Relazione di mini-progetto gruppo 02

Bazzani Mattia[1], Bernini Carlo Guido[2], and Cugnigni Daniele[3]

[1] Corso di laurea magistrale in Scienze Statistiche, matricola 2062088
mattia.bazzani@studenti.unipd.it
[2] Corso di laurea magistrale in Scienze Statistiche, matricola 2040623
carloguido.bernini@studenti.unipd.it
[3] Corso di laurea magistrale in Scienze Statistiche, matricola 2054519
daniele.cugnigni@studenti.unipd.it

Sommario Il seguente progetto tratta un problema di recommendation systems. Sono state approfondite le tecniche della decomposizione UV e del factorization machine che rientrano in un approccio collaborative-filtering. L'obiettivo è sottolineare le differenze tra i due metodi dal punto di vista previsivo, di costo computazionale e di memoria. Il dataset utilizzato per il confronto fa riferimento a recensioni di utenti relative a birre. Dall'applicazione di tali metodologie si è potuto notare che la decomposizione UV risulta essere più efficiente sia dal punto di vista del costo computazionale che per l'utilizzo di memoria. Per quanto riguarda l'efficacia la tecnica migliore dipende dalla misura (percentuale di osservazioni correttamente classificate o RMSE) considerata per il confronto.

Keywords: recommender systems · decomposizione UV · factorization machines

1 Introduzione

Nel contesto dei sistemi di raccomandazione esistono due tipologie di entità: utenti (user) ed item. A partire dal fatto che gli utenti esprimono giudizi, secondo una certa scala di valutazione, per alcuni item, può essere costruita la matrice di utilità R, detta anche dei rating, di dimensione $n \times m$. Le dimensioni fanno riferimento rispettivamente al numero di utenti ed item distinti, mentre la singola cella rappresenta la valutazione data dall'i-esimo utente al j-esimo item. Tale matrice è sparsa, ovvero caratterizzata da un elevato numero di elementi mancanti, dato che difficilmente un utente recensisce tutti gli item presenti. Lo scopo generico dei sistemi di raccomandazione è quello di riuscire a prevedere i giudizi di un utente relativamente ad un item.
Tale obiettivo può essere raggiunto tramite due approcci: content based e collaborative filtering. In questa applicazione di data science viene trattato quest'ultimo approccio che sfrutta le relazioni insite tra utenti ed item; in particolare vengono approfondite le tecniche della decomposizione UV e del factorization machine.
Il problema risulta essere significativo dal momento che fare una stima attendibile sulla valutazione di un utente per un item potrebbe portare vantaggi

considerevoli. Ad esempio per un'azienda consigliare un prodotto con caratteristiche gradite ad un cliente potrebbe far aumentare le probabilità di vendita di quest'ultimo. Con l'applicazione di queste due metodologie si vuole vedere se il problema di base sopra descritto possa essere risolto in maniera efficace, andando a valutare quanto attendibili risultano essere le previsioni per i rating con le due tecniche considerate, e quali siano le differenze in termini di costo computazionale e di memoria tra le due procedure.

2 Base di partenza

Il primo approccio preso in considerazione per affrontare il problema descritto in precedenza è la decomposizione UV [1]. Come è possibile vedere dalla Figura 1, l'idea sottostante tale metodo è che, data una matrice di utilità $R_{[n\times m]}$, ci siano k dimensioni (o fattori latenti) che consentono di caratterizzare sia gli user che gli item. Più specificatamente, la decomposizione UV ha lo scopo di determinare

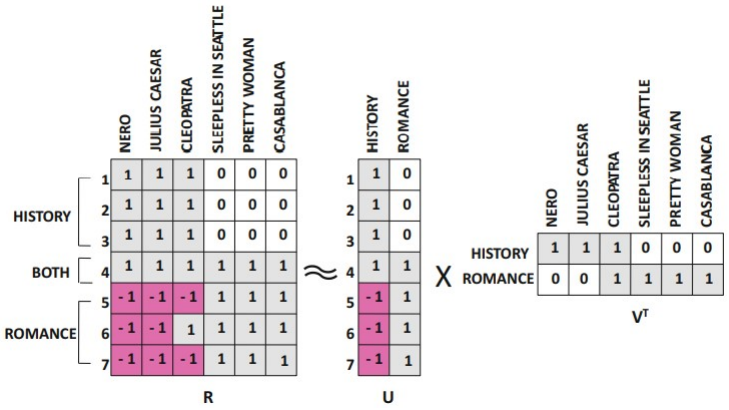

Figura 1: Esempio decomposizione UV (fonte Aggarwal, C.C. (2016). Recommender Systems The Textbook. Springer International Publishing, 1st Edition.)

senza porre alcun vincolo e basandosi sulla funzione di perdita:

$$J = \frac{1}{2} \sum_{(i,j)\in S} e_{ij}^2 = \frac{1}{2} \sum_{(i,j)\in S} (r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js})^2 \tag{1}$$

una matrice $U_{[n\times k]}$ e una matrice $V_{[m\times k]}$ tali per cui $E = R - UV^T \approx 0$, con e_{ij}, $i = 1, ..., n$, $j = 1, ..., m$, generico elemento della matrice E.
Ci sono molteplici algoritmi che consentono di ricavare le matrici U e V, tra i più conosciuti vi è lo Stochastic Gradient Descent (SGD). Il vantaggio di questo

algoritmo è che permette di arrivare in tempi relativamente brevi alla stima delle matrici U e V, anche se generalmente si ottengono delle previsioni meno accurate rispetto ad altri algoritmi computazionalmente più onerosi. Il funzionamento dello SGD è illustrato nello Pseudo-codice 1. In una singola iterazione: viene perturbato casualmente l'ordine dei rating osservati, sulla base del nuovo ordine si visita un rating alla volta, per ognuno si calcola l'errore corrispondente, si aggiornano le corrispondenti righe di U e di V e, dopo aver visitato tutti i rating osservati, si verifica la condizione di convergenza. É da notare come nell'algoritmo entrino in gioco solamente i rating osservati e i corrispondenti errori calcolati.

Algoritmo SGD (Matrice di utilità: R, tasso di apprendimento: α)
begin
  Inizializza casualmente le matrici U e V;
  S = $\{(i,j) : r_{ij}$ è osservato$\}$
  while not (convergenza) **do**
  begin
    Mescola casualmente le entrate osservate in S;
    for ogni $(i,j) \in S$ nell'ordine dopo il mescolamento **do**
    begin
      $e_{ij} = r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}$;
      for ogni $q \in 1, ..., k$ **do** $u_{iq}^{+} = u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq}$;
      for ogni $q \in 1, ..., k$ **do** $v_{jq}^{+} = v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq}$;
      for ogni $q \in 1, ..., k$ **do** $u_{iq} = u_{iq}^{+}$ e $v_{jq} = v_{jq}^{+}$;
    end;
    Controlla la condizione di convergenza;
  end
end

Pseudo-codice 1: Stochastic Gradient Descent decomposizione UV

Uno dei problemi più comuni in cui si incorre con la decomposizione UV, quando si ha l'obiettivo di approssimare una matrice di partenza sparsa, è l'overfitting, ossia l'eccessivo adattamento della matrice UV^T alla matrice di partenza e, di conseguenza, uno scarso risultato in termini previsivi. Una delle soluzioni più comunemente utilizzate per ovviare a tale problema è la regolarizzazione, ovvero l'inserimento di un parametro λ all'interno della funzione obiettivo (1) che permette di ottenere un trade-off tra adattamento ai dati osservati ed efficacia in termini di previsioni. In questo caso, la funzione obiettivo da minimizzare diventa:

$$\begin{aligned} J = \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2 = \\ \frac{1}{2} \sum_{(i,j) \in S} (r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js})^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2 \end{aligned} \tag{2}$$

Il secondo approccio trattato è il factorization machine. I factorization machine sono una classe di modelli introdotta nel 2010 da Steffen Rendle [2].
Nell'ambito dei sistemi di raccomandazione è una tecnica di collaborative-filtering che sfrutta informazioni di contorno per ottenere le previsioni dei rating. Le informazioni solitamente sono di tipo context-aware, come l'orario o il luogo dove si trovava lo user nel momento in cui è stata lasciata la recensione, oppure di tipo content-based, ossia sulle caratteristiche degli item.
Differentemente dalla maggior parte delle tecniche di collaborative-filtering, i dati di input dell'algoritmo non sono nella struttura della matrice di utilità, ma hanno una forma tabellare. Nella tabella la singola colonna rappresenta una feature del dataset. Le colonne, tranne l'ultima, sono divise in gruppi: il primo gruppo di colonne sono solitamente gli user, il secondo gruppo di colonne gli item, i successivi gruppi corrispondono a feature aggiuntive, mentre l'ultima colonna contiene i rating (target). Per rappresentare user e item viene usato lo 'one hot encoding': per ogni riga si pone un valore pari a uno in corrispondenza delle colonne relative allo user e item che si stanno considerando, mentre pari a zero per i restanti user e item. Ogni riga rappresenta l'interazione tra uno specifico user ed uno specifico item. In generale si può quindi dire che questa tabella rappresenta la stessa informazione della matrice di utilità, includendo inoltre anche altre feature che possono essere utili all'analisi. Continuando il paragone, la tabella presenta più zeri della matrice di utilità, rendendola quindi ancora più sparsa. Per un esempio grafico si veda la Figura 2.
Una volta definita la struttura di dati del factorization machine possiamo analizzare il modello utilizzato per prevedere rating sconosciuti:

$$\tilde{r}^{(l)} = \omega_0 + \sum_{i=1}^{p} \omega_i x_i^{(l)} + \sum_{i=1}^{p} \sum_{j=i+1}^{p} W_{i,j} x_i^{(l)} x_j^{(l)} \tag{3}$$

l'indice l indica la riga della tabella e $\tilde{r}^{(l)}$ è il rating che si vuole stimare per la riga l. Gli indici i e j rappresentano colonne della tabella, $x_i^{(l)}$ è l'elemento della tabella corrispondente alla riga k e alla colonna i e può essere 0 o 1, mentre p è il numero totale di feature nel dataset. I parametri che si vogliono "apprendere" per prevedere il rating sono ω_0 , gli ω_i e i $W_{i,j}$. Essi non dipendono dalla riga l, perciò sono gli stessi per ogni interazione user/item/side info.
Il modello sopra descritto è composto da tre componenti, una costante, una vettoriale ed una matriciale:

- la prima componente è ω_0, che rappresenta un effetto globale interpretabile come il rating medio su tutti gli item da tutti gli user;
- la seconda componente è ω, un vettore di parametri di dimensione p. I singoli ω_i rappresentano l'effetto di ciascuna feature. Per ogni interazione user/item/side info, moltiplicando gli ω_i per x_i fondamentalmente si tengono nel modello solo gli effetti delle feature presenti nell'interazione perchè tutti gli altri ω_i saranno moltiplicati per 0;

– la terza componente permette di considerare nel modello anche le relazioni tra gruppi di feature, ossia quanto il rating vari quando due feature compaiono assieme in una interazione. Questo viene rappresentato utilizzando una matrice di parametri W. Solo quando sia $x_i^{(l)}$ che $x_j^{(l)}$ sono pari a 1 il corrispondente $W_{i,j}$ influenzerà la previsione del rating. La matrice W contenente i $W_{i,j}$ ha dimensione $p \times p$.

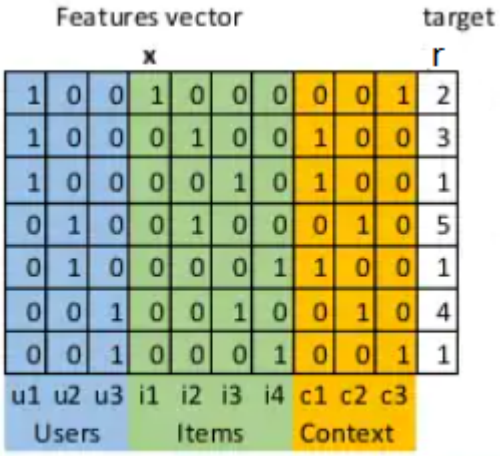

Figura 2: Esempio tabella Factorization Machine (fonte https://www.slideshare.net/Evention/the-factorization-machines-algorithm-for-building-recommendation-system-pawe-agodziski-sas-institute)

Il numero totale di parametri nel modello è quindi $1 + p + p^2$.
La matrice W rappresenta un problema, poiché è molto grande e sparsa e spesso, con dati reali, porta il numero totale dei parametri del modello ad essere maggiore del numero di recensioni disponibili per allenarlo, causando overfitting.
La soluzione è ridurre la dimensione di W attraverso una fattorizzazione. Si utilizza una matrice $V_{[p \times k]}$ con $k << p$ tale che $W \approx VV^T$ cosicché i parametri da stimare passino da $1 + p + p \times p$ a $1 + p + p \times k$, dove k è il numero di fattori latenti. Questi ultimi, differentemente da quelli considerati nella decomposizione UV, non sono soggetti ad interpretazione.
Ci sono diversi metodi per apprendere i parametri del factorization machine, il metodo utilizzato in questo progetto è lo Stochastic Gradient Descent (SGD) con regolarizzazione, algoritmo proposto da Rendle nel suo lavoro del 2012 [3].
Il funzionamento dello SGD è illustrato nello Pseudo-codice 2. In una singola iterazione: l'ordine dei rating del dataset viene perturbato casualmente, sulla base del nuovo ordine si visita un rating osservato alla volta, per ognuno si calcola

la funzione di perdita (scarto quadratico), si aggiornano i parametri del modello con in input le feature della recensione estratta e, dopo aver visitato tutti i rating del dataset, si verifica la condizione di convergenza.

Algoritmo SGD Factorization Machine (Tabella dei dati: S, tasso di apprendimento: α, parametro di regolarizzazione λ, σ inizializzati)
$\omega_0 \leftarrow 0; \omega \leftarrow (0, \ldots, 0); \mathbf{V} \sim \mathcal{N}(0, \sigma)$
repeat
  for (**x**,r) $\in S$ **do** ;
    $\omega_0 \leftarrow \omega_0 - \alpha \left(\frac{\partial}{\partial \omega_0} l(\tilde{r}(\mathbf{x} \mid \Theta), r) + 2\lambda^0 \omega_0 \right)$
    for $i \in 1, ..., p \wedge x_i \neq 0$ **do**
      $\omega_i \leftarrow \omega_i - \alpha \left(\frac{\partial}{\partial \omega_i} l(\tilde{r}(\mathbf{x} \mid \Theta), r) + 2\lambda_\omega \omega_i \right)$;
    for ogni $f \in 1, ..., k$ **do**
      $v_{i,f} \leftarrow v_{i,f} - \alpha \left(\frac{\partial}{\partial v_{i,f}} l(\tilde{r}(\mathbf{x} \mid \Theta), r) + 2\lambda_v v_{i,f} \right)$;
    end
  end
end
until criterio di convergenza è soddisfatto;

Pseudo-codice 2: Stochastic Gradient Descent Factorization Machine

I gradienti della funzione di perdita presenti nello Pseudo-codice 2 sono pari a:

$$\frac{\partial}{\partial \theta} l(\tilde{r}(\mathbf{x} \mid \Theta), r) = \begin{cases} 2(\tilde{r} - r), & \text{if } \theta \text{ is } w_0 \\ 2(\tilde{r} - r)x_i, & \text{if } \theta \text{ is } w_i \\ 2(\tilde{r} - r)x_i \sum_{j \neq i} v_{j,f} x_j, & \text{if } \theta \text{ is } v_{i,f} \end{cases} \tag{4}$$

3 Problema affrontato

In questa relazione, le tecniche descritte sono applicate ad un dataset che fa riferimento a recensioni di birre lasciate nel sito "Ratebeer". I dati sono stati richiesti a Joe Tucker, referente del sito che ne ha concesso l'utilizzo per fini accademici, e coprono un periodo temporale di oltre dieci anni, a partire da Aprile 2000 fino a Novembre 2011. Il dataset contiene quasi tre milioni di recensioni, con ognuna che include valutazioni riguardo: aspetto, aroma, palato, gusto e valutazione complessiva della birra. Oltre a queste variabili vengono riportate informazioni sul prodotto, sull'utente, sulla gradazione alcolica ed infine una recensione scritta. Il numero totale di utenti è di 29265, mentre le diverse tipologie di birra presenti sono 110369.
Il dataset di partenza ha una dimensione pari a $1,61$ GB e si presenta in formato di testo (.txt). Ogni recensione è composta da 13 variabili e viene separata dalla successiva con una riga vuota.
Si prosegue ora con la descrizione dei passaggi necessari che hanno permesso di poter lavorare con i dati a disposizione. Nella fase di pre-processing la prima

operazione effettuata è stata selezionare le variabili utili all'analisi ed importare i valori osservati di ognuna in un array distinto. Per fare ciò è stata utilizzata una regular expression tale da andare a selezionare la parte di testo presente dopo ogni intestazione della variabile considerata. Questa operazione è stata effettuata per quattro delle tredici variabili di partenza, in particolare per ID della birra, gradazione alcolica, valutazione complessiva in scala da 1 a 20 data dall'utente e nominativo di quest'ultimo. Le altre variabili, in quanto non di diretto interesse, non sono state importate per ottimizzare l'allocazione di spazio in memoria. Nel corso del progetto, quando si è trattato di andare a costruire la matrice di utilità, è stato riscontrato il problema che alcuni rating assumevano valori maggiori di 20. A seguito di alcune verifiche è stato notato che effettivamente il dataset conteneva coppie (user, item) che si ripetevano. Per ovviare il problema, è stata selezionata solamente la prima coppia (user,item) in ordine di apparizione, considerando il fatto che le recensioni non compaiono nel dataset secondo qualche criterio prestabilito. A questo punto, un'ulteriore selezione è stata fatta eliminando quegli utenti con un numero di birre recensite inferiore o uguale a 20. La soglia è stata scelta in modo arbitrario, ragionando sul fatto che per ottenere delle previsioni affidabili per ciascun utente ci dovesse essere un numero minimo di recensioni effettuate da ognuno di essi.

4 Esperimenti

A partire dai quattro array contenenti i valori delle variabili oggetto di interesse descritti nella sezione 3, è stato possibile costruire una lista di liste, dove al generico elemento della lista principale è associata la lista di quattro elementi caratterizzanti l'i-esima recensione. Dalla lista di liste è stato possibile effettuare la ripartizione del dataset in validation, training e test set, rispettivamente nelle seguenti proporzioni: 5%, 70% e 25%. Nel validation set sono state testate varie combinazioni di iper-parametri, da cui sono state selezionate le migliori per allenare i modelli con i dati del training set. Il passo successivo è stato quello di costruire due dizionari mappando il nome degli utenti e l'identificativo della birra con un numero univoco e progressivo. Gli array, la lista di liste e i due dizionari sono le strutture dati che costituiscono una base di partenza comune per i due approcci, ma da questo punto in poi sarà possibile osservare alcune differenze.
Focalizzando in un primo momento l'attenzione sulla decomposizione UV, grazie alla mappatura si è potuto considerare la tripla (item, user, rating) necessaria per andare ad ottenere la matrice di utilità. Come nella maggior parte dei casi, queste matrici sono sparse. Per ovviare problemi di spazio di archiviazione e di costo computazionale sono state utilizzate rappresentazioni matriciali alternative rispetto a quella classica. É stato fatto riferimento alla rappresentazione Compressed Sparse Row (CSR), Compressed Sparse Column (CSC) e Coordinate (COO) del modulo Scipy di Python [4]. Secondo quest'ultimo tipo di rappresentazione, la matrice di utilità viene descritta da tre vettori contenenti:

- i valori dei rating osservati;

– gli indici di riga corrispondenti ai rating osservati;
– gli indici di colonna corrispondenti ai rating osservati.

Prima dell'implementazione dello Stochastic Gradient Descent è stato necessario centrare la matrice di utilità andando a sottrarre ad ogni cella la corrispondente media di riga e di colonna. Per questa procedura è stata considerata la matrice in formato CSR per il calcolo della media degli utenti e il formato CSC per la media degli item. In seguito, è stata eseguita la conversione nel formato COO e da questa si è passati all'implementazione dell'algoritmo.
Facendo riferimento allo pseudo-codice 1, sono stati inseriti come parametri, oltre alla matrice di utilità centrata R* e al tasso di apprendimento α, il parametro di regolarizzazione λ, il numero di dimensioni latenti d, il numero massimo di iterazioni ammesso, la soglia ε che stabilisce se sia stata raggiunta o meno la convergenza e due seed, necessari per rendere replicabile l'esperimento. In particolare, in seguito al tuning degli iper-parametri sul validation set e al consiglio dato dal referente di Ratebeer sulle possibili classificazioni delle birre, sono stati fissati α pari a 0.004 ed ε pari a 0.005, facendo variare λ tra 0 (ovvero assenza di regolarizzazione), 0.4 e 0.8, il numero di fattori latenti tra 15 e 17. Questi ultimi due valori indicano rispettivamente le differenti tipologie di birre per provenienza e per gusto. Un'altra decisione presa è stata inizializzare gli elementi delle matrici U e V con la radice quadrata del rapporto tra la media empirica dei rating centrati e il numero di dimensioni latenti, nel caso di media dei rating maggiore di zero, e con zero nel caso di media empirica dei rating centrati inferiore a zero. A partire da tale inizializzazione, si è andati ad aggiungere ad ogni elemento una diversa costante c generata (pseudo) casualmente da una variabile Normale con media e varianza pari a media e varianza dei rating centrati e con seed per le matrici U e V pari rispettivamente a 123 e 456. Come conseguenza dell'utilizzo di una funzione di perdita quadratica, si è deciso di prendere il Mean Squared Error (MSE) come valutazione dell'approssimazione alla fine di ogni singola iterazione. Si è confrontata quindi la differenza relativa in valore assoluto tra l'MSE al passo corrente ed al passo precedente con ε: se questa era minore di ε si è giunti a convergenza.
Passando ora al factorization machine, come strutture dati di partenza sono state utilizzate la lista di liste e i due dizionari per item e user precedentemente citati, grazie alle quali è stata ricostruita la tabella dei dati necessaria come input per il factorization machine. La tabella è stata rappresentata con un formato simile al CSR, con tre "vettori" rappresentati da numpy.ndarray (struttura dati del pacchetto numpy [5]) che contengono rispettivamente:

– il numero di valori, per riga, diversi da zero cumulati;
– indici di colonna dei valori non nulli;
– i rating non nulli.

È stato poi creato un dizionario contenente i tre vettori, che viene utilizzato come input per lo SGD (pseudo-codice 2).
Più nello specifico, nella tabella sono stati considerati quattro gruppi di colonne relativi rispettivamente a: user, item, informazione di contorno content-based

sulla birra riguardante la gradazione alcolica e rating osservati. Con riferimento alla gradazione della birra, è stato considerato un gruppo di tre colonne:

- nella prima colonna un 1 corrisponde ad una birra con gradazione alcolica minore di 5, 0 altrimenti.
- nella seconda colonna un 1 corrisponde ad una birra con gradazione alcolica compresa tra 5 (incluso) e 6 (escluso), 0 altrimenti.
- nella terza colonna un 1 corrisponde ad una birra con gradazione alcolica compresa tra 6 (incluso) e 8 (escluso), 0 altrimenti.
- se in una riga tutte e tre le colonne sono pari a 0 allora la birra ha una gradazione alcolica maggiore o uguale a 8.

Gli intervalli sono stati scelti guardando i quantili (0.25, 0.5 e 0.75) della distribuzione della gradazione alcolica della birra nel dataset.
Facendo riferimento allo Pseudo-codice 2, sono stati inseriti come parametri, oltre alla tabella dei dati S e al tasso di apprendimento α, i parametri di regolarizzazione λ (uno per l'aggiornamento di ω_0, uno per l'aggiornamento degli ω_i, uno per l'aggiornamento dei $v_{i,f}$), il numero di fattori latenti k, il numero massimo di iterazioni ammesso e la soglia ε che stabilisce se sia stata raggiunta o meno la convergenza. In particolare, in seguito al tuning degli iper-parametri sul validation set, sono stati scelti due set:

- α pari a 0.001, λ pari a $[0.1, 0.4, 0.7]$ ed ε pari a 0.05
- α pari a 0.001, λ pari a $[0.01, 0.1, 0.4]$ ed ε pari a 0.05

In entrambi i casi il numero dei fattori latenti è stato fissato a 1 per rendere il modello il meno complesso possibile. Anche in questo caso, come conseguenza dell'utilizzo di una funzione di perdita quadratica, si è deciso di prendere il Mean Squared Error (MSE) come valutazione dell'approssimazione alla fine di ogni singola iterazione. In particolare si è confrontata la differenza relativa in valore assoluto tra l'MSE al passo corrente e al passo precedente con ε: se questa era minore di ε si è giunti a convergenza.
Un'altra decisione presa è stata inizializzare gli elementi della matrici V da una Normale con media 0 e varianza pari a 0.001. Tale inizializzazione è stata effettuata fissando un seed pari ad 1.
L'ultimo passo effettuato è stato quello di valutare efficacia ed efficienza dei due metodi trattati. Per il primo aspetto sono stati calcolati il Root Mean Squared Error (RMSE) e la percentuale di osservazioni del test set classificate correttamente, mentre per il secondo è stata prestata attenzione al costo computazionale, al numero di iterazioni per arrivare a convergenza e allo spazio di memoria allocato nel tempo.
La Tabella 1 mostra i risultati ottenuti con le diverse combinazioni degli iper-parametri selezionate in fase di validazione. Nel caso in cui si focalizzi l'attenzione sul RMSE, si può notare come la performance migliore sia stata ottenuta con il factorization machine: con α pari a 0.001, il vettore dei λ pari a [0.01, 0.1, 0.4] e il numero di fattori latenti pari a 1 il RMSE è risultato essere pari a 3.12. D'altra parte, se ci si concentra sulla percentuale di rating classificati correttamente, è

possibile evidenziare come il risultato più efficace sia stato ottenuto nella decomposizione UV: con α pari a 0.004, λ pari a 0.8 e un numero di fattori latenti pari a 15 si è ottenuta una percentuale pari a 16.85 quando si è mantenuta la scala originale (ovvero la scala da 1 a 20) e una percentuale pari a 58.02 quando si è passati ad una scala da 1 a 5. Una peculiarità è che la percentuale di osservazioni classificate correttamente con le diverse combinazioni di iper-parametri scelte per la decomposizione UV si attesta sempre tra il 15% e il 17% con riferimento alla scala [1-20] e tra il 53% e il 59% considerando la scala [1-5].
Per quanto riguarda l'efficienza, possiamo rilevare una notevole differenza di costo computazionale in termini di CPU time e di numero di iterazioni per arrivare a convergenza tra la decomposizione UV e il factorization machine: il primo metodo impiega un numero maggiore di iterazioni rispetto al secondo, ma quest'ultimo è computazionalmente più oneroso rispetto alla decomposizione UV.

	UV						**FM**	
α	0.004	0.004	0.004	0.004	0.004	0.004	0.001	0.001
λ	0	0.4	0.8	0	0.4	0.8	[0.1, 0.4, 0.7]	[0.01, 0.1, 0.4]
Fattori latenti	15	15	15	17	17	17	1	1
ε	0.005	0.005	0.005	0.005	0.005	0.005	0.05	0.05
Numero iterazioni	10	7	5	10	7	5	2	3
CPU time	383.19	271.27	192.39	381.70	272.59	194.09	1055.25	1786.33
RMSE test	4.32	3.45	3.26	4.57	3.55	3.32	3.49	**3.12**
Classificazione 1-20 (%)	15.51	16.51	**16.85**	15.28	16.35	16.71	8.94	13.08
Classificazione 1-5 (%)	54.06	57.13	**58.02**	53.44	56.74	57.82	37.45	51.02

Tabella 1: Risultati ottenuti

Riferendosi allo spazio di memoria allocato durante l'esecuzione dello SGD per la decomposizione UV e per il factorization machine, il confronto è stato effettuato utilizzando il modulo memory-profiler [6] e la sua funzionalità mprof. Osservando le Figure 3 e 4, emergono due aspetti rilevanti:

- si nota dall'asse delle ascisse, contenente il tempo in secondi per l'esecuzione degli algoritmi, che il factorization machine sia molto più lento rispetto alla decomposizione UV, confermando ulteriormente la maggiore efficienza di quest'ultimo dal punto di vista del tempo di esecuzione;
- il factorization machine comporta un utilizzo di memoria maggiore rispetto alla decomposizione UV, mostrandosi quindi inefficiente anche sotto questo punto di vista.

L'implementazione è stata fatta interamente attraverso Python, in particolare tutti i passaggi precedentemente descritti nella relazione (sia riguardo la decomposizione UV sia riguardo il factorization machine) sono stati implementati in

più moduli separati. Attraverso un modulo finale (in cui sono stati importati tutti i moduli scritti precedentemente) l'applicazione dei due algoritmi al dataset (comprensiva di pre-processing, tuning degli iper-parametri, allenamento dei modelli sul training set, riscontro dei risultati dei due modelli sul test set) è stata resa interattiva e più intuitiva da utilizzare.

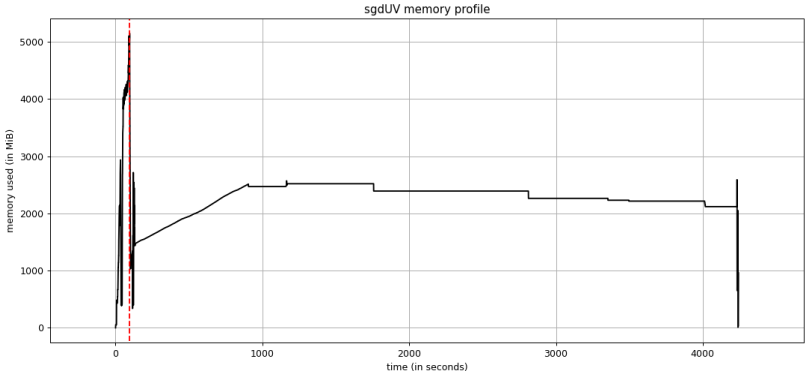

Figura 3: Memory profile SGD decomposizione UV

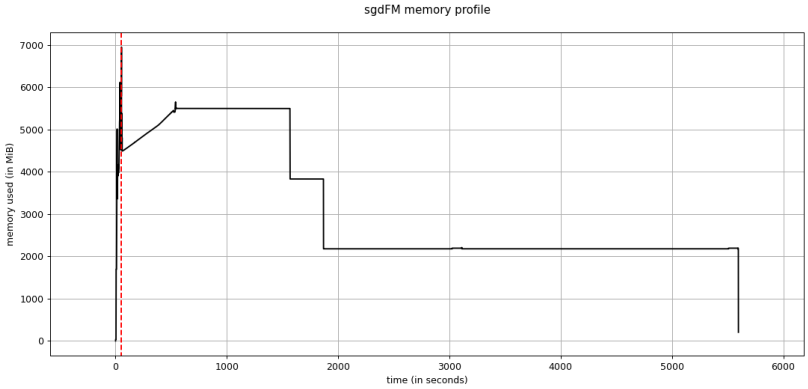

Figura 4: Memory profile SGD factorization machine

Riferimenti bibliografici

1. C. C. Aggarwal, *Recommender Systems - The Textbook*. Springer, 2016.

2. S. Rendle, "Factorization machines," in *2010 IEEE International Conference on Data Mining*, pp. 995–1000, 2010.
3. S. Rendle, "Factorization machines with libFM," *ACM Trans. Intell. Syst. Technol.*, vol. 3, pp. 57:1–57:22, May 2012.
4. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.
5. C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.
6. F. Pedregosa, "memory-profiler module." https://pypi.org/project/memory-profiler/.