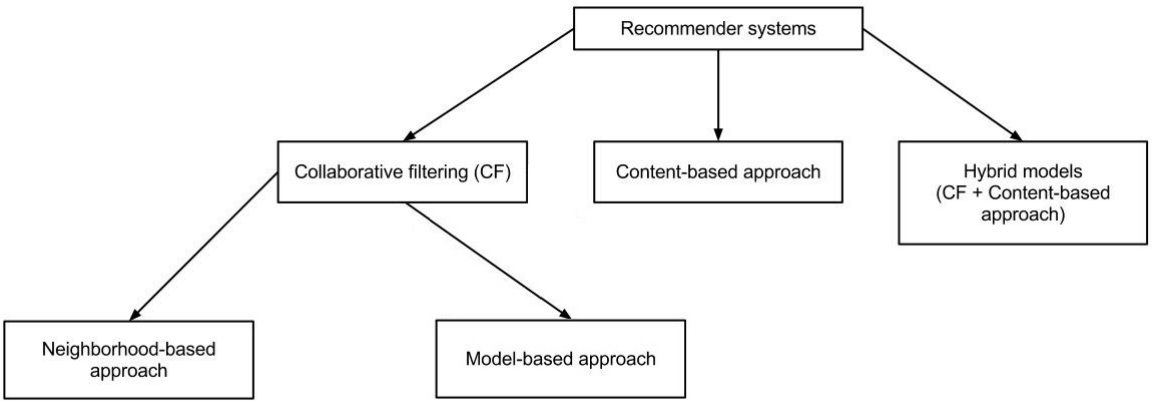

Figura 2.7:

[4] Un sistema di raccomandazione filtra i dati usando differenti algoritmi e raccomanda gli item più rilevanti agli utenti, attraverso un procedimento a 3 fasi:

raccolta di dati : questa è il primo step e anche quello più importante per poter costruire un sistema di raccomandazione che produca risultanti rilevanti e consistenti. I dati possono essere raccolti in due modi: esplicitamente, cioè attraverso i dati che vengono prodotti direttamente

dagli utenti, ad esempio le valutazioni di un prodotto; mentre attarverso l'approccio implicito, vengono raccolti dati che non sono prodotti in modo intenzionale dall'utente ma raccolti dai costanti flussi di dati come la cronologia di ricerca, i click effettuati, lo storico degli ordini, etc.

memorizzazione di dati : la quantità di dati definisce quanto efficace un modello di raccomandazione possa di diventare. Ad esempio, in un sistema di raccomanzione per film, maggiori sono le valutazioni fornite dagli utenti, e migliore sarà il sistema di raccomandazione per gli altri utenti. Il tipo di dati che si vuole raccogliere determina anche il supporto di memorizzazione più adatto.

Filtraggio dei dati : dopo la fase di raccolta e memorizzazione dei dati, essi vanno filtrati per poter estrarre le informazioni rilevanti per poter effettuare le raccomandazioni finali, e sono già disponibili diversi algoritmi che semplificano quest ultima fase.

I sistemi di raccomandazione possono essere suddivisi nelle seguenti categorie, ma speso si preferisco degli approcci imbridi cioè delle combinazioni di sistemi di raccomandazione basati sul contenuto (*Content-based filtering*) e quelli collaborativi (*Collaborative filtering*) in modo da essere più efficaci sfruttando i pregi di entrambi gli approcci.

2.4.1 Content-based filtering

Un Content-based filtering è un sistema di raccomandazione in cui vengono suggeriti item simili a un particolare item (oggetti o prodotti).

Questo approccio sfrutta i metadati dell'item, che possono essere il genere, una descrizione, uno o più autori, la categoria di appartenenza etc. per fare queste raccomandazioni; l'idea base che sta dietro questi raccomandatori, è che se ad un utente piace o interessa un particolare item allora gli piaceranno anche altri item simili.

Questo algoritmo suggerisce prodotti che piacevano all'utente nel passato ed è limitato a item dello stesso tipo. Un content-based recommender fa riferimento a quegli approcci, che provvedono raccomandazioni comparano la rappresentazione del contenuto che descrive un item e la rappresentazione del contenuto dell'item interessato dall'utente.

Questi metodi sono usati quando si sanno a priori delle informazioni sugli item che si vuole suggerire, ma non sugli utenti. In questo sistema, delle keyword (parole chiave) sono utilizzate per caratterizzare gli item e un profilo dell'utente è costruito per indicare quali item gli piacciono. In altre parole,

questi algoritmi cercano di raccomandare item che all'utente sono piaciuti o ha usato nel passato e sta esaminando nel presente. La costruzione del profilo dell'utente, spesso temporaneo, non viene basata su un modulo di registrazione che l'utente stesso deve compilare, ma su informazioni lasciate indirettamente dall'utente. Più precisamente, tra vari item candidati da raccomandare all'utente si passa per un processo di confronto con gli item piaciuti dall'utente e gli item migliori vengono suggeriti.

2.4.2 Collaborative filtering

I filtri collaborativi (*Collaborative filtering*) lavorano costruendo un database di preferenze di utenti su item (o prodotti), sfruttano tecniche di analisi dei dati al problema di aiutare gli utenti a trovare gli item che gli potrebbero piacere producendo una lista dei top-N item da raccomandare per un dato utente. Un nuovo utente subisce un processo di matching all'interno del database per scoprire quali sono i possibili vicini (*neighbors*), che corrispondo agli altri utenti aventi storicamente simili preferenze al nuovo utente. Agli item maggiormente preferiti dai vicini sono raccomandati al nuovo utente, visto che potrebbero essere di suo interesse.

Questi sistemi tentano di predirre la valutazione o la preferenza che un utente darebbe a un item basandosi su preferenze date da altri utenti, queste preferenze possono essere ottenute o in modo esplicito dagli utenti o tramite qualche misurazione implicita. I filtri collaborativi non richiedono l'uso di metadati associati agli item come nella loro controparte, i filtri content-based. A un utente vengono raccomandati item basandosi su valutazioni passate collezionate da altri utenti.

Tuttavia, restano ancora oggi alcune sfide significative a cui sono sottoposti i sistemi di raccomandazione basati su filtraggio collaborativo. Il primo obbiettivo è quello di migliorare la scalabilità degli algoritmi di filtri collaborativi; questi algoritmi sono in grado di cercare anche diecimila di potenziali vicini (utenti simili) in tempo reale, ma la richiesta dei sistemi moderni è di cercare dieci milioni di potenziali vicini. Algoritmi esistenti hanno problemi di performance con i singoli utenti quando essi hanno molte informazioni. Il secondo obbiettivo è quello di migliorare la qualità dei sistemi di raccomandazione per gli utenti. Gli utenti vogliono raccomandazioni di cui possono fidarsi e che possono aiutarli a trovare item che potrebbero essere di loro gusto. Per certi versi questi due obbiettivi sono in conflitto tra di loro e per ottenere dei risultati validi e di una certa importanza è necessario trattarli in contemporanea perchè aumentare solamente la scalabilità diminuirebbe la sua qualità e viceversa. [8]

Il principale modello di filtro collaborativo studiato in questo elaborato è il metodo definito come *Memory-based* e il vantaggio di utilizzare queste tecniche sta nel fatto di essere semplici da implementare e i risultati ottenuti sono altrettanto semplici da spiegare; mentre ci possono essere anche filtri collaborativi che sfruttano metodi *Model-based* che si basano sulla fattorizzazione di matrici e sono molto più funzionali per gestire il problema della sparsità dei dati. Questi ultimi sono sviluppati usando algoritmi di data mining e machine learning per predirre le valutazioni di utenti su item senza valutazioni, inoltre sono spesso associati a tecniche come la dimensionality reduction per migliorare la precisione.

2.4.3 Challenges and limitations

Cosa succederebbe se un nuovo utente o un nuovo item è aggiunto al dataset? Questa situazione è chiamata *Cold Start*. Ci possono essere due tipi di Cold Start:

Visitor Cold Start :

Product Cold Start :

Cold start problem What will happen if a new user or a new item is added in the dataset? It is called a Cold Start. There can be two types of cold start: Visitor Cold Start: means that a new user is introduced in the dataset. Since there is no history of that user, the system does not know the preferences of that user. It becomes harder to recommend products to that user. So, how can we solve this problem? One basic approach could be to apply a popularity based strategy, i.e. recommend the most popular products. These can be determined by what has been popular recently overall or regionally. Once we know the preferences of the user, recommending products will be easier. Product Cold Start: means that a new product is launched in the market or added to the system. User action is most important to determine the value of any product. More the interaction a product receives, the easier it is for our model to recommend that product to the right user. We can make use of Content based filtering to solve this problem. The system first uses the content of the new product for recommendations and then eventually the user actions on that product.

Cons Scalability: The more K neighbors we consider (under a certain threshold), the better my classification should be. Nevertheless, the more users there are in the system, the greater the cost of finding the nearest K neighbors will be. Cold-start: New users will have no to little information

about them to be compared with other users. New item: Just like the last point, new items will lack of ratings to create a solid ranking (More of this on 'How to sort and rank items').

SPARSITY

Stated simply, most users do not rate most items and, hence, the user ratings matrix is typically very sparse. is is a problem for collaborative +ltering systems, since it decreases the probability of +nding a set of users with similar ratings. This problem often occurs when a system has a very high item-to-user ratio, or the system is in the initial stages of use. This issue can be mitigated by using additional domain information or making assumptions about the data generation process that allows for high-quality imputation.

Capitolo 3

Recommendation Systems

3.1 Memory-based

I filtri collaborativi Memory-based sono stati introdotti per via delle osservazioni che vennero fatti sugl utenti, i quali si fidano maggiormente delle raccomandazioni di altri che la pensano allo stesso modo. Questi metodi mirano a calcolare le relazioni tra utenti e item attraverso lo schema dei vicini che identifica sia coppie di item che tendono ad essere usati insieme o hanno un grado di similarità alto o utenti con uno storico di item usati simile. [6] Questi approcci divennero molto famosi grazie alla loro semplicità di implementazione, molto intuitivi, non necessitano il training e aggiustamento di molti parametri, e l'utente può capire la ragione che sta dietro ogni raccomandazione

Filtri collaborativi che usano metodi Memory-based (definiti anche *Neighborhood-based*) possono essere classificati in altre due categorie:

3.1.1 User-based filtering

Questi sistemi, definiti anche con l'acronimo UB-CF (*User-based Collaborative Filter*) raccomandano una serie di item a un utente che utenti simili hanno usato o valutato. Questo algoritmo prima trova un valore che rappresenta la similarità tra utenti. E basandosi su questi valori, prende uno o più utenti tra quelli che risultano simili e raccomanda item che questi utenti simili hanno usato o valutato in precedenza.

Molti di questi approcci possono essere generalizzati dall'algoritmo definito dai seguenti step:

1. Specificare qual'è il l'utente a cui si vuole applicare l'algoritmo di raccomandazione e recuperare quali utenti possono avere dato valutazioni o usato item simili al utente target. Piuttosto che recuperare tutti gli utenti, per velocizzare l'esecuzione dell'algoritmo, è possibile selezionare soltanto un gruppo di utenti in modo casuale oppure associare dei valori di similarità tra tutti gli utenti e confrontando questi valori con quello dell'utente target, selezionare i relativi utenti che superano una soglia scelta, oppure utilizzare tecniche di clustering. un numero limitato per effettuare la raccomandazione,

2. Estrarre quegli item a cui l'utente target non ha mai interagito e per questo motivo gli possono interessare, e mostrarli all'utente target.

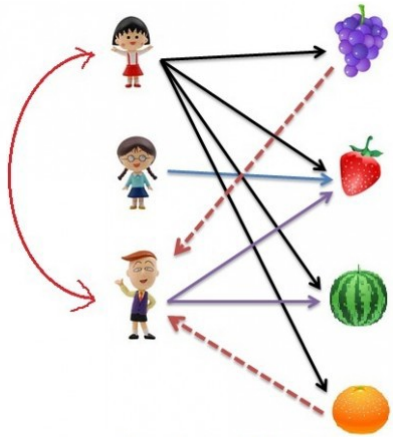

Figura 3.1: Esempio di applicazione di un sistema di raccomandazione User-based

Questi approcci sono facilmente implementabili, indipendenti dal contesto in cui sono applicati e possono essere più accurati rispetto a tecniche basate sul content-based dall'altra parte all'aumentare del numero di utenti che vado a considerare per fare le raccomandazioni migliore è la precisione di questo processo ma anche è maggiore il costo per compiere questo procedimento. Altro problema che affligge questi sistemi, e che verrà approfondito nel prossimo paragrafo, è definito di Cold-start.

3.1.2 Item-based filtering

Quando viene applicato per milioni di utenti e item, l'algoritmo UB-CF non è molto efficente, per via della complessa computazione della ricerca di utenti simili; così in alternativa è stato introdotto l'algoritmo di filtraggio Item-based, definito anche IB-CF (*Item-based Collaborative Filter*): dove piuttosto che effetuare il confrotto tra utenti simili, viene fatto un confronto tra gli item dell'utente a cui si vule raccomandare e i possibili item simili.

Questi sistemi sono estremamente simili ai sistemi di raccomandazione Content-based, e identificano item simili in base a come utenti gli hanno usati nel passato.

[8]

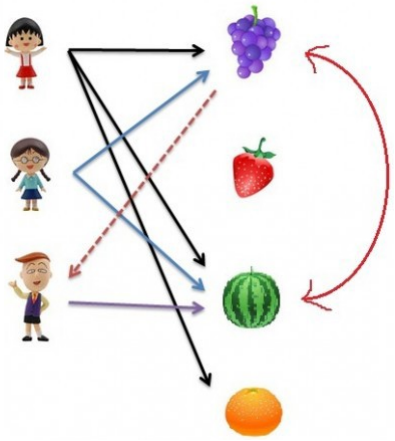

Figura 3.2: Esempio di applicazione di un sistema di raccomandazione Item-based

3.2 Model-based filtering

Gli algoritmi Model-based tentano di comprimere grandi database in un modello ed effettuare il processo di raccomandazione applicando dei meccanismi di riferimento all'interno di questo modello. I CF Model-based posssono rispondere alle richieste degli utenti istantaneamente. [4]

Capitolo 4

Descrizione della soluzione

Capitolo 5

Conclusioni

Presentazione dei risultati e conclusioni: la presentazione dei risultati dovrebbe consistere in una descrizione tecnica dei risultati raggiunti,unitamente ad un commento critico e ad un'analisi della rispondenza agli obiettivi iniziali (si consiglia pertanto di motivare la rilevanza dei risultati e l'eventuale scostamento dagli obiettivi iniziali). La sezione relativa ai risultati dovrebbe infine contenere una sintesi critica e un giudizio sull'esperienza effettuata, che renda conto di aspetti positivi e negativi per il tirocinante e per l'ente ospitante, del valore formativo, professionale e umano, così via.

Bibliografia

[1] M. Anisetti et al. «A semi-automatic and trustworthy scheme for continuous cloud service certification». In: *IEEE TRANSACTIONS ON SERVICES COMPUTING* (2017). DOI: 10.1109/TSC.2017.2657505.

[2] M. Anisetti et al. «Moon Cloud: A Cloud Platform for ICT Security Governance». In: (dic. 2018), pp. 1–7. DOI: 10.1109/GLOCOM.2018.8647247.

[3] *Django documentation*. https://docs.djangoproject.com/en/2.2/. 2019.

[4] Minh-Phung Do, Dung Nguyen e Academic Network of Loc Nguyen. «Model-based approach for Collaborative Filtering». In: ago. 2010.

[5] *MDN Django documentation*. https://developer.mozilla.org/it/docs/Learn/Server-side/Django/. 2019.

[6] Miquel Montaner, Beatriz López e Josep Lluís de la Rosa. «A Taxonomy of Recommender Agents on the Internet». In: *Artificial Intelligence Review* 19.4 (giu. 2003), pp. 285–330. ISSN: 1573-7462. DOI: 10.1023/A:1022850703159. URL: https://doi.org/10.1023/A:1022850703159.

[7] *Python 3.7 documentation*. https://docs.python.org/3.7/. 2019.

[8] Badrul Sarwar et al. «Item-based Collaborative Filtering Recommendation Algorithms». In: WWW '01 (2001), pp. 285–295. DOI: 10.1145/371920.372071. URL: http://doi.acm.org/10.1145/371920.372071.