UNIVERSITA' DEGLI STUDI DI PADOVA

Attività formativa:
Tesina

Recommendation System e loro Applicazioni

Laureando: Davide Callegaro

Relatore: Andrea Alberto Pietracaprina

Corso di Laurea in Ingegneria dell'Informazione

22 novembre 2013

Anno Accademico 2013/2014

Indice

Capitolo 1

Introduzione

Questo lavoro nasce dal mio interesse di comprendere come funzionano i sistemi di suggerimento degli amici sui social network, come ad esempio *Facebook*. Il percorso intrapreso mi ha portato a studiare uno degli strumenti più potenti ed utilizzati al fine di consigliare oggetti in Internet: i Recommendation System (R.S.). Tali sistemi, oltre ad avere un impatto importante nei social network, hanno un ruolo centrale nel mondo del commercio on-line, ove sono usati per consigliare agli utenti nuovi prodotti. Il risultato raggiunto è un algoritmo che propone un gruppo di possibili amici da sottoporre al giudizio dell'utente.

Si comincerà la trattazione (Cap. 2) con alcune nozioni di base necessarie per la comprensione dei concetti successivi, approfondendo in particolare il concetto di *distanza*, accompagnandolo da qualche esempio, e descrivendo la tecnica del *clustering*. Si passerà poi (Cap. 3) ad una presentazione dei R.S., sottolineandone l'importanza con una breve sintesi della loro storia. Si continuerà con lo sviluppo della teoria, prediligendo l'approccio classico. Questo divide i R.S.in sistemi *content based*, basati sui contenuti degli oggetti, e i sistemi *collaborative filtering*, a filtro collaborativo, che non necessitano la conoscenza gli elementi, ma si limitano ad osservare il comportamento degli utenti in relazione a questi. Gran parte di questa sezione prende spunto dai capitoli 3, 7 e 9 di *Mining of Massive Datasets*[7].

Nella parte conclusiva (Cap.4) si mostra un algoritmo utilizzabile per il suggerimento di amici su *Facebook* [10] sviluppato autonomamente in seguito allo studio delle tecniche generali.

Capitolo 2

Concetti di base

Un R.S.è innanzitutto una tecnica di *data mining*, ramo dell'informatica che si occupa di estrarre informazioni da grandi insiemi di dati. E' inoltre una tecnica di *Information filtering*, in quanto si occupa di filtrare le informazioni che devono arrivare all'utente, facendo passare tra le sue maglie solo quelle più interessanti. Un R.S.deve essenzialmente consigliare ad un *utente* uno o più *oggetti* che non conosce, ma che potrebbero interessargli. Per fare questo si devono conoscere elementi della nozione di distanza e delle tecniche di clustering. Infatti la *distanza* ci permette di capire quanto due oggetti, o due utenti, siano diversi (o alternativamente simili); il *clustering* ci dà la possibilità di raggruppare quelli che si somigliano, siano essi oggetti o utenti. Questi sono sotto-obiettivi per arrivare allo sviluppo di un consiglio.

2.1 Trovare somiglianze tra gli oggetti

Trovare oggetti simili all'interno di un insieme è uno dei problemi fondamentali di *data mining*. Tale operazione è utile ad esempio per rilevare plagi o, come nel nostro caso, per cercare persone simili da consigliare come amici in un social network. Vale allora la pena soffermarsi in particolare su alcune distanze notevoli, utili esempi per poi sviluppare distanze adattabili ai problemi più disparati.

2.1.1 Distanza euclidea, jaccardiana e coseno

Definizione 2.1.1:
Sia dato un insieme di punti, detto spazio. Ivi definiamo la funzione $d(x, y)$, detta *misura di distanza*, la quale dovrà soddisfare queste proprietà:

- $d(x, y) \geq 0$: le distanze non possono essere negative

- $d(x, y) = 0 \iff x = y$: la distanza da un punto a se stesso è nulla

- $d(x, y) = d(y, x)$: le distanze godono della proprietà di simmetria

- $d(x, y) \leq d(x, z) + d(z, y)$: deve valere la *disuguaglianza triangolare*, o *disuguaglianza di Cauchy-Schwartz*

Si noti che la proprietà maggiormente vincolante è la disuaguaglianza triangolare. Infatti essa evidenzia che se una distanza è ben definita, non esiste alcun vantaggio nel passare per un particolare punto z, oltre a collegare il concetto di distanza all'idea di percorso minimo tra due punti. Ovviamente l'ultima interpretazione va considerata con attenzione, perché l'idea di percorso minimo va adattata allo spazio in cui ci si trova: si pensi ad esempio ad uno spazio non isotropo, nel quale tale considerazione non sarebbe vera.

Per definire la distanza di Jaccard si deve prima definire la *somiglianza di Jaccard*.

Definizione 2.1.2:
Dati due insiemi S e T appartenenti allo stesso Universo, si denota:

$$SIM(S, T) = [S \cap T]/[S \cup T] \tag{2.1}$$

Da cui deriva:

Definizione 2.1.3:
La distanza di Jaccard è definita come:

$$d_j(x, y) = 1 - SIM(x, y) \tag{2.2}$$

Si verifica facilmente che tale distanza rispetta le proprietà fondamentali.

In uno spazio euclideo *n-dimensionale*, la distanza più semplice è la distanza euclidea.

Definizione 2.1.4:
La *distanza euclidea*, o *norma*, di L_r si definisce come:

$$d([x_1, ..., x_n], [y_1, ..., y_n]) = \sqrt[r]{\sum_{i=1}^{n} |x_i - y_i|^r} \tag{2.3}$$

Per concludere questo elenco di distanze notevoli, introduciamo la distanza coseno.

Definizione 2.1.5:
Un'altro tipo di distanza tra vettori in spazi euclidei è la *distanza coseno*: dati due vettori $\vec{u} = [u_1, ..., u_n]$ e $\vec{v} = [v_1, ..., v_n]$ si definisce:

$$\cos(u, v) = \frac{\sum_{i=1}^{n} v_i \cdot u_i}{\sqrt{\sum_{i=1}^{n} v_i^2 \cdot \sum_{i=1}^{n} u_i^2}} \tag{2.4}$$

Essa, come vedremo, è quella che utilizzeremo maggiormente nelle applicazioni.

In generale bisogna muoversi con attenzione in spazi ad alta dimensione: essi infatti sono dotati di proprietà non intuitive. Si parla in inglese di *Curse of Dimensionality* (maledizione della dimensionalità) ed essa si manifesta nel rendere i punti all'incirca equidistanti, e nel rendere tutti i vettori all'incirca ortogonali.

Prendiamo ad esempio lo spazio euclideo *d-dimensionale* con valori scelti in $[0, 1] \subset \mathbb{R}$. Ricordando la (2.3)

$$d([x_1, ..., x_d], [y_1, ..., y_d]) = \sqrt[r]{\sum_{i=1}^{d} |x_i - y_i|^r}$$

per d molto grande, è molto probabile che qualche coppia $|x_i - y_i|$ sia circa 1. In tale evenienza la sommatoria risulterà circa unitaria. La successiva applicazione della radice r-esima accumulerà ancora più termini intorno all'unità. Per questo motivo la distanza per ogni coppia di punti $\{x, y\}$ sarà circa 1.

Allo stesso modo immaginiamo in tale spazio di utilizzare la distanza coseno (2.4):

$$\cos(x, y) = \frac{\sum_{i=1}^{d} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \cdot \sqrt{\sum_{i=1}^{d} y_i^2}}$$

Ipotizzando che i punti siano presi in modo casuale nello spazio, possiamo vedere ogni coordinata x_i, y_i come una variabile aleatoria uniforme sui reali, indipendente dalle altre coordinate. Valutando ora il denominatore, si osserva che esso aumenta all'aumentare della dimensione, e in particolare non è nullo a meno che uno dei due punti non sia l'origine. D'altro canto il numeratore, essendo prodotto di variabili aleatorie sia positive che negative, al crescere della dimensione tende alla sua media, cioè 0. Questo fa tendere allora tutta la frazione a 0, che per definizione è condizione di ortogonalità, provando che due vettori casuali in alte dimensioni sono circa ortogonali.

2.1.2 Trovare somiglianze in insiemi di oggetti con il Minhashing

Consideriamo due insiemi, C ed S, rispettivamente contenenti *oggetti* ed *insiemi*.

Consideriamo ora la matrice $M = [C \times S]$, composta da elementi binari, ad esempio in $\{0, 1\}$. Ogni elemento (c, s), con $c \in C$ e $s \in S$, sarà 1 se $c \in s$, oppure 0 altrimenti.

Il *minhashing* è una tecnica utilizzata per trovare la distanza di Jaccard in modo efficiente.

Definizione 2.1.6:
Il *minhash* è una funzione che associa alla colonna di una matrice binaria un numero intero. Prima si opera una permutazione delle righe della matrice, poi si associa ad ogni colonna un numero che rappresenta la riga a cui si incontra il primo 1.

Ad esempio se una colonna permutata fosse [00001...], la funzione le assocerebbe valore 5.

Tale tecnica è utile grazie alla seguente proprietà, in cui osserviamo che c'è un legame tra tale valore e la distanza di Jaccard. Infatti è vero che:

Teorema 2.1.7:
La probabilità che la funzione di minhash in una permutazione casuale delle righe produca lo stesso valore per due insiemi, è uguale alla somiglianza di Jaccard tra questi due insiemi.

Ovviamente avere la somiglianza non è un problema visto che è legata alla distanza dall'equazione (2.2). Per approfondire il minhash e le sue applicazioni si veda il Capitolo 3 di [7].

2.2 Clustering

Definizione 2.2.1:
Il *clustering* è un insieme di tecniche di analisi multivariata di dati, volte al raggruppamento di elementi in insiemi omogenei. Determinante affinché la tecnica abbia successo, è una buona definizione di distanza tra gli elementi da raggruppare.

2.2.1 Introduzione al clustering

Gli algoritmi di clustering possono essere divisi in due categorie:

- *Gerarchico*: l'algoritmo comincia considerando ogni punto un cluster a sé, e procede accorpando i più simili tra loro. Il processo si ferma quando si raggiunge un certo numero di cluster o quando i cluster rimasti sono ad una certa distanza.

- *Ad assegnazione*: in questo caso i punti sono considerati in un certo ordine e poi assegnati al cluster al quale sono più adatti (solitamente più vicini). Tale fase è di norma preceduta da una di pre-processazione per determinare i cluster di partenza.

Nel caso ci si trovi in uno spazio euclideo una nozione utile è il *centroide*.

Definizione 2.2.2:
Dato uno spazio euclideo S e un *cluster* formato dai punti $\{p_1, ..., p_n\}$, con $p_i \in S$, si dice che il punto $c \in S$ è il loro *centroide* se per ogni coordinata di c vale:

$$c_i = \frac{1}{n} \sum_j x_{j,i} \tag{2.5}$$

ove $x_{j,i}$ è la coordinata i-esima del j-esimo punto.

Si noti che il centroide è solo una rappresentazione formale del cluster, per cui non è detto che sia un punto che appartiene davvero al cluster.

2.2.2 Clustering gerarchico

Cominciamo la trattazione dai cluster gerarchici. All'inizio ogni punto sarà un cluster indipendente, poi, man mano che procederà l'algoritmo di aggregazione, essi diminuiranno di numero, aumentando in dimensione. L'algoritmo di base di ogni tecnica di clustering gerarchico è del tipo

while *non è tempo di fermarsi* **do**
    trova i due cluster più vicini da unire;
    unisci di due cluster trovati;
end

A questo punto si pongono in particolare tre problemi:

1. Come vengono *rappresentati* i cluster?

2. Come *scegliere* i cluster più vicini?

3. Quando *fermare* l'algoritmo?

Negli spazi euclidei possiamo utilizzare i centriodi per rappresentare i cluster (proprio come un baricentro in un insieme di punti dotati di massa). Segue l'idea di utilizzare la distanza euclidea (2.3) per determinare i due cluster a minor distanza, ed unirli.

Mentre le scelte precedenti, trovandoci in uno spazio euclideo, sono quasi obbligate, per scegliere quando fermarci nel processo di aggregazione abbiamo almeno due opzioni: possiamo fermarci una volta raggiunto un numero target di cluster, oppure quando la prossima unione si dimostrerebbe inadeguata. Nel primo caso l'algoritmo di clustering è preceduto da una fase di *pre-processing*, atta a determinare il numero di cluster ottimale (un esempio è l'algoritmo *k-means*). In altre applicazioni, invece, il numero target può derivare da qualcosa che è successo in precedenza, o magari essere impostato dall'utente. Nel secondo caso, per capire che la prossima unione sarebbe inadeguata, posso utilizzare la distanza dal centroide: mi fermo se dopo la prossima unione la distanza media all'interno del nuovo cluster oltrepassa un certo limite. Si noti che questo approccio verte sull'ipotesi che i cluster non possano distribuirsi troppo nello spazio.

Approcci alternativi per scegliere i cluster da unire si basano sulla definizione di nuovi parametri.

Definizione 2.2.3:
Si dice *raggio* di un cluster, il massimo tra le distanze dei punti del cluster dal centroide. Si dice *diametro* di un cluster, il massimo delle distanze tra ogni coppia di punti del cluster.

Utilizzando queste definizioni possiamo scegliere di unire due cluster in modo che il raggio o il diametro dell'unione sia il minore possibile.

Un'osservazione interessante è che in questo contesto i concetti di raggio e diametro non sono legati da alcuna legge di proporzionalità. Si immagini, ad esempio, un cluster con una forte concentrazione di punti intorno all'origine: tutti nella sfera di raggio unitario, e un punto a distanza 10 volte maggiore. Il centroide è circa nell'origine, e quindi il raggio circa 1. Il diametro sarà invece circa 11.

Infine osserviamo che l'algoritmo *brute force*, usato per dividere un gruppo di punti in cluster cercando tutte le distanze ogni volta, risulta avere andamento $O(n^2 \log n)$ nel numero di punti, e per questo può essere utilizzato solo per piccoli insiemi.

Teorema 2.2.4:
Un algoritmo di clustering con approccio brute force, cioè utilizzando le

distanze tra ogni coppia di punti, ha andamento asintotico $O(n^2 \log n)$.

Dimostrazione. Partiamo con un numero k di elementi da dividere in l cluster. L'algoritmo *brute force* procede così:

$h \leftarrow$numero di cluster attuale;
ogni punto è cluster di centroide se stesso;
$h \leftarrow k$;
trova la distanza tra ogni coppia di centroidi;
while $h > l$ **do**
    trova la distanza minima tra quelle che separano le coppie di centroidi;
    unisci i cluster che hanno centroidi con distanza minima;
    trova il centroide del nuovo cluster;
    calcola le distanze tra il nuovo centroide e gli altri;
end

Il ciclo è eseguito $k - l$ volte, all'interno di ogni ciclo si eseguono

$$\#\text{distanze} + 2 = \sum_{i=1}^{n-1} i + 2 = \frac{n(n-1)}{2} + 2 \tag{2.6}$$

Perciò per ogni iterazione del ciclo, il numero di operazioni eseguite è $O(n^2)$, e man mano che l'algoritmo prosegue, il numero di punti diminuisce, per cui è una somma di quadrati da 1 a n che dà $O(n^3)$.

Notiamo che utilizzando alcune strutture dati nell'analisi si può essere più precisi:

1. all'inizio processiamo le distanze tra tutti i punti: abbiamo visto che questo costa $O(n^2)$;

2. utilizziamo una *priority queue* ove immagazzinare le distanze relative a ciascuna coppia: anche questo costa $O(n^2)$. Questo permette di trovare sempre la distanza minima in un passo;

3. quando decidiamo di unire i cluster C e D, estraiamo tutte le *entry* dalla *priority queue* in $O(n \log n)$ essendoci $2n$ cancellazioni da fare, richiedendo $O(\log n)$ per ogni cancellazione ;

4. eseguiamo poi tutte le distanze tra il nuovo cluster, o meglio il suo centroide, e gli altri cluster. Questa azione richiede tempo $O(n \log n)$ essendoci al più n elementi nella coda. Infine si inseriscono tali distanze nella coda, operazione che richiede tempo $O(n \log n)$.

Essendo i primi due punti eseguiti una volta, mentre i secondi due svolti n volte, il costo asintotico complessivo è $O(n^2 \log n)$. □

2.2.3 Clustering ad assegnazione: cenno agli algoritmi k-means

L'altra famiglia di algoritmi di clustering sono quelli ad assegnazione, dei quali diamo un esempio: la famiglia degli algoritmi di *k-means*. Le ipotesi per utilizzarli sono: trovarsi in uno spazio euclideo, e conoscere a priori il numero k di cluster finali. Tale famiglia di algoritmi ha una struttura di base del tipo:

```
Inizialmente scegli k punti che probabilmente appartengono a cluster
diversi;
Utilizza tali punti come centroidi dei cluster;
for ogni punto p rimasto do
    trova il centroide al quale p è più vicino;
    aggiungi p al cluster di quel centroide;
    ricalcola il centroide del cluster di p;
end
```

Da tale pseudocodice si riconoscono i punti salienti: trovare k e scegliere i primi k punti.
Per quanto riguarda il primo problema si osserva sperimentalmente che il diametro dei cluster diminuisce molto rapidamente al crescere di k, fino ad assestarsi ad un certo valore, dal quale non si discosta più molto. Perciò l'idea è di eseguire l'algoritmo di k-means per valori crescenti esponenzialmente di k: $k = 1, 2, 4, 8, ...$ Da un certo valore in poi il diametro si assesterà ed in seguito si sceglierà di quanto, e come, raffinare il risultato.
Per il secondo problema, ossia scegliere i primi k punti, si possono seguire due strade: cercare quelli più distanti tra loro, oppure applicare una tecnica di clustering su un campione dei dati.

Nel secondo caso si può ad esempio scegliere una tecnica di clustering gerarchico, ma ci si rende conto che serve un po' di tempo per l'elaborazione. Nel primo caso si può fare qualcosa come:

```
Prendi a caso il primo punto;
while ci sono meno di k punti do
    aggiungi il punto la cui distanza minima dai punti selezionati è la
    massima possibile;
end
```

2.2.4 Accenno agli algoritmi di clustering in spazi non euclidei

Si vuole qui di seguito enunciare un algoritmo valido in spazi non euclidei. Questo per vedere come viene affrontato il problema di non poter utilizzare un punto formale, il centroide, per la rappresentazione del centro del cluster, ma si deve usare proprio un punto appartenente al cluster. Infatti non possiamo processare una serie di punti e trovare sempre un punto medio tra loro in spazi che non godano delle proprietà euclidee. Si pensi ad esempio di cercare il punto medio tra due stringhe *abcd* ed *aedb*.

Ci riferiamo nel seguito all'algoritmo di GRGPF (V. Ganti, R. Ramakrishnan, J. Gehrke, A.Powell e J. French). In tale algoritmo, pur senza la condizione di essere in uno spazio euclideo, è necessario ipotizzare che sia ivi definita una qualche misura di distanza, altrimenti nessuna operazione di clustering potrebbe essere eseguita. Seguono poi due definizioni importanti:

Definizione 2.2.5:
Sia la funzione *rowsum(p,C)* una funzione che associa a $p \in C$ la somma dei quadrati delle distanze dagli altri punti dell'insieme C da p stesso.

E' quindi un parametro che essenzialmente è il quadrato della media quadratica delle distanze dei punti di un cluster, e vuole rappresentare quanto p sia *centrale* in C. Ovviamente minore è la $rowsum(p, C)$, più p sarà centrale in C. Secondo il ragionamento di cui sopra definiamo il punto più centrale del cluster C.

Definizione 2.2.6:
Definiamo *clusteroide* del cluster C, il punto c tale che:

$$c = \underset{\forall p \in C}{argmin}\, rowsum(p, C) \tag{2.7}$$

Questo in pratica è l'equivalente del centroide negli spazi non euclidei.
L'algoritmo si basa sull'organizzare i cluster in un albero.

Definizione 2.2.7:
L'albero dell'algoritmo GRGPF è così formato: ha nelle foglie una o più caratteristiche che rappresentino un cluster. Ha poi nei nodi interni un campione finito, di numero fissato n, di clusteroidi, e per ciascuno ha la sua *rowsum* rispetto al cluster di appartenenza. Tali clusteroidi sono scelti casualmente tra i cluster rappresentati in ciascun sottoalbero. Ad esempio ogni nodo potrebbe scegliere i clusteroidi tra quelli rappresentati dai propri figli, ma non è necessario.

Una buona rappresentazione di un cluster in una foglia potrebbe essere:

1. N, numero di punti in un cluster;
2. il *clusteroide*;
3. la $rowsum(c_i, C_i)$ del clusteroide c_i rispetto al suo cluster C_i.
4. scelta una costante k, i k punti più vicini al clustroide e le loro $rowsum$; essi sono necessari poiché con l'aggiunta di punti, il nuovo clustroide sarà uno di questi punti.
5. i k punti più lontani dal clustroide e le loro $rowsum$; questi punti sono utili a delimitare il cluster e ci faranno capire se punti appena fuori dal cluster formato sono abbastanza vicini per essere uniti al cluster.

Viene fatto uno sforzo per mantenere tra i figli di un certo nodo, cluster vicini tra loro. In tale modo si può semplicemente assegnare un punto p ad un certo cluster scendendo l'albero con approccio *top-down*. Un algoritmo potrebbe essere:

$p \leftarrow$ punto da aggiungere;
$r \leftarrow$ radice dell'albero;
$N_i \leftarrow$ nodo di cui si vogliono esplorare i figli;
getChildren(Nodo) è una funzione che restituisce in un array i nodi figli di Nodo;
getCluster(Nodo) è una funzione che restituisce i clusteroidi memorizzati in Nodo in un cluster;
$next \leftarrow$ indice del figlio da visitare;
$n[j] \leftarrow$ il figlio j-esimo di N_i;
$N_i \leftarrow r$;
$next \leftarrow 0$;
while N_i *non è foglia* **do**
  $n[j] \leftarrow$ getChildren(N_i);
  next $\leftarrow$ l'indice j che minimizza la $rowsum(p, getCluster(n[j]))$;
  $N_i \leftarrow$ n[next];
end
ricalcola parametri del cluster;
seguendo la strada verso la radice, si ricercano i nodi nei quali compare il clusteroide di tale cluster, e si aggiorna il suo valore;

L'algoritmo organizza i cluster gerarchicamente in un albero, per cui un punto viene assegnato al cluster corretto scendendo di padre in figlio. Ogni

nodo contiene le coordinate del clusteroide, in modo da poter fare il confronto con il punto e fargli prendere la direzione minimizzando la *rowsum*. Si intuisce che scendendo si raggiunge un dettaglio sempre maggiore fino ad arrivare alle foglie. Le foglie contengono le caratteristiche viste sopra dei veri cluster, che dovranno essere aggiornate con l'arrivo del nuovo punto in modo da ripristinare la correttezza delle informazioni.

Tuttavia, se il raggio di un cluster con l'unione di un punto aumentasse troppo, si dovrebbe ricorrere alla divisione dello stesso. Se al contrario i punti in ciascun cluster diminuissero troppo, si potrebbe unirne due. Si noti come queste operazioni si realizzino sempre tra cluster vicini tra loro anche nella rappresentazione ad albero, poiché l'algoritmo GRGPF lavora proprio per mantenere la proprietà di vicinanza anche in tale rappresentazione. Si veda per maggiori dettagli su tali funzioni [7].

Capitolo 3

I Recommendation System

Cominciamo dandone una definizione.

Definizione 3.0.8:
Si definisce un Reccomendation System (in seguito R.S.), una classe di sistemi di *Information filtering* che puntano a scoprire un voto o una preferenza, che un utente assegnerebbe ad un oggetto che non ha ancora considerato, basandosi sulla storia delle interazioni tra l'utente ed altri oggetti.

3.1 I Recommendation System sino ad oggi

I R.S.assunsero una certa rilevanza circa vent'anni fa, quando agli inizi degli anni '90, vennero studiati i primi sistemi a filtro collaborativo. Da allora le industrie ed il mondo accademico spinsero, e lo fanno tutt'oggi, per affinare le tecniche di raccomandazione, data la loro vasta area applicativa.

Essi sono infatti chiamati a fare da filtro tra l'infinita offerta della rete e l'utente, cercando di interfacciarlo con un mondo finito, senza lasciarlo "perdersi" in tale vastità. Tali sistemi si possono vedere applicati in siti come Amazon [3], ove consigliano libri, CD ecc., come MovieLens [4] ove consigliano film, ma sono presenti anche all'interno dei social network, nella proposizione di nuove pagine da preferire, utenti da seguire o con i quali stringere amicizia.

E' interessante osservare che un'importante impulso alla ricerca nei R.S.è stata la sfida di NetFlix [5], la nota società di noleggio DVD e videogiochi via Internet. Questa offrì un milione di dollari alla prima squadra che avesse battuto il loro algoritmo di raccomandazione, il CineMatch, del 10%. Batterlo del significava diminuire di tale percentuale l'RMSE (*Root Mean Squared Error*), uno stimatore dell'errore medio riferito alla differenza tra la vera va-

lutazione e la stimata. Tale sfida, ed il montepremi, fu vinto dopo tre anni, quando due squadre arrivarono al 10.06%.

I R.S.non sono ancora al loro limite, anzi, si intravedono con un po' di fantasia molte applicazioni, quali consigli su talune branche della finanza, su vacanze, o ancora sviluppando shopping card intelligenti. Tutto però ruota intorno al come rappresentare i dati sempre crescenti, come analizzarli ed infine come trovare il consiglio migliore per l'utente.

3.1.1 Problema del *Long tail*

In generale il fenomeno del *Long tail* si manifesta quando si trova un grafico del tipo 3.1.

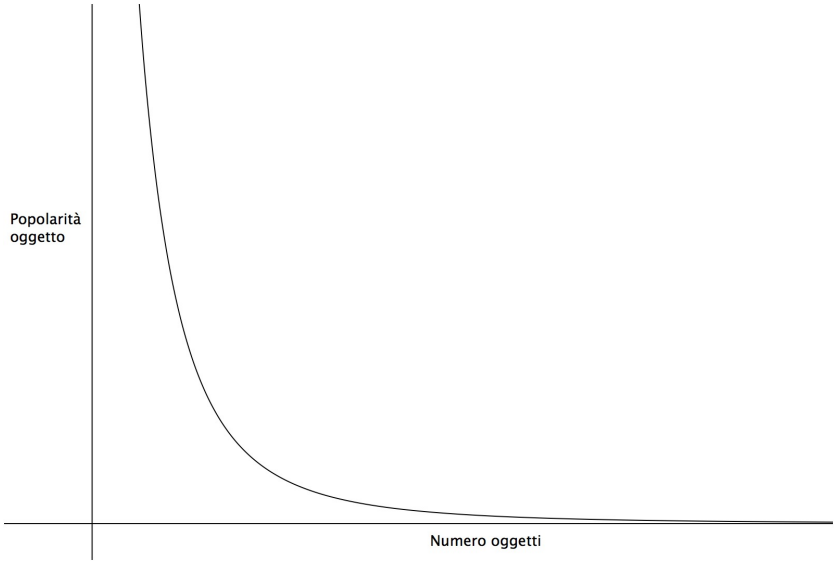

Figura 3.1: Distribuzione del tipo *Long tail*

Nel nostro caso immaginiamo che tale curva sia l'interpolazione dell'istogramma che descrive, per un dato numero di oggett,i la loro popolarità. Perciò verso l'origine ci saranno raggruppamenti di pochi oggetti, che mostrano di essere molto conosciuti, mentre per ascisse crescenti, in coda al grafico, gruppi numerosissimi avranno popolarità quasi nulla.

Questo non è mai stato un grande problema, anzi è un vantaggio nei negozi fisici, come le normali librerie o negozi di dischi. Infatti il gestore dell'ipotetica libreria tiene nel suo negozio solo articoli di grande popolarità, i più venduti, consigliando a chi cercasse qualcosa di specifico di ordinarlo. Nei negozi on-line invece il discorso è diverso: infatti lo store potrebbe presentare tutti gli oggetti sullo stesso piano, senza discernere tra quelli più o

meno popolari. In tale ipotesi un sistema non intelligente potrebbe essere quello di presentare una selezione di 10 elementi presi a caso dallo store. Assumendo che la popolarità di un oggetto sia in qualche modo proporzionale al gradimento dell'utente medio, ipotesi non del tutto irragionevole, possiamo dire che il consiglio migliore per un sito di e-commerce sia quello di mostrare all'utente uno o più oggetti con una popolarità alta. Ma a causa della lunga coda, in realtà ci accorgiamo subito che prendendo un oggetto a caso, esso sarà difficilmente uno dei pochi molto popolari, ma al contrario sarà uno dei tanti sconosciuti.

Allora il compito di un R.S.rudimentale potrebbe essere quello di filtrare gli oggetti in base al loro numero di acquisti e presentare all'utente solo quelli più venduti, che si suppone siano più popolari. Ma la forza di un R.S., ed il nostro obiettivo, è quello di offrire un'esperienza unica all'utente, al quale potremmo presentare non i libri che piacciono in genere, ma i libri che più facilmente possono piacere al singolo. Questo è possibile cercando di ottenere informazioni circa chi sta acquistando, ad esempio osservando i precedenti acquisti, o le preferenze assegnate ad altri oggetti, cercando di capire i gusti e consigliando in modo mirato. E' un po' la differenza tra un pessimo, un buon ed un ottimo libraio: il primo dà un libro a caso in mano al povero acquirente, che probabilmente non lo gradirà. Il secondo propone i libri che vanno per la maggiore, sapendo che di solito piacciono. L'ultimo chiede quali siano gli ultimi libri che ha letto, se gli sono piaciuti, e consiglia in base a tali informazioni qualcosa che, secondo la sua esperienza, potrebbe piacergli. Un buon R.S.dovrebbe fare proprio questo.

3.1.2 Applicazioni dei Recommendation System

I R.S.sono utilizzati in vari campi, facciamo qui di seguito vari esempi:

- siti di $e-commerce$ (*Amazon, eBay, ecc.*) nei quali l'utente può navigare, guardare, informarsi a proposito di svariati prodotti, e comprarli. Ogni visita, ogni click ed ogni acquisto sono indizi che il suo R.S.deve captare ed interpretare per rendere l'esperienza dell'utente più piacevole, personalizzata e quindi ottimizzata, spingendo l'utente a tornare per effettuare altri acquisti.

- siti di servizi (*Youtube, Google, ecc.*) ove l'utente non acquista direttamente oggetti, ma vi interagisce, per cui anche qui è possibile, salvando i suoi spostamenti nel sito, cercare di migliorare la navigazione consigliando, ad esempio, video affini a quelli già visionati.

- siti di informazione, come giornali elettronici o servizi di rss. Anche questi ultimi possono rendere migliore l'interazione con l'utente consigliandogli gli articoli sullo sport preferito, o scritti da un certo giornalista, o comunque proporre diversi ordini per gli articoli del giorno a seconda dei gusti individuali.

- i *social network* (*Facebook, Google+, Twitter,...*) ove l'utente interagisce con un insieme di utenti che possono essere amici o *follower*, e alcuni oggetti, come ad esempio eventi e pagine. Qui il social network potrebbe avere il compito di presentare all'utente nuovi utenti con cui potrebbe voler interagire, nuove pagine da esplorare e nuovi elementi da condividere.

3.2 Esposizione del problema

L'approccio classico allo studio dei R.S.è ricavare i giudizi inespressi di un utente sulla base di quelli immagazzinati nel sistema, i quali possono essere stati raccolti in modo diretto o indiretto. Diretto nel senso che l'utente ha espressamente votato un oggetto, valutandolo ad esempio con 3 su 5. In modo indiretto potrebbe essere ricavare i gusti dell'utente osservando le sue ricerche, gli articoli visionati, ma magari non acquistati. Ognuna di questa informazioni può essere utilizzata per tracciare il profilo dell'utente.

Sia C un insieme di utenti ed S un insieme di oggetti. Notiamo subito che sia C che S possono essere molto grandi, tanti quanti i miliardi di persone nel mondo, i milioni di film girati, o decine di miliardi di canzoni registrate.

Definiamo ora una funzione u, detta *funzione di utilità*, tale che $u : C \times S \to R$, ove R è un insieme totalmente ordinato [9]. Inoltre, per ogni $c \in C$, noi vogliamo trovare un $s' \in S$ che massimizzi la funzione utilità. Ovvero:

$$s'_c = \underset{s \in S}{argmax}\, u(c, s),\ \forall c \in C \tag{3.1}$$

Nei R.S.ci esprimiamo parlando dei valori di $u(c, s)$ come di un giudizio di c su s, ad esempio se a c è piaciuto Harry Potter 5, darà voto 8 (su 10) a tale film. Ogni elemento di C è un profilo, contenente dati personali (nome, età, ecc.) e storico dei voti, o magari semplicemente un codice identificativo. Allo stesso modo ogni elemento di S è un profilo dell'oggetto, con memorizzato un insieme di caratteristiche.

Gli R.S.si possono dividere in due tipi di architetture: i primi sono quelli basati sui contenuti (*Content Based*), i quali si focalizzano sulle proprietà e somiglianze tra gli oggetti. In secondo luogo ci sono gli R.S.a filtro collaborativo (*Collaborative Filtering*), i quali si focalizzano sulle proprietà che

relazionano gli utenti agli oggetti, cercando le somiglianze nelle interazioni di utenti verso gli oggetti.

3.3 Matrice di utilità

In un R.S.ci sono due classi di entità: gli utenti e gli oggetti. Gli utenti hanno delle preferenze in merito ad alcuni tra tutti gli oggetti, e tali dati sono proprio gli elementi della *matrice di utilità*.

Definizione 3.3.1:
Si dice *matrice di utilità* la matrice che in ogni colonna rappresenta un oggetto, in ogni riga un utente. L'elemento (i, j) è il *giudizio* che l'utente i-esimo ha espresso in merito all'oggetto j-esimo.

Tali valori possono essere di vario tipo, a seconda delle applicazioni: interi da 1 a 5, decimali tra 0 e 1, di vario segno, o booleani. I R.S.sono pensati in particolare per applicazioni in cui la matrice risulti sparsa, cioè ogni utente abbia poche opinioni in merito agli oggetti presi in considerazione.

L'obiettivo dei R.S.è di stimare i giudizi mancanti nella matrice di utilità. Una volta determinati quest'ultimi, sarà semplice consigliare un oggetto tra quelli che l'utente non conosce, semplicemente prendendo quello con voto stimato maggiore.

3.3.1 Popolare la matrice di utilità

Senza un'adeguata matrice di utilità è praticamente impossibile effettuare consigli. Inoltre acquisire dati è spesso piuttosto complicato. Per risolvere tale problema si utilizzano due approcci diversi:

- si possono chiedere in modo diretto all'utente dei giudizi su prodotti che conosce, come fa ad esempio Youtube con le sue stelline. Tale approccio ha un'efficacia relativa, dato che gli utenti difficilmente forniscono un feedback su tutto quello che vedono o hanno acquistato. Allo stesso tempo tali dati possono avere un bias dovuto all'umore, al carattere, o comunque a fattori esterni riguardanti l'utente.

- si possono ricavare dati in modo indiretto dal comportamento dell'utente. Se un utente compra un prodotto su Amazon o vede un video su Youtube, possiamo dire che l'oggetto con cui ha interagito gli sia piaciuto. Possiamo utilizzare questi dati mettendo nello spazio interessato

1, e tenere a 0 quelli non visualizzati. Notiamo che si può desumere l'interesse dell'utente verso un oggetto anche solo se ne ha letto la recensione.

3.4 Approccio *Content-based*

Seguiamo ora l'approccio basato sui contenuti.

3.4.1 Profili degli oggetti

In un sistema basato sui contenuti, dev'essere costruito per ogni oggetto un profilo che ne rappresenti le caratteristiche salienti. In casi semplici, come un film, tale dato può essere del tipo:

- elenco degli attori che vi hanno partecipato, per gli utenti che amano certi attori.

- il regista: alcuni utenti amano il modo in cui certi registi raccontano una storia.

- l'anno in cui il film è stato girato: c'è chi apprezza i classici, chi i film più recenti.

- il genere: c'è chi ama le commedie, chi i thriller, chi gli horror,etc.

Notiamo comunque come anche le caratteristiche elencate non siano esaustive per descrivere il film, e come tra l'altro esse siano molto diverse tra loro. Infatti mentre i primi tre sono attributi oggettivi (il cast di un film non cambia a seconda di chi ha visto il film), il quarto dato è in parte vago e soggettivo; un simile problema si può risolvere ad esempio utilizzando l'*Internet Movie Database*, che si occupa di dare un genere a ciascun film.

Una volta capito cosa ci interessa immagazzinare ci manca solo il come. Noi vogliamo arrivare a rappresentare un profilo dell'oggetto ed un profilo utente che esprima le sue preferenze. E' facile capire intuitivamente cosa vogliamo fare pensando ai film: immaginando i dati immagazzinati come booleani, pensiamo ad esempio di avere una componente per ogni attore. Ci sarà un 1 se l'attore recita in un film, uno 0 altrimenti. Tuttavia non in tutte le applicazioni i dati possono essere di natura booleana: a volte bisogna prendere in considerazione insiemi come quello dei razionali. In tal caso dobbiamo stare attenti a considerare nelle distanze il fatto che due dati molto simili, a seconda della scala utilizzata, devono essere considerati uguali.

Si deve anche prestare particolare attenzione se si pensa di utilizzare diversi tipi di dati, pesandoli in modo da non portare il sistema a contraddizioni. Ovviamente questi valori possiamo immaginarli come delle costanti moltiplicative, che devono essere aggiustate con tecniche di stima che si basino sulle osservazioni note.

Otterremo così una matrice che possiamo chiamare *matrice oggetti*.

Definizione 3.4.1:
Si definisce *matrice oggetti* la matrice che ha come colonne degli oggetti e come righe tutte le caratteristiche che possono assumere. Ogni cella (i, j) descrive l'associazione tra l'oggetto i-esimo e la caratteristica j-esima. Spesso i dati usati in questa matrice sono binari, ed esprimono se un oggetto possiede o meno una certa caratteristica.

Anche questa matrice sarà molto grande e sparsa. Infatti nell'esempio dei film avremo migliaia di attori, dei quali solo pochi reciteranno in un dato film, e avranno un 1 in corrispondenza della loro cella.

Utilizzando ora la matrice di utilità, formata come *Utenti* $\times$ *Oggetti* e la matrice oggetti appena definita che lega *Oggetti* $\times$ *Caratteristiche*, con una semplice moltiplicazione matriciale possiamo giungere all'associazione dei gusti dell'utente in merito ad una certa caratteristica. Ad esempio se l'utente A ha dato voti alti a diversi film con *Scarlett Johansson*, si avrà che il sistema con la moltiplicazione sopra citata potrà identificare tale preferenza ed in accordo con la filosofia *content based* gli proporrà un altro film in cui reciti tale attrice. Si prenda quest'ultimo solo come un esempio, visto che un R.S.non raccomanda in base ad una singola caratteristica, bensì utilizzando ad esempio la distanza coseno, come descritto di seguito.

3.4.2 Consiglio di un oggetto ad un utente

Vogliamo descrivere ora un algoritmo che consigli un nuovo oggetto ad un utente utilizzando l'approccio *content based*.

Immaginiamo di avere applicato le tecniche di clustering viste nel Paragrafo 2.2 agli oggetti, in modo da averli organizzati in un albero utilizzando come distanza tra gli oggetti la distanza coseno (2.4). Chiaramente lo spazio degli oggetti che si viene così a creare non sarà euclideo, perciò si dovranno utilizzare tecniche simili a quelle descritte nel Paragrafo 2.2.4, ovvero facendo uso di *clusteoridi*. Avendo i dati così ordinati basterà verificare la distanza tra l'utente ed un numero piuttosto limitato di oggetti, invece di confrontare ogni profilo con tutta la libreria.

Ipotizziamo che il sistema sia a regime, cioè che ci siano già dati organizzati ed immagazzinati e che l'utente che consideriamo non sia al suo primo accesso, in modo da poter lavorare con i dati già immagazzinati nel suo profilo. Avremo allora:

- T un albero n-ario del tipo descritto nell'algoritmo GRGPF, in cui siano immagazzinati tutti gli oggetti della libreria, ordinati utilizzando come misura di distanza la distanza coseno;

- U la matrice di utilità come definita in 3.3.1;

- M la matrice degli oggetti come definita in 3.4.1;

Possiamo allora confrontare le preferenze dell'utente con gli oggetti della libreria per trovarne uno simile al suo profilo, il migliore da consigliare.

$T \leftarrow$ albero degli oggetti;
$U \leftarrow$ la matrice di utilità;
$M \leftarrow$ la matrice degli oggetti;
$i \leftarrow$ il numero identificativo dell'utente, che descrive anche qual è la sua riga nella matrice di utilità;
$u = e_i^T \times U$ è il profilo dell'i-esimo utente;
$p = M \times u$ è il profilo dell'utente che lo lega direttamente alle sue preferenze;
scendi fino alle foglie di T come nell'algoritmo $GRGPF$ fino a trovare il cluster con clusteroide più vicino a p;
esamina il cluster trovato e trova il profilo oggetto con distanza coseno minima da p;
tale oggetto è quello da consigliare;

Nella notazione utilizzata e_i^T è il vettore trasposto del vettore colonna di tutti 0 tranne un 1 nell'i-esimo posto.

3.4.3 Problematiche

Il primo problema di un sistema strutturato come sopra risiede nel numero limitato delle caratteristiche attribuibili agli oggetti, e la corretta attribuzione automatica delle stesse agli oggetti. Senza contare che prende in esame solo in modo secondario l'interazione utente-utente, che invece si rivelerà essere la chiave di volta del metodo a filtro collaborativo.

Tale sistema di raccomandazione pone anche delle notevoli sfide: trovare modi compatti di rappresentare l'informazione, stimare in modo piuttosto

preciso le costanti moltiplicative applicate per poter rendere confrontabili le diverse colonne delle caratteristiche delle matrici. Un'altra problematica è la monotonia delle raccomandazioni: se io ho letto molti romanzi gialli, il sistema tenderà a consigliarmi sempre lo stesso genere, del quale potrei anche stancarmi. A questo allora si può alternare una raccomandazione generica, come uno dei best seller della settimana, o, ancora meglio, una raccomandazione che metta in relazione il mio profilo di appassionato di gialli con un altro giallista, che magari ha scoperto un libro/genere, che può piacere anche a me.

Altro problema che si rivede spesso in sistemi simili è ciò che si definisce la *cold start*: cioè la gestione del nuovo utente. Su di lui infatti, non abbiamo dati che ci permettano di associarlo ad un genere, ad un autore, ecc. Si possono allora utilizzare altri dati, che lui decide di mettere a nostra disposizione: la località da cui si connette, il sesso, ecc., cercando di inserirlo in un profilo medio per cominciare a suggerirgli qualcosa. In questo senso si possono leggere numerosi lavori che propongono di prendere dati da qualsiasi altro account l'utente possa avere, e il seguente modo di interpretarli: dai social network alle utenze di posta elettronica, fino ad arrivare a servizi simili[2].

3.5 Approccio *Collaborative Filtering*

L'approccio a filtro collaborativo (*collaborative filtering*) è un'altra tecnica di raccomandazione. Essa verte sul trovare utenti che hanno dato giudizi simili agli stessi oggetti, creando così un legame tra gli utenti, ai quali verranno suggeriti oggetti che uno dei due ha recensito in modo positivo, o semplicemente con cui ha interagito. In questo modo cerchiamo associazioni tra utenti, e non più tra gli oggetti come accadeva in precedenza.

3.5.1 Misurazione della somiglianza

Si può osservare che le misurazioni di somiglianza o distanza, in questo come in altri casi, risulta molto sensibile ai cambiamenti di misura, o ad aggiustamenti che si possono introdurre. In particolare si può osservare, anche con semplici esempi, che la distanza di Jaccard funziona bene solo in casi in cui sono obbligato ad usarla a causa di dati di tipo booleano, ma che se sono in grado di utilizzare distanze come quella coseno, grazie a dati di tipo intero o razionale, quest'ultima darà risultati più accurati. La perdita di accuratezza della distanza di Jaccard è causata dalla perdita di informazione conseguente alla trasformazione dei dati da razionali a booleani.

Un'altra scelta su cui riflettere è la normalizzazione dei dati: essa può essere fatta spostando in 0 la media di voti (ad esempio da 1-5), in modo da normalizzare i dati senza che il bias introdotto da ciascuno di noi li caratterizzi. In questo modo il sistema analizza solo gli scarti dalla media dei voti dati, che per una persona generosa vorrà dire abbassare i suoi voti, e viceversa per una molto critica.

3.5.2 La dualità della somiglianza

Per spiegare il concetto di dualità in quest'ambito partiamo da alcune semplici considerazioni: la matrice di utilità rappresenta le preferenze di un utente nei confronti di un oggetto, ma, se letta per colonne, evidenzia a chi un certo film è piaciuto o non è piaciuto. In questo modo possiamo vedere come una somiglianza tra due oggetti possa anche esprimersi senza la *matrice oggetti*, semplicemente osservando che i film che sono piaciuti alle stesse persone probabilmente sono in qualche modo simili.

Questo ci porterebbe a pensare di poter utilizzare similarmente le tecniche già viste in precedenza direttamente sulle colonne della matrice di utilità, ipotizzando questa simmetria tra quanto tale matrice dice sia degli utenti che degli oggetti. Tale simmetria tuttavia non è piena, infatti ci sono almeno due fenomeni da evidenziare che in pratica la rompono.

Immaginiamo in primo luogo di aver immagazzinato gli utenti in cluster a seconda della loro distanza, processata ad esempio con la formula della distanza coseno come abbiamo fatto prima con gli oggetti. In questo caso, trovati un certo numero di utenti simili, possiamo analizzare gli elementi valutati complessivamante in modo positivo dal gruppo, per poi raccomandarli a chi di loro non l'abbiano ancora valutati. Abbiamo quindi immagazzinato i dati, in questo caso gli utenti, in un albero come prima avevamo fatto con gli oggetti, tuttavia il procedimento non è affatto lo stesso, poiché in queste ipotesi ci sono dei passi supplementari da apportare, come trovare gli elementi preferiti da gruppo e trovare chi di loro non abbia ancora recensito un oggetto. La dissimilitudine proviene dal diverso approccio da utilizzare con i due tipi di alberi: in quello di oggetti utilizziamo la vicinanza di questi ultimi per permettere all'algoritmo di cercare solo in sottogruppi dell'albero; nell'albero di utenti utilizziamo la loro vicinanza per identificare dei gusti comuni.

In secondo luogo c'è una differenza di fondo nella classificazione dei due concetti di oggetti ed utenti. Infatti mentre i primi sono facilmente classificabili con semplici caratteristiche come genere, anno di pubblicazione, ecc., i secondi possono apprezzare generi diversi, e quindi non si possono sempli-

cemente associare due utenti perché amano entrambi il rock anni '70, perché ognuno di loro ha probabilmente interessi diversi in molti altri campi.

3.5.3 Come creare ed usare i cluster di utenti e oggetti

Un problema che si incontra studiando i R.S.è l'avere a che fare con una matrice sparsa: in essa le celle sono quasi tutte vuote. In questo caso si possono usare le tecniche di clustering viste nel Paragrafo 2.2, avvalendosi di una delle distanze definite nel Paragrafo 2.1.1, per riassumere gruppi di utenti e gruppi di oggetti in un unico utente o oggetto, creando quindi una matrice di utilità *compatta*. In questo modo si utilizzano i dati del gruppo per riempire le celle della matrice estesa ed utilizzare tali voti per le raccomandazioni. Succedesse che anche la matrice compatta avesse delle celle vuote potrebbe iterare il processo e derivarle da una matrice ancora più riassuntiva.

3.5.4 Problematiche

L'approccio a filtro collaborativo e quello basato sui contenuti hanno in comune diversi problemi. Il primo è quello del nuovo utente. La prima raccomandazione può essere fatta solo basandosi su criteri generici quali genere, età, posizione geografica, ecc., consigliando un oggetto che in genere gode di grande popolarità, ma per nulla personalizzato.

Un altro problema che l'approccio collaborativo condivide con il sistema basato sui contenuti è la sparsità delle informazioni: ci sono moltissimi oggetti, e ogni utente darà solo qualche opinione, sicuramente non confrontabile con il numero di giudizi da prevedere.

Un altro problema che l'approccio a filtro collaborativo condivide con il sistema basato sui contenuti è il numero molto limitato di informazioni disponibili. Infatti, confrontando il numero ridotto di dati con quello enorme dei campi ignoti, si capisce come sia complesso il lavoro di un R.S.. Si possono allora includere dati anagrafici, quelli utilizzati nella prima raccomandazione, per arricchire il profilo utente.

Un terzo problema comune con l'approccio *content based* è quello della *super-specializzazione* dei consigli. Infatti anche quì si rischia di non consigliare mai nulla che esuli dal genere preferito del gruppo, il che può rendere i consigli del sistema noiosi e ridondanti. Tuttavia le raccomandazioni di un sistema a filtro collaborativo, cercando similitudini tra gli utenti e non tra gli oggetti, introducono per loro natura una maggiore variabilità, essendo, come già osservato, gli interessi di ogni utente sicuramente molteplici. In questo modo se un gruppo gradisce qualcosa di estraneo alle categorie che recensisce

generalmente, questo oggetto potrebbe essere raccomandato in un sistema a filtro collaborativo, cosa che non accadrebbe nell'altro approccio.

Un nuovo problema che invece si presenta nel *collaborative filtering* è quello del nuovo oggetto. Infatti nell'approccio *content based* un oggetto viene inserito nel sistema, descrivendolo, e quindi non esiste differenza tra un oggetto nuovo e uno che non lo è nella matrice degli oggetti, poiché tutti sono descritti allo stesso modo. Nella matrice di utilità invece questa differenza si nota, essendo un oggetto nuovo ovviamente non ancora recensito da nessuno: per questoi non potrà essere consigliato essendo lontano da qualsiasi oggetto. Questo problema può essere aggirato utilizzando un R.S.*ibrido*: un approccio che si avvalga a volte del sistema basato sui contenuti, a volte di quello collaborativo. In questo caso l'approccio basato sui contenuti si rivela indispensabile per cominciare da subito a raccomandare il nuovo oggetto, che altrimenti rimarrebbe senza voti per molto tempo.

Il secondo problema caratteristico di questo approccio rispetto al *collaborative filtering* è quello dell'utente isolato. Infatti nel caso in cui un utente sia fuori da qualsiasi gruppo, cioè la sua distanza dagli altri utenti sia sempre relativamente alta, non gli si possono fare raccomandazioni di alcun tipo, perché il suo profilo rende impossibile il confronto con i gusti di chiunque altro. Anche in questo caso potrebbe essere utile un sistema ibrido, che in tal caso potrebbe consigliare oggetti simili a quelli graditi dall'utente, utilizzando l'approccio basato sui contenuti[1].

3.6 Riduzione della dimensione

Un altro approccio che si può adottare è quello della riduzione della dimensione (*dimensionality reduction*), nel quale la matrice di utilità viene scomposta come prodotto di matrici. In realtà quella che viene scomposta non è tutta la matrice, ma una sottomatrice piuttosto piena, che chiameremo *scelta*. L'idea è selezionare utenti con interessi simili, che abbiano quindi molti voti negli stessi oggetti, ignorando gli elementi che pochi o nessuno di loro ha recensito. Tale metodologia può anche essere applicata ad esempio alla matrice riassuntiva dello scorso paragrafo, che si contraddistingue per avere quasi tutte le celle piene. La riduzione della dimensione opera approssimando la matrice di utilità *scelta* con il prodotto di altre due matrici. Queste avranno solo una dimensione libera, poiché le altre due sono vincolate alla matrice originaria. La dimensione libera va comunque scelta in modo che il numero di incognite sia uguale, o di poco inferiore, al numero di elementi noti nella matrice originale. Sia allora M una matrice $m \times n$, che vogliamo decomporre in due matrici $U \times V$, ove U ha dimensioni $m \times d$ e V ha dimensione $d \times n$.

d deve essere scelto in modo che se sono note $m \cdot n - i$ entry nella matrice M, valga la relazione:

$$m \cdot n - i \geq m \cdot d + d \cdot n = d \cdot (m + n) \tag{3.2}$$

Un esempio potrebbe essere $M = U \times V$

$$\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{bmatrix} \times \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{bmatrix} \tag{3.3}$$

in cui abbiamo 20 incognite e 23 entry. Ovviamente i valori delle incognite vanno approssimati in modo da ridurre l'errore, a questo proposito utilizziamo il già citato RMSE (*root-mean-square error*). Immaginiamo che ci venga data una prima approssimazione, che noi dobbiamo migliorare. Cercheremo di migliorare un elemento alla volta delle matrici U e V, il che vuol dire lavorare su una riga o una colonna della matrice risultante. Si sommerà una variabile x al valore precedentemente attribuito ad una casella $u_{i,j}$ e il nostro scopo sarà trovare la x che minimizza l'RMSE di una riga (o colonna). Non riportiamo i calcoli che si possono trovare all'interno del libro di riferimento [7].

Capitolo 4

Applicazione al suggerimento di nuovi utenti su *Facebook*

Con le nozioni sviluppate nei capitoli precedenti è semplice creare una funzione di suggerimento di un nuovo amico in *Facebook*. Infatti, definito lo spazio, la matrice di utilità e la distanza, gli algoritmi visti in precedenza permettono in modo efficace di trovare le migliori raccomandazioni.

Cominciamo sottolineando il campo in cui vogliamo lavorare: supponiamo infatti che la rete sociale sia già sviluppata, con un certo numero di utenti al suo interno che abbiano già diverse interazioni tra loro. Vogliamo ora sviluppare un R.S.su tale rete che possa suggerire ad un utente un altro utente della rete. Pensiamo di svilupparlo con approccio a filtro collaborativo.

Serve allora creare la matrice di utilità (Definizione 3.3.1): essa avrà come righe gli utenti di *Facebook*, e come colonne gli elementi da consigliare. Quest'ultimi possono essere di diverso genere, ma essenzialmente possiamo dividerli in due grandi famiglie: quelli che sono *utenti* e quelli che non lo sono, che chiamiamo genericamente *pagine*. Ipotizziamo di popolare tale matrice con elementi binari, che indichino se c'è o meno una relazione tra due utenti o tra un utente ed una pagina.

	u_1	u_2	...	u_n	p_1	...	p_l
u_1	–	0	–	1	0	–	1
u_2	0	–	–	0	0	–	0
...	–	–	–	–	–	–	–
u_n	1	0	–	–	1	–	1

Si può facilmente osservare che in questo caso la sotto-matrice degli utenti è simmetrica, proprietà che ci consente di utilizzare meno memoria per salvarla.

Per la funzione di distanza ci serviamo della distanza coseno modificata. Infatti vogliamo, ad esempio, che l'amicizia tra due utenti valga di più del *like* di un utente verso una pagina. Per questo motivo pre-moltiplichiamo i valori afferenti alle pagine per $\alpha \in [0; 1]$. Per cui la funzione sarà del tipo:

$$cos(u, v) = \frac{\sum_{i=1}^{n} v_i \cdot u_i + \sum_{i=n+1}^{n+l} \alpha v_i \cdot \alpha u_i}{\sqrt{\sum_{i=1}^{n} v_i^2 + \sum_{i=n+1}^{n+l} \alpha v_i^2} \cdot \sqrt{\sum_{i=1}^{n} u_i^2 + \sum_{i=n+1}^{n+l} \alpha v_i^2}} \tag{4.1}$$

ove come nella matrice sopra le componenti dei vettori u e v, righe della matrice di utilità, hanno n componenti relative agli utenti e le seguenti l relative alle pagine.

Ora si devono trovare i gruppi in cui cercare gli utenti da consigliare, non volendo processare la distanza per ogni coppia di utenti in *Facebook*, che richiederebbe un tempo quadratico nel numero di utenti, deci eccessivo. Una valida idea potrebbe essere quella di diversificare la ricerca. Un gruppo potrebbere+

essere quello degli amici degli amici, quindi tutti i nodi del grafo a distanza 2 dall'utente. Tale compito ha costo quadratico nel numero di amici, il che non è eccessivo essendo il numero medio di amici di circa 130 [8]. Un altro gruppo in cui cercare utenti da consigliare potrebbe essere quello di un evento al quale l'utente ha partecipato.

Al di là della distanza potremmo imporre in talune situazioni, di proporre un utente come consiglio per un nuovo amico. Una di queste situazioni è ad esempio quella in cui due utenti compaiono nella stessa foto: c'è una buona probabilità che tali persone si siano conosciute e vogliano stringere amicizia anche sul *social network*.

Riassumiamo tutto in un breve pseudocodice.

Un approccio un po' più vicino alla realtà è quello di utilizzare dei numeri razionali per rappresentare un indice di vicinanza tra gli utenti, invece dei soli $\{0, 1\}$. In tal caso definiamo nulla la funzione nel caso manchi la relazione di amicizia. Se invece due utenti sono amici, cerchiamo di capire quanto lo siano, utilizzando una funzione che deve essere definita appositamente. Sicuramente possiamo azzardare che in tale funzione debbano avere un ruolo, ad esempio, i post pubblicati nelle rispettive bacheche. Questi accorgimenti sono applicabili non solo tra gli utenti, ma anche tra la relazione di *like* ad una pagina ed un utente: la funzione sarà nulla se non c'è la relazione di like, altrimenti potrebbe avere valore diverso a seconda delle interazioni tra l'utente e la pagina.

ProposeFriends(u,r)
Data: u utente a cui consigliare nuovi possibili amici;
r massimo numero di amici da consigliare;
$fof[] \leftarrow$ amici a distanza 2 da u senza ripetizioni e senza u; {amici degli amici}
if $r \leq size(fof)$ **then**
  return fof;
end
$P \leftarrow$ min-priority-queue di elementi $(key, value)$;
for $i \leftarrow 1$ *to* $size(fof)$ **do**
  if u *compare in una foto con* $fof(j)$ **then**
    $P.add(0, fof(j))$;
  end
  else
    if $!u.isFriendOf(fof(i))$ **then**
      $P.add(cos(u, fof(i)), fof(i))$;
    end
  end
end
return i primi r elementi di P;

Capitolo 5

Conclusioni

Come già espresso, questa tesina triennale è stata sviluppata a partire dal testo *Mining of Massive Datasets* [7], approfondendo e rielaborandone i concetti fondamentali utili alla comprensione dei R.S.. Si è voluto concludere il lavoro con un breve capitolo originale, nel quale sono presentate alcune idee su come le conoscenze acquisite potrebbero portare alla realizzazione di un sistema di raccomandazione di amici su un social network. Tuttavia la correttezza di tali scelte non è assoluta, poiché pur essendo basate su una solida teoria, è assente una validazione empirica. Per questo un interessante prosieguo di questo lavoro potrebbe essere l'implementazione di un piccolo social network. L'algoritmo da me proposto potrebbe essere utilizzato, in tale contesto, per consigliare nuove amicizie, ed in seguito ottimizzato osservando quali suggerimenti si rivelino efficaci.

Bibliografia

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

[2] Judy Kay Amit Tiroshi, Tsvi Kuflik and Bob Kummerfeld. Recommender systems and the social web. *Lecture Notes in Computer Science*, 7138:60–70, 2012.

[3] Jeff Bezos. Amazon, 1994.

[4] Minnesota University GroupLens Research. Movielens, 1997.

[5] NetFlix. Netflix, 1997.

[6] Sebastiano Vigna Paolo Boldi. Four degrees of separation, really. disponibile su `http://arxiv.org/format/1205.5509v1`, 2012.

[7] A. Rajaraman and J.D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

[8] Wikipedia. Facebook, 2005.

[9] Wikipedia. Total order, 2008.

[10] Mark Zuckerberg. Facebook, 2004.