UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Matematica e Informatica

Corso di Laurea Magistrale in Informatica

Rosario Scalia

(1000008648)

Progetto di Big Data

Anno Accademico 2019 - 2020

Indice

Capitolo 1

Introduzione

Negli ultimi anni, il trend della profilazione Online degli Utenti sta crescendo sempre di più in una moltitudine di settori.

Tutto ciò, è giustificato dalla possibilità di estrapolare insights realmente utili a favore dei gestori delle piattaforme che ospitano gli utenti.

In tale contesto, un'applicazione di rilievo è la *Raccomandazione*.

Tale task, si prefigge l'obbiettivo di suggerire nuovi prodotti ,*in maniera intelligente* , agli Utenti; ossia suggerire dei prodotti che *verosimilmente* incontreranno i Loro gusti.

1.1 Descrizione Task

In generale, la computazione associata ai Sistemi di Raccomandazione si snoda in due fasi:

1. **Raccolta Dati**
2. **Inferenza Preferenze Utenti**

Nella fase di *Raccolta Dati*, vengono collezionate le preferenze espresse degli Utenti sui Prodotti.

Quest ultime possono appartenere a due categorie: Esplicite o Implicite.

Nel caso *Esplicito*, viene chiesto all'Utente di esporre la sua Preferenza su un contenuto, magari attraverso la scelta di uno score.

Nel caso *Implicito*, il task si complica dato che è necessario mettere in atto una strategia che ricavi le suddette preferenze a partire dal comportamento degli Utenti sulla piattaforma.

In tutti i casi, le preferenze vengono sempre archiviate in una matrice Utenti-Prodotti, chiamata *Utility Matrix*.

Un ulteriore tassello importante dei Sistemi di Raccomandazione è la capacità di inferire il sentimento degli Utenti sui Prodotti,

In tale contesto, esistono due approcci comuni in letteratura:

- **Content-Based**
- **Collaborative Filtering**

Nell'approccio *Content-Based*, viene costruito un Profilo dell'Utente sulla base dei prodotti da lui valutati.

Successivamente, si passa al calcolo della raccomandazione sfruttando un criterio di *Massima Similarità* fra il profilo dell'Utente e una serie di Profili di Prodotti che quest'ultimo non ha ancora acquistato.

Invece, nell'approccio *Collaborative Filtering* si raccomandano i prodotti agli Utenti sulla base delle valutazioni di Utenti Simili all'Utente per il quale si vuole fare Raccomandazione.

1.2 Difficoltà del Task

La computazione indotta dai Sistemi di Raccomandazione porta in dote una serie di difficoltà che rendono il task challenging.

Una fra le principali è riconducibile alla enorme mole di dati raccolti, tutto ciò ,di riflesso, implica la scelta accurata dell'infrastruttura hardware/software che andrà ad ospitare gli algoritmi.

Un'ulteriore problematica è legata alla *qualità* delle preferenze raccolte nel caso Implicito.

Del resto, tali preferenze sono raccolte tracciando il comportamento degli Utenti e per tanto presentano un certo grado di *distorsione* rispetto alle corrispettive preferenze Esplicite.

1.3 Scopo del Progetto

Il seguente progetto si prefigge l'obbiettivo di confrontare una serie di algoritmi di raccomandazione impiegando un dataset avente una cospicua quantità di dati.

Nello specifico, gli algoritmi confrontati sono elencati di seguito:

- **SVD-LFM**, in tale algoritmo viene impiegata la decomposizione a valori singolari dell'Utility Matrix per rappresentare gli Utenti e i Prodotti nello *Spazio delle Dimensioni Latenti*.

  Fatto ciò, si sfrutteranno le *proprietà del suddetto Spazio* per inferire le preferenze degli Utenti sui Prodotti.

- **LSH-Reccomendation**, l'LSH-Reccomendation è un algoritmo di raccomandazione che esibisce ottime prestazioni anche in presenza di dati *ad elevata dimensionalità*.

  Tutto ciò, in virtù dell'impiego del *Localitive-Sensitive Hashing (LSH)* [4] [8].

  Dal punto di vista dell'inferenza, l'algoritmo impiega un approccio *Collaborative-Filtering* su di una serie di cluster ottenuti dal suddetto LSH.

- **ALS-LFM**, in tale algoritmo viene impiegato un processo di learning per apprendere una *buona rappresentazione* degli Utenti e dei Prodotti nello *Spazio delle Dimensioni Latenti*.

  Fatto ciò, si sfrutteranno le proprietà di tale Spazio per inferire le preferenze degli Utenti sui Prodotti.

Capitolo 2

Algoritmi

2.1 SVD-LFM

L'algoritmo SVD-LFM (Latent Factors Model) è un particolare algoritmo di collaborative-filtering che sfrutta il concetto di *Fattorizzazione di Matrici* per fornire le raccomandazioni agli Utenti.

Nello specifico, l'algoritmo presenta due importanti fasi:

1. Rappresentazione delle Entità (Utenti-Prodotti) all'interno dello Spazio a Dimensioni Latenti, per tale scopo viene impiegata la *Decomposizione a Valori Singolari (SVD)* dell'Utility Matrix.

2. Calcolo Raccomandazione sfruttando le proprietà del suddetto Spazio.

2.1.1 SVD-Embedding

La Decomposizione a Valori Singolari (SVD), è una tecnica di decomposizione di matrici avente le seguenti caratteristiche:

- Sia $M \in \mathbb{R}^{m \times n}$ una matrice avente m righe e n colonne; quest'ultima è assimilabile ad un'Utility Matrix contente i rating di m Utenti e n prodotti.

- Allora, la decomposizione SVD permette di scomporre la matrice M in tre matrici *a più bassa dimensionalità* nel seguente modo:

$$M = U \cdot \Sigma \cdot V^T$$

$$\begin{aligned} M &\in \mathbb{R}^{m \times n} \\ U &\in \mathbb{R}^{m \times r} \\ \Sigma &\in \mathbb{R}^{r \times r} \\ V^T &\in \mathbb{R}^{r \times n} \end{aligned} \tag{2.1}$$

- La Matrice **U** permette di rappresentare gli Utenti nello Spazio delle Dimensioni Latenti.

Dal punto di vista Matematico, quest'ultima è una Matrice avente Colonne *Ortonormali*.

- La Matrice $\mathbf{V^T}$ permette di rappresentare i Prodotti nello Spazio delle Dimensioni latenti

  Dal punto di vista Matematico, $\mathbf{V^T}$ è una Matrice avente Righe *Ortonormali*.

- La Matrice $\mathbf{\Sigma}$ permette di quantificare la *forza dei fattori latenti* all'interno della Matrice Originale M.

  Dal punto di vista Matematico, $\mathbf{\Sigma}$ è una Matrice *Diagonale* dove ogni elemento delle diagonale viene chiamato *Valore Singolare* e permette di quantificare la forza di uno specifico Fattore Latente all'interno della Matrice M.

- La decomposizione SVD di una Matrice M è la decomposizione che *minimizza* la norma di *Frobenius* dell'errore di ricostruzione, ovvero:

$$\begin{aligned} M &= \text{Matrice Originale} \\ M' &= SVD(M) = U \cdot \Sigma \cdot V^T \\ M' &\longrightarrow \min ||M - M'|| \end{aligned} \tag{2.2}$$

Dal punto di vista semantico, possiamo dire che l'SVD riesce ad estrapolare le relazioni *implicite* fra le Entità presenti nella Matrice Originale.

Ad esempio, nel caso di un'Utility Matrix di Film è ragionevole pensare che i Film e gli Utenti possano essere implicitamente legati dal *Genere dei Film*.

Dal punto di vista Matematico, possiamo asserire che l'SVD è strettamente correlata al concetto di *Auto-Decomposizione di una Matrice*.

Nello specifico, è possibile calcolare l'SVD attraverso l'auto-decomposizione della Matrice di Gram ($M^T \cdot M$) ottenuta dalla Matrice Originale.

Quest'ultimo fatto, pone dei *limiti* sul Numero di Fattori Latenti *Estraibili*.

Del resto, tale valore è limitato dal rango della suddetta Matrice di Gram.

Per tanto, possiamo concludere che il rango della matrice di Gram *codifica* il numero di fattori latenti presente nella matrice Originale.

2.1.2 Logica di Raccomandazione

Una volta che Utenti e Prodotti sono stati mappati nello Spazio delle Dimensioni Latenti, è possibile effettuare la raccomandazione nel seguente modo:

- Sia $k \in \mathbb{R}^n \wedge k \neq 0^T$ il record dell'Utente per cui vogliamo fare raccomandazione.

- Sia $M \in \mathbb{R}^{m \times n}$ l'Utility Matrix e sia $M = U \cdot \Sigma \cdot V^T$ la sua Decomposizione a Valori Singolari.

- Allora, è possibile effettuare la raccomandazione sfruttando i seguenti passi:

1. Rappresentazione dell'Utente all'interno dello Spazio dei Fattori Latenti:
$$k^{(trasf)} = k \cdot V \tag{2.3}$$
2. Rappresentazione dell'Utente all'interno dello Spazio dei Prodotti:
$$recc = k^{(trasf)} \cdot V^T \tag{2.4}$$

- Il vettore finale (*recc*) rappresenta la raccomandazione per l'Utente k.

2.2 LSH-Reccomendation

L'algoritmo *LSH-Reccomendation* è un algoritmo di raccomandazione basato sul Localitive-Sensitive Hashing [4] [8].

Nello specifico, l'LSH viene impiegato per costruire una *clusterizzazione* dei Prodotti presenti nel dataset.

Una volta costruiti i suddetti cluster, verrà impiegato un approccio di tipo *Collaborative-Filtering* per inferire le preferenze degli Utenti sui Prodotti.

2.2.1 Localitive-Sensitive Hashing (LSH)

Il Localitive-Sensitive Hashing (LSH) è una tecnica che permette ,attraverso l'utilizzo dell'hashing, di clusterizzare i punti di uno spazio vettoriale in gruppi di punti *contigui* dal punto di vista della metrica vigente nel suddetto spazio.

Dal punto di vista matematico, quanto detto è equivalente alla seguente formulazione:

$$\begin{aligned}
\phi(X, m) &= \text{Generico Spazio Metrico} \\
X &= \text{Insieme di Punti appartenenti a } \phi \\
m &= \text{Metrica vigente su } \phi \\
\mathcal{F} &= \text{Insieme di Funzioni Hash Localitive-Sensitive} \\
\mathcal{F} &= \{h_i : X \longrightarrow S \mid i \in \{1, 2, \ldots, q\} \wedge S = \{0, 1, \ldots, r-1\} \wedge q, r \in \mathbb{N}\} \\
R &= \text{Soglia di } \textit{Vicinanza} \text{ sulla Metrica } m \\
\exists\, (R, c) \in \mathbb{R} \times \mathbb{R} &: R > 0 \wedge c > 1 \\
\Gamma &= \text{Oggetto che estrae una funzione da } \mathcal{F} \text{ con probabilità Uniforme} \\
\exists\, (P_1, P_2) \in [0, 1] \times [0, 1] &: P_1 > P_2
\end{aligned}$$

$$\begin{aligned}
\forall (a, b) \in X \wedge \forall h &: h = \Gamma(\mathcal{F}) \\
&\Rightarrow m(a, b) \leq R \rightarrow P(\, h(a) = h(b)\,) \geq P_1 \\
&\Rightarrow m(a, b) \geq cR \rightarrow P(\, h(a) = h(b)\,) \leq P_2 \\
&\Rightarrow \mathcal{F} \text{ è una famiglia di funzioni } (R\,,\, cR\,,\, P_1\,,\, P_2)\text{-sensitive}
\end{aligned} \tag{2.5}$$

Per tanto, l'LSH fornisce un ottimo framework per la risoluzione efficiente di numerosi task come la *Nearest Neighbor Search* o la *Riduzione della Dimensionalità*.

2.2.2 BucketRandomProjection

Nel corrente progetto, è stata impiegata una famiglia LSH associata alla *Distanza Euclidea*.

Nello specifico, è stata impiegata la famiglia *BucketRandomProjection*.

Quest'ultima è una famiglia di funzioni che ,*rispettivamente*, prendono in input un punto dello Spazio e lo proiettano su una retta *random* appartenente a tale Spazio, che a sua volta è suddivisa in k porzioni/bucket.

Alla fine del calcolo, l'output sarà *il bucket* su cui giace la proiezione.

Tutto ciò, è modellato matematicamente dalla seguente formulazione matematica:

$$\begin{aligned} \Psi &= \text{Funzione estratta dalla famiglia LSH BucketRandomProjection} \\ \eta &= \text{Vettore Random appartenente allo Spazio Euclideo considerato} \\ x &= \text{Punto dello Spazio Euclideo che si vuole clusterizzare} \\ w &= \text{Grandezza Bucket} \\ hashed\_x &= \Psi(x) = \left\lfloor \frac{x \cdot \eta}{w} \right\rfloor \end{aligned} \tag{2.6}$$

2.2.3 Logica di Raccomandazione

La logica di raccomandazione dell'LSHReccomendation si snoda nelle seguenti due fasi:

- **Clustering dei Prodotti**, in tale fase verrà impiegata la famiglia di funzioni *BucketRandomProjection* per *clasterizzare* i Prodotti presenti nell'Utility Matrix.
- **Raccomandazione**, nella fase di raccomandazione è prevista la stima del rating y dell'utente x sul prodotto z attraverso la media dei rating dei prodotti presenti nel bucket di z valutati dall'Utente x.

Di seguito, viene mostrato lo pseudo-codice dell'algoritmo proposto:

Algorithm 1: LSH-Reccomendation

Input: usr_2_prod = <U , P> , utilityMatrix = $M \in \mathbb{R}^{m \times n}$,
clusters = `array<`**set**`>`

Output: usr_2_prod = <U , P , R = $r \in \{1, 2, 3, 4, 5\}$>

foreach $(u, p) \in usr\_2\_prod$ **do**

rated_prod = *utilityMatrix*.`filter` ("prodotti valutati da u")

p_cluster = *clusters*.`findByElement` (p)

voters = *rated_prod*.`intersect` (p_cluster)

if *voters*.`length` > *0* **then**
$(u, p) \longrightarrow (u, p, \underline{avg(voters)})$
else
$(u, p) \longrightarrow (u, p, \underline{avg(rated\_prod)})$

end

2.2.4 Aspetti Computazionali

L'algoritmo mostrato nella sezione precedente rappresenta una semplificazione rispetto a quello realmente implementato; tutto ciò, per *ragioni illustrative*.

Inoltre, è da sottolineare che un'implementazione *puntuale* di **1** implicherebbe una complessità lineare rispetto alle dimensioni dell'input, $O(n)$.

Tutto ciò, comporterebbe delle prestazioni di runtime sempre più scadenti al crescere della dimensione dei dati di input.

Per tanto, l'algoritmo implementato include una serie di ottimizzazioni che vanno a sfruttare le capacità di *parallelismo* offerte dalle moderne CPU *Multi-Core*.

Grazie a tali ottimizzazioni, l'algoritmo riesce a raggiungere la seguente complessità:

$$\begin{aligned} n\_cpu &= \text{Numero di CPU del Cluster} \\ cpu\_core &= \text{Numero di Core di ogni CPU} \\ < n\_cpu\,,\, cpu\_core > &= n\_cpu \cdot cpu\_core \\ complessita\_algoritmo &= O\left(\frac{n}{< n\_cpu\,,\, cpu\_core >}\right) \end{aligned} \tag{2.7}$$

È da sottolineare, come la complessità continui ad essere dipendente dalla cardinalità del dataset di input.

Ciò nonostante, la formula ci dice che possiamo *bilanciarla* con l'incremento del numero di macchine e/o dei core della CPU.

Dal punto di vista *analitico*, quanto detto equivale al seguente limite:

$$\lim_{< n\_cpu\,,\, cpu\_core > \to \infty} \frac{n}{< n\_cpu\,,\, cpu\_core >} = 0 \tag{2.8}$$

2.3 ALS-LFM

L'algoritmo *Alternative Least Square-Latent Factors Model (ALS-LFM)* [3] è un algoritmo di Raccomandazione che sfrutta il già citato concetto di *Fattorizzazione dell'Utility-Matirix* per generare le raccomandazioni.

Nello specifico, le fasi principali dell'algoritmo sono illustrate di seguito:

1. Generazione di una *buona rappresentazione* degli Utenti e dei Prodotti all'interno dello Spazio delle Dimensioni Latenti.

   Per ottenere la *suddetta rappresentazione*, verrà impiegato un processo di learning *iterativo*.

2. Utilizzo delle proprietà del *nuovo spazio* per generare le raccomandazione.

2.3.1 ALS-Embedding

Come accennato in precedenza, la mappatura degli Utenti/Prodotti nello Spazio a Dimensioni Latenti avviene attraverso un processo di learning.

Quest'ultimo esibisce le seguenti caratteristiche:

- **Modello**, il Modello associato all'ALS-LFM è rappresentato da due Matrici che mappano Utenti e Prodotti nello Spazio a Dimensioni Latenti, ovvero:

$$U = \begin{bmatrix} u_{00} & u_{01} & \dots & u_{0f} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ u_{n_u 0} & u_{n_u 1} & \dots & u_{n_u f} \end{bmatrix}, M = \begin{bmatrix} m_{00} & m_{01} & \dots & m_{0f} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ m_{n_m 0} & m_{n_m 1} & \dots & m_{n_m f} \end{bmatrix} \tag{2.9}$$

  Come si può intuire dalla formulazione Matematica, la matrice U permette di rappresentare gli Utenti nello Spazio Latente mentre la matrice M permette di eseguire lo stesso compito ma con i Prodotti.

  Chiaramente, il numero di fattori latenti f è un iperparametro dell'algoritmo.

- **Dataset**, il dataset adibito al fitting del modello ALS è rappresentato da un set di terne $< Utente, Prodotto, Rating >$, ovvero:

Utente	Prodotto	Rating
...	...	...
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
...	...	...

Tabella 2.1: Schema del dataset per l'algoritmo ALS-LFM

- **Funzione Costo**, l'obbiettivo dell'Ottimizzazione è illustrato di seguito:

$$
\begin{aligned}
J(U,M) &= \sum_{\forall (i,j,r) \in \mathbb{D}} (r - u_i^T \cdot m_j)^2 + \lambda \cdot \left(\sum_i n_{u_i} \cdot ||u_i||^2 + \sum_j n_{m_j} \cdot ||m_j||^2 \right) \\
\mathbb{D} &= \text{Dataset} \\
u_i &= \text{i-esima colonna della matrice U} \\
m_j &= \text{j-esima colonna della matrice M} \\
r &= \text{k-esimo Rating del Dataset} \\
\lambda &\in [0,1] \\
\lambda &= \text{Parametro di Regolarizzazione} \\
n_{u_i} &= \text{Numero Rating dati dall'Utente i} \\
n_{m_j} &= \text{Numero Rating ricevuti dal Prodotto j}
\end{aligned}
\tag{2.10}
$$

- **Learning**, la procedura di learning che permette di minimizzare J sfrutta una successione di *Applicazioni Alternate dei Minimi Quadrati*.

  Più precisamente, le fasi principali dell'algoritmo sono riassunte di seguito:

  1. Inizializza la Matrice M assegnando alla prima riga la media dei Rating del primo Prodotto e in tutte le altre dei numeri random piccoli
  2. Fissando M, minimizza J rispetto a U attraverso la risoluzione di una serie di problemi *ai Minimi Quadrati* in quantità pari al numero di colonne di U (ovvero al numero di fattori latenti).
  3. Fissando U, minimizza J rispetto a M attraverso la risoluzione di una serie di problemi *ai Minimi Quadrati* in quantità pari al numero di colonne di M (ovvero al numero di fattori latenti).
  4. Cicla a *2* fin quando non si raggiunge un criterio d'arresto.

A questo punto della trattazione, verrà dato un esempio di come ricavare una generica colonna u_i di U al passo *2* della procedura appena mostrata.

È da sottolineare che tale procedura è speculare nel caso opposto, ovvero quando dobbiamo ricavare una colonna m_i di M al passo *3* della suddetta procedura.

Detto ciò, per ricavare una generica colonna u_i di U è necessario risolvere un problema *lineare ai Minimi Quadrati* sfruttando i rating dati dall'Utente i e i corrispettivi Prodotti m_j.

Dal punto di vista matematico, si andranno ad effettuare i seguenti passi:

$$\frac{1}{2} \cdot \frac{\partial J}{\partial u_{ki}} = 0 \quad \forall i, k$$

$$\begin{aligned} I_i &= \text{Set di Prodotti valutati dall'Utente i} \\ r_{ij} &= \text{Rating dato dall'Utente i sul Prodotto j} \\ f &= \text{Numero di Fattori Latenti} \end{aligned}$$

$$\begin{aligned} &\Rightarrow \sum_{j \in I_i} (u_i^T \cdot m_j - r_{ij}) \cdot m_{kj} + \lambda \cdot n_{u_i} \cdot u_{ki} = 0 \quad \forall i, k \\ &\Rightarrow \sum_{j \in I_i} (m_{kj} \cdot m_j^T \cdot u_i) + (\lambda \cdot n_{u_i} \cdot u_{ki}) = \sum_{j \in I_i} m_{kj} \cdot r_{ij} \quad \forall i, k \end{aligned}$$

$$\begin{aligned} M_{I_i} &= \text{Sottomatrice di M avente i Prodotti valutati dall'Utente i} \\ E &= \text{Matrice Identica } f \times f \\ R(i, I_i) &= \text{i-esima riga dell'Utility-Matrix filtrata rispetto ai Prodotti valutati da i} \end{aligned}$$

$$\Rightarrow (M_{I_i} \cdot M_{I_i}^T + \lambda \cdot n_{u_i} \cdot E) \cdot u_i = M_{I_i} \cdot R^T(i, I_i) \quad \forall i$$

$$\begin{aligned} A_i &= M_{I_i} \cdot M_{I_i}^T + \lambda \cdot n_{u_i} \cdot E \\ V_i &= M_{I_i} \cdot R^T(i, I_i) \end{aligned}$$

$$\Rightarrow u_i = A_i^{-1} \cdot V_i \quad \forall i \tag{2.11}$$

2.3.2 Logica di Raccomandazione

La logica di raccomandazione per l'algoritmo ALS-LMF è identica alla variante SVD dell'algoritmo.

Per tanto, il rating associato alla *generica* coppia $< Utente : i\,,\, Prodotto : j >$ è modellato dal seguente prodotto scalare:

$$r_{ij} = u_i \cdot m_j \tag{2.12}$$

Capitolo 3

Dataset

Il dataset impiegato per gli esperimenti è un dataset di Film collezionato dal gruppo di ricerca *GroupLens*, quest'ultimo asserisce all'Università del Minnesota.

Il suddetto dataset viene chiamato *MovieLens-1M* [7] e contiene le valutazioni date dagli Utenti dell'omonimo servizio su di una serie di film a catalogo.

Entrando nel dettaglio, il dataset contiene le valutazioni di 6.040 Utenti su 3.900 Film, per un totale di 1.000.209 di rating.

Inoltre, ogni rating del dataset è un numero intero compreso fra 1 e 5.

Dal punto di vista strutturale, il dataset è archiviato in un file csv avente il seguente schema:

ID Utente	ID Film	Rating	Timestamp
...	...	...	...
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
...	...	...	...

Tabella 3.1: Schema MovieLens-1M Dataset

Inoltre, è da sottolineare che ogni Utente del dataset ha valutato almeno 20 film; tutto ciò evitata di avere record di Utenti pressoché nulli.

Oltre a quanto detto, il dataset contiene due *ulteriori* file in formato `csv`:

- `users.csv`
- `movies.csv`

Il file `users.csv` include una serie di informazioni demografiche sugli Utenti del servizio.

Tale file, esibisce il seguente schema:

ID Utente	Sesso	Età	Occupazione	ZIP-Code
...	...	...	...	...
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
...	...	...	...	...

Tabella 3.2: Schema Informazioni Demografiche Utenti del MovieLens-1M Dataset

Invece, il file `movies.csv` include una serie di informazioni sui Film valutati dagli Utenti.

Nello specifico, il file adotta il seguente schema:

ID Film	Titolo	Genere
...	...	...
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
...	...	...

Tabella 3.3: Schema Informazioni Film presenti nel MovieLens-1M Dataset

Capitolo 4

Esperimenti Quantitativi

4.1 Setup Sperimentale

L'ambiente di sviluppo per tutti gli esperimenti del Progetto è stata una macchina *Linux* equipaggiata con la libreria *Apache Spark* [5].

La scelta di quest'ultima è riconducibile alla sua enorme versatilità ed efficienza rispetto ad altre soluzioni, come ad esempio *Apache Hadoop*.

Dal punto di vista del linguaggio di programmazione, è stato scelto *Scala* per via della sua *ottima compatibilità* con Spark; del resto Spark stesso è scritto in Scala.

Dal punto di vista della *Persistenza*, è stato scelto un file system distribuito.

Più precisamente, la scelta è ricaduta su *Apache HDFS* [6].

CPU	Memoria	Storage
4 Core/8 Thread	16 GB	SSD

Tabella 4.1: Specifiche Macchina di Sviluppo Impiegata

4.2 Metodologie Sperimentali

La misura di valutazione impiegata per gli esperimenti del progetto è il *Root Mean Square Error (RMSE)*, quest'ultima è la radice quadrata degli errori elevati al quadrato commessi dall'algoritmo.

Dal punto di vista matematico, l'RMSE ha la seguente formulazione:

$$\begin{aligned} y_i &= \text{ground-truth associato all'i-esimo record del dataset} \\ \hat{y}_i &= \text{inferenza di un generico algoritmo sull'i-esimo record del dataset} \\ RMSE &= \sqrt{\sum_{i=1}^{m} (y_i - \hat{y}_i)^2} \end{aligned} \tag{4.1}$$

Entrando nel merito dei dati, possiamo dire che il dataset ha subito le seguenti suddivisioni per tutti gli esperimenti:

Tipo di Set	Percentuale sull'Originale	N Record	Tipo Split
Training-Set	80 %	800.763	Random
Test-Set	20 %	199.446	Random

Tabella 4.2: Schema Splitting Dataset

Inoltre, gli algoritmi *ALS-LFM* e *SVD-LFM* hanno beneficiato anche di un validation-set, tutto ciò si riflette nella seguente *nuova* suddivisione:

Tipo di Set	Percentuale sull'Originale	N Record	Tipo Split
Training-Set	80 % sul Dataset	640.610	Random
Validation-Set	20 % sul Training-Set	160.152	Random
Test-Set	20 % sul Dataset	199.446	Random

Tabella 4.3: Schema Splitting Dataset con Validation-Set

Dal punto di vista della *taratura degli algoritmi*, quest'ultimi presentano ,rispettivamente, i seguenti parametri:

Algoritmo	Parametro	Intervallo
ALS-LFM	Numero Iterazioni	$[1, \infty]$
ALS-LFM	Regolarizzazione	$[0, 1]$
ALS-LFM	Fattori Latenti	$[1, rank(M)]$
SVD-LFM	Fattori Latenti	$[1, rank(M)]$
LSH-Reccomendation	Lunghezza Bucket	$[1.0, \infty]$
LSH-Reccomendation	Numero Tavole Hash	$[1.0, \infty]$
LSH-Reccomendation	Soglia Distanza	$[1, \infty]$

Tabella 4.4: Parametri degli Algoritmi Proposti

Inoltre, la strategia di *tuning* utilizzata per *tarare* i suddetti parametri è la *GridSearch*.

Quest'ultima, consiste nelle seguenti tre fasi:

1. Quantizzazione dello Spazio di Ricerca dei Parametri.
2. Valutazione delle Prestazioni di *tutte le combinazioni di parametri* presenti nello Spazio Quantizzato.
3. Alla fine dei Test, il set di parametri *ottimale* sarà quello che che avrà ottenuto uno *score superiore* rispetto a tutti gli altri set presi in considerazione.

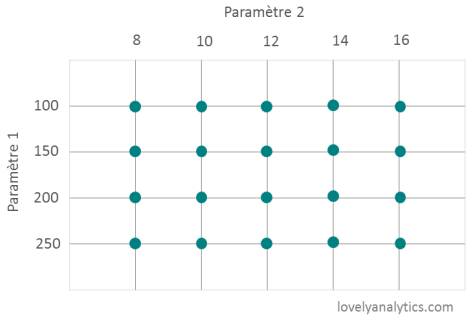

Figura 4.1: Esempio di GridSearch con Spazio di Ricerca Bidimensionale

Osservazione 1 (Dimensionalità della GridSearch degli Esperimenti). Per ragioni computazionali, le GridSearch degli esperimenti condotti saranno limitate ad una dimensionalità pari a 3.

L'unica eccezione, è rappresentata dal parametro *Numero Fattori Latenti* negli algoritmi *ALS-LFM e SVD-LFM*; del resto quest'ultimo avrà una dimensionalità pari a 5.

4.3 Esperimenti con SVD-LFM

Gli esperimenti condotti sull'algoritmo SVD-LFM implicano la taratura del *Numero di Fattori Latenti* della Decomposizione.

Nello specifico, sono stati effettuati una serie di test con *rispettivamente* 10, 50, 100, 150 e 200 Fattori Latenti.

I risultati dei suddetti test sono evidenziati nel seguente diagramma a barre:

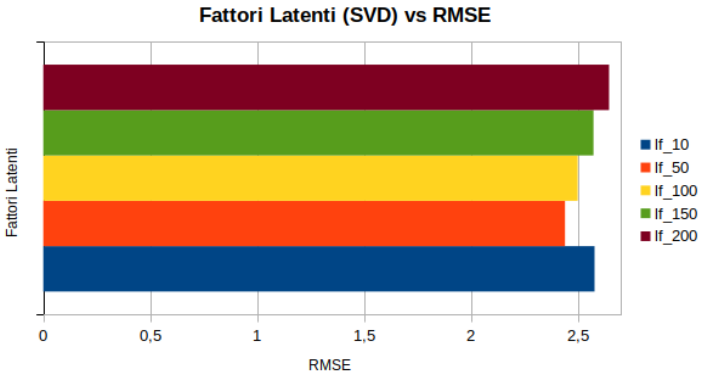

Figura 4.2: Diagramma a Barre *Fattori Latenti vs RMSE* sul Validation-Set

Come si può evincere dalla figura, la scelta *migliore* è stata quella con 50 Fattori Latenti.

Inoltre, il grafico evidenzia ,rispetto al miglior risultato, un peggioramento costante delle performance (RMSE) all'aumentare del numero di fattori latenti.

Tale fatto, è attribuibile alla *sparsità* dell'Utility Matrix come sottolineato in [2].

Infine, l'algoritmo restituisce le seguenti performance sul test-set:

Fattori Latenti	RMSE
50	2,49

Tabella 4.5: Performance Miglior Modello SVD-LFM sul Test-Set

4.4 Esperimenti con LSH-Reccomendation

Gli esperimenti condotti con l'algoritmo *LSH-Reccomendation* prevedono la taratura dei seguenti parametri:

- Dimensione Bucket
- Numero Tavole Hash
- Soglia Distanza

Per tale scopo, è stata adottata la seguente strategia di tipo *GridSearch*:

Parametro	Sample da Esaminare
Dimensione Bucket	[1 , 10 , 50]
Numero Tavole Hash	[10 , 20 , 30]
Soglia Distanza	[10 , 40 , 100]

Tabella 4.6: GridSearch per l'Algoritmo LSH-Reccomendation

Di seguito, vengono mostrati una serie di grafici inerentemente gli esperimenti condotti:

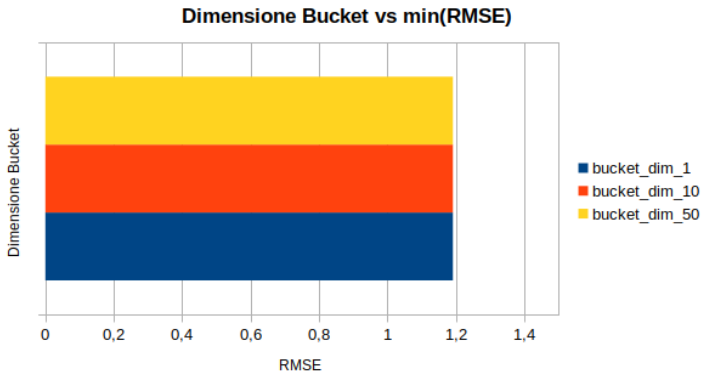

Figura 4.3: Diagramma a Barre *Dimensione Bucket vs min (RMSE)* sul Validation-Set

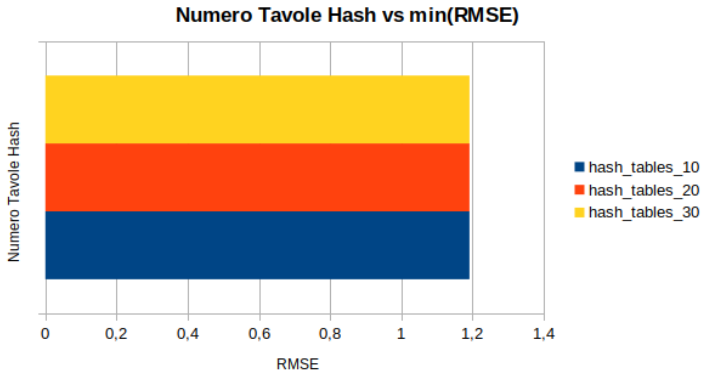

Figura 4.4: Diagramma a Barre *Numero Tavole Hash vs min (RMSE)* sul Validation-Set

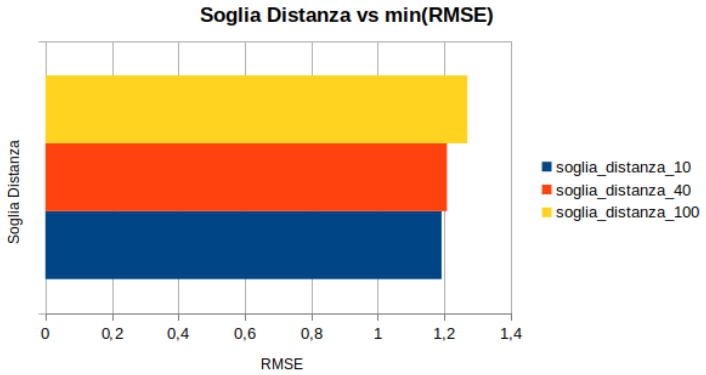

Figura 4.5: Diagramma a Barre *Soglia Distanza vs min (RMSE)*

Come si può osservare dai diagrammi, l'unico parametro che influenza *realmente* le performance è la *Soglia Distanza* mentre i due restanti sono *del tutto ininfluenti*.

Quest'ultimo fatto ,verosimilmente, è da attribuire alla sparsità dei vettori che formano l'Utility Matrix.

A conclusione di questa sezione, possiamo dire che l'algoritmo nel *suo insieme* restituisce un RMSE di 1.19 sul Test-Set.

4.5 Esperimenti con ALS-LFM

Gli esperimenti condotti sull'algoritmo ALS-LFM prevedono il tuning dei seguenti tre parametri: *Numero Fattori Latenti, Iterazioni e Regolarizzazione*.

Per tale scopo, è stata impiegata la seguente strategia di GridSearch:

Parametro	Sample da Esaminare
Fattori Latenti	[10 , 20 , 40]
Regolarizzazione	[0.9 , 0.8 , 0.7]
Numero Fattori Latenti	[10 , 50 , 100 , 150 , 200]

Tabella 4.7: GridSearch per l'Algoritmo ALS-LFM

I risultati della taratura dei suddetti parametri sono evidenziati dai seguenti grafici:

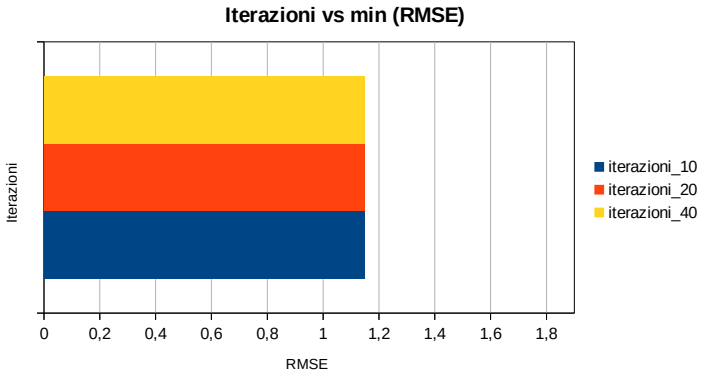

Figura 4.6: Diagramma a Barre *Iterazioni vs min (RMSE)* sul Validation-Set

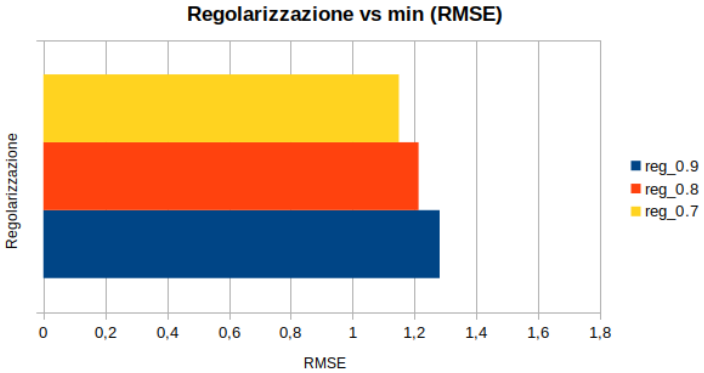

Figura 4.7: Diagramma a Barre *Regolarizzazione vs min (RMSE)* sul Validation-Set

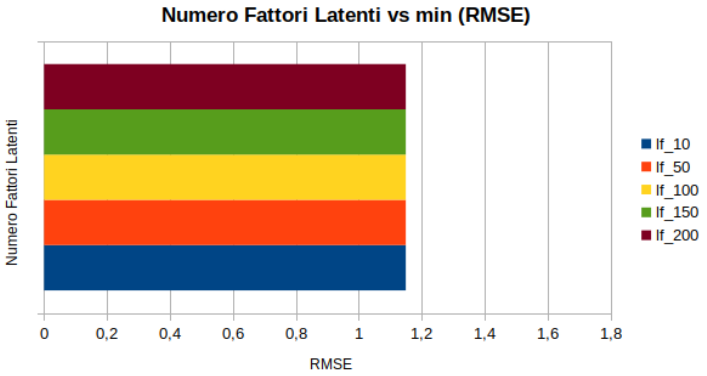

Figura 4.8: Diagramma a Barre *Fattori Latenti vs min (RMSE)* sul Validation-Set

Come si può apprezzare dalle immagini, le variazioni dei parametri *Iterazioni e Fattori Latenti* non hanno fornito alcun miglioramento alle performance del Modello.

Invece, il parametro *Regolarizzazione* influisce *realmente* sulle prestazioni di quest'ultimo.

Infatti, abbassando *sensibilmente* tale parametro si ottiene un *buon* decremento dell'RMSE e quindi un incremento di prestazione.

Dal punto di vista delle performance complessive, l'algoritmo esibisce il seguente RMSE sul Test-Set:

Fattori Latenti	**Regolarizzazione**	**Iterazioni**	**RMSE**
10	0,7	40	1,15

Tabella 4.8: Performance Miglior Modello ALS-LFM sul Test-Set

Conclusione

L'esperienza maturata in questo progetto ha messo alla luce tanti aspetti *significativi* nell'ambito della disciplina del Data Mining (su *grandi* quantità di dati).

Dal punto di vista algoritmico, la soluzione più performante è stata l'*Alternative Least-Square*.

Ciò nonostante, l'algoritmo basato su *Localitive-Sensitive Hashing* riesce a garantire performance vicine a *ALS-LFM* senza l'impiego di un processo di learning.

Tale prerogativa ,in determinate situazioni, potrebbe essere preferibile.

Un esempio, potrebbe essere la *spiegabilità* delle scelte dell'algoritmo.

Del resto, l'algoritmo *ALS-LFM* ,come tutti gli algoritmi di learning, implica l'ottimizzazione di una funzione costo.

Tale fatto, *inibisce* una reale comprensione del *perché l'algoritmo fa determinate scelte*.

Un altro aspetto *significativo* emerso da questo progetto, è l'*efficienza* delle implementazioni degli algoritmi.

In particolar modo, il Sottoscritto ha potuto apprezza *realmente* gli effetti negativi di un'implementazione inefficiente.

Entrando nei dettagli, la prima implementazione dell'algoritmo LSH-Rec (corrispondente allo pseudo-codice in 1) implicava un tempo di calcolo di $44\,s$ a record.

Considerando che il dataset originale aveva 800.763 record, il tempo di calcolo complessivo sarebbe stato di *circa* 1 anno.

Per ovviare a ciò, il Sottoscritto ha dovuto *riscrivere da Zero* l'algoritmo in modo tale che venissero sfruttati *al massimo* i core presenti sulla macchina.

Tale fatto, ha portato ad un incremento poderoso delle prestazioni di *runtime*, del resto siamo passati dal processare *1 record ogni 44 secondi* a processarne *120.000 c.a. ogni 44 secondi*.

In virtù della suddetta esperienza, posso dire che l'ottimizzazione degli algoritmi in ambito Big Data è *fondamentale* se si vuole ottenere un *certo* livello di performance.

Bibliografia

[1] Bell et. al. - Matrix Factorization Techniques for Recommender Systems

[2] Bell et. al. - Scalable Collaborative Filtering with Jointly Derived Neighborhood. Interpolation Weights.

[3] Zhou et. al. - Large-scale Parallel Collaborative Filtering for the Netflix Prize

[4] Wikipedia - LSH

[5] Zaharia - Spark: Cluster Computing with Working Sets

[6] Shvachko et. al. - The Hadoop Distributed File System

[7] GroupLens - MovieLens Datasets

[8] Leskovec et. al. - Mining of Massive Datasets

[9] Nocedal, Wright - Numerical Optimization