
# Singular Value Decomposition for Latent Semantic Index in Recommendation Systems

Anno Accademico 2016/2017

A cura di
Gemma Martini

28 marzo 2017

# Indice

# 1 Introduzione

L'insegnamento è da secoli guidato dalla figura di un docente, che propone ad un gruppo (talvolta composto da un solo elemento) di studenti un insieme di tematiche, secondo un ordinamento che possa facilitarne l'apprendimento.

L'obiettivo di questa trattazione è di valutare l'utilità della *singular value decomposition* (da qui SVD) e del *latent semantic indexing* (in seguito LSI) applicati ai *recommendation system* per la piattaforma di allenamento per le Olimpiadi Italiane di Informatica, il *Contest Management System* (da qui CMS), con l'obiettivo di suggerire agli utenti iscritti quali argomenti (o meglio problemi) affrontare.

Si assume nel lettore la conoscenza della SVD, ma è possibile trovare, per i meno esperti, le nozioni teoriche necessarie per spiegare LSI, in appendice A.

# 2 La teoria

## 2.1 Latent Semantic Indexing

L'obiettivo teorico di questo algoritmo è la modellazione dei dati su un sistema, in modo da ottenere informazioni concise, ma espressive circa le grandezze in gioco.

Più in dettaglio, nella prima parte vengono gettate le fondamenta per introdurre lo *sketch*, che rappresenta sinteticamente le informazioni sugli utenti e sui *task*. Nella seconda parte, in base a questo *sketch*, viene definita la **somiglianza** di più utenti tra loro.

### 2.1.1 I dati

Sia $T$ un insieme di problemi di cardinalità $t$.

Sia $P$ l'insieme degli utenti che hanno accesso alla piattaforma (e quindi ai $t$ *task*), con $|P| = p$.

Sia $M \in M(p, t, \mathbb{N})$ la matrice dei punteggi totalizzati dagli utenti di $P$ sui *task* di $T$. In particolare $(m_{ij})$ è uguale al punteggio ottenuto da $p_i$ sul *task* $t_j$.

Sia $B = M^t M \in S(t, \mathbb{N})$ la matrice di elementi $(b_{ij})$, che rappresentano la somiglianza tra i due *task* $t_i$ e $t_j$.

Sia $C = MM^t \in S(p, \mathbb{N})$ la matrice di elementi $(c_{i,j})$, che rappresentano la somiglianza tra l'utente $p_i$ e l'utente $p_j$.

### 2.1.2 Il modello

Con i dati definiti come sopra è possibile decomporre $M$ mediante SVD, ottendendo tre matrici:

Matrice sinistra dei vettori singolari $S \in M(p, r, \mathbb{R})$

Matrice destra dei vettori singolari $U \in M(t, r, \mathbb{R})$

Matrice diagonale dei valori singolari $\Sigma \in D(r, \mathbb{R})$

tali che $M = S\Sigma U^t$. La matrice $\Sigma$ ha, sulla diagonale, elementi decrescenti; di conseguenza gli ultimi elementi possono essere trascurati. Si può decidere di ridurre la matrice $\Sigma$ ad una matrice in $M(k, \mathbb{R})$, ottenendo $\Sigma_k$, $S_k$ e $U_k$, per calcoli più veloci e per ridurre il rumore dovuto a dati non significativi.

$M_k = S_k\Sigma_k U_k^t$ è di dimensione $p \times t$, come $M$, e la approssima.

### 2.1.3 Informazioni nascoste

In modo un po' informale è possibile definire che cos'è un' "informazione nascosta": come la diagonalizzazione di una matrice descrive, tramite gli autovettori, delle direzioni privilegiate dalle quali guardare la trasformazione, così gli autovettori in $S$ ed in $U$ rappresentano l'informazione sui problemi e sugli utenti in modo più comodo.

Lo *sketch* di questo algoritmo risiede nell'interpretazione di $S_k\Sigma_k$ e $\Sigma_k U_k^t$ come rappresentazione essenziale rispettivamente di utenti e *task* in termini di combinazione delle informazioni nascoste.

Concludendo, la **somiglianza** tra più utenti è espressa dalla distanza del coseno tra i due vettori $p_i$ e $p_j$, ossia $\frac{p_i p_j}{|p_i||p_j|}$.

### 2.1.4 Vicini

Una volta trovato un modo efficace di individuare la somiglianza tra utenti, è necessario selezionare, per ogni utente $p_i$ un certo numero $h$ di utenti "simili", individuati dall'insieme $Closest(p_i)$. Questo espediente è fondamentale per collezionare informazioni utili sulle capacità di soluzione degli utenti "simili", così da poter individuare una lista di suggerimenti per l'utente preso in esame.

Un modo per quantificare la fattibilità di un *task* nella cerchia dei vicini dell'utente $p_i$ è quello di ottenere la frequenza di soluzione di ogni *task* su quegli utenti, in termini più formali

$$\forall t_j : m_{i,j} \neq 100 \; freq(t_j) = \sum_{\substack{p_s \in Closest(p_i) \\ m_{s,j}=100}} m_{s,j}$$

In base a questo calcolo si ottiene un ordinamento sui *task* non risolti dall'utente $p_i$, che riflette la loro propedeuticità per quell'utente.

# 3 La pratica

## 3.1 La piattaforma

Il sito di allenamenti `https://cms.di.unipi.it/#/overview` offre qualche centinaia di problemi o *task* agli utenti iscritti, da risolvere senza limiti di tempo nè di numero di sottomissioni. Ad ogni sottomissione viene assegnato un punteggio che va da 0 a 100. Il sistema tiene come *score* su quel problema il migliore tra i punteggi ottenuti su quel determinato *task* da quell'utente.

## 3.2 Le assunzioni

Nella fusione fra teoria e pratica è necessario definire dettagliatamente il sistema in oggetto e precisare tutte le assunzioni, come segue:

Selezione dei dati I dati sul sistema sono talvolta fuorvianti, nel senso che esistono *task* sui quali nessun utente si è cimentato ed esistono utenti che nonhanno provato a risolvere nemmeno un *task*. Per eliminare errori dovuti a questa situazione si è deciso di non tener traccia di tali valori nella costruzione della matrice.

Suggerimenti propedeutici Nella sezione 2.1.4 si prendono in considerazione come *task* suggeriti quelli sui quali l'utente in considerazione non ha totalizzato punteggio pieno (100 punti) su quel problema. La scelta è stata fatta considerando che potrebbe essere formativo riuscire a risolvere completamente un certo *task* prima di passare al successivo, poichè aggiungere anche solo pochi punti allo *score* potrebbe risultare decisivo nel processo di apprendimento.

Sparsità della matrice Dalle evidenze sperimentali e dai risultati ottenuti da *Badrul M. Sarwar* et al.[1] si è deciso di utilizzare la matrice $(user, task)$ desparsificata, dove sono sostiuiti ai valori 0 la media dei punteggi di tutti gli utenti su quel *task*.

## 3.3 La scelta dei parametri

Prima ancora di arrivare al codice vero e proprio è necessario fare alcune considerazioni sulla **scelta dei parametri**. Nella sezione 2.1.2 si parla di $k$, indice con il quale ridurre la taglia delle matrici che decompongono $M$, per abbassare rumore e complessità di calcolo. Inoltre, nella sezione 2.1.4 viene introdotto il valore $h$, che indica il numero di vicini più simili rispetto ad un certo utente $i$ da considerare per trovare i suggerimenti.

La scelta di tali parametri deve essere valutata, infatti intuitivamente la matrice $M_k$ può essere confrontata con $M$ per differenza. Applicando la norma di Frobenius alla differenza tra le due matrici si nota che essa tende a 0 col crescere di $k$. Quindi il metodo di scelta di $k$ non può basarsi sul valore di tale norma, perchè il risultato è chiaramente il $k$ massimo.

Per individuare i valori di $k$ e $h$ con i quali si ottengono suggerimenti più verosimili è stato scelto di confrontare uno *screenshot* di CMS di un mese fa ed uno di adesso, valutando *precision* e *recall* dei risultati prodotti dall'algoritmo. In particolare, sia $A = (a_{i,j}) = \left\lfloor \frac{M_{fine}}{100} \right\rfloor - \left\lfloor \frac{M_{inizio}}{100} \right\rfloor$, la **bontà** dell'algoritmo sui vari parametri è stata calcolata come segue:

$$goodness(k, h) = \sum_{p_i \in P} g(p_i)$$

con

$$g(p_i) = \begin{cases} -l, & se\ A_{i,j} = 1\ e\ t_j \notin s_i \\ l + 1 - pos(t_j), & se\ A_{i,j} = 1\ e\ t_j \in s_i \end{cases}$$

dove $s_i$ è il vettore dei *task* suggeriti all'utente $i$ e $pos(t_j)$ è la posizione del *task* $j$ all'interno di $s_i$.

[1] Application of Dimensionality Reduction in Recommender System – A Case Study by Badrul M. Sarwar, George Karypis, Joseph A. Konstan, John T. Riedl

# 4 Implementazione

## 4.1 La struttura

L'algoritmo descritto sopra è stato implementato in *Python*, perchè potesse essere facilmente inserito sulla piattaforma.

Si struttura in due soli file:

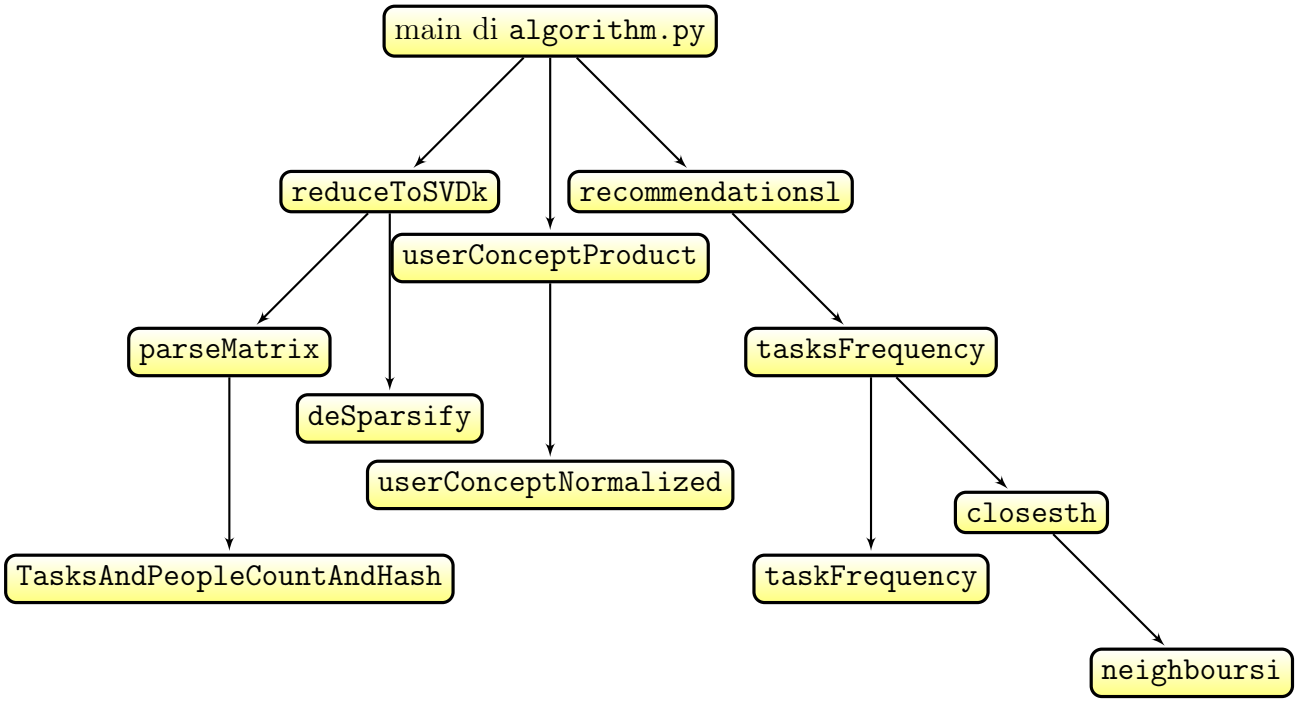
`algorithm.py` Contiene solo lo scheletro dell'algoritmo, con un minimo di UI ed una struttura dati `Useful`, istanziata con `myData`, in modo da incapsulare gli elementi importanti

`functions.py` È il vero e proprio pilastro del programma: definisce le funzioni usate da `algorithm.py` ed alcune funzioni ausiliare, per rendere il codice più snello. Di seguito una breve descrizione di funzioni e procedure principali:

- `deSparsify` modifica la struttura dati rendendo la matrice non sparsa. Si noti che questa funzione può essere chiamata oppure no, a seconda della scelta dell'utente. La funzione calcola, per ogni *task* $j$, la media dei punteggi totalizzati dagli utenti su quel *task* e sostiuisce gli 0 sulla colonna $M^j$ con il valore calcolato.
- `tasksAndPeopleCountAndHash` inizializza alcuni campi della struttura dati e stabilisce una funzione di *hash* per ottenere una matrice più "sintetica" possibile. Infatti, del file di testo che rappresenta CMS in un dato istante, non è che fuorviante prendere in considerazione gli utenti iscritti che non hanno mai inviato una sottomissione o i problemi che non sono mai stati provati da nessun partecipante.
- `parseMatrix` si occupa di leggere e memorizzare i valori dal file nella matrice e di chiamare `tasksAndPeopleCountAndHash`.
- `reduceToSVDk` chiama le funzioni sopracitate aggiornando la struttura dati `myData`, dopodichè riduce $M$ tramite SVD e riduce in base a $k$ le dimensioni delle matrici $S$ e $\Sigma$.
- `userConceptProduct` modifica la struttura dati inizializzando e aggiornando gli elementi della matrice che ha in posizione $(i, j)$ la somiglianza tra l'utente $i$ e l'utente $j$, come definita alla sezione 2.1.3.
- `recommendationsl` per ogni utente restituisce il vettore degli utenti a lui "simili" in ordine decrescente si somiglianza.
- `printMatrix` si occupa di scrivere a video in modo ordinato i risultati ottenuti.

Si verifica facilmente che l'operazione con costo computazionale maggiore è il calcolo dell'SVD.

Di seguito un diagramma che indica la relazione d'uso tra le varie funzioni/procedure:




Per quanto riguarda la scelta dei parametri $k$ e $h$, il codice è stato scritto in *Bash* (per iterare su tutte le coppie e per ridirigere input ed output) ed in *Python* (per eseguire i veri e propri conti, avvalendosi di strutture dati per migliorare il costo computazionale).

Il codice integrale per effettuare tali prove si trova in appendice B, fatta eccezione per le funzioni ausiliarie, che sono le stesse del codice che segue.

## 4.2 Il codice

### 4.2.1 `algorithm.py`

```
1  import math
2  import random
3  import sys
4  import numpy as np
5  import readline
6  from numpy import linalg as LA
7  from functions import *
8
9  """Struttura dati necessaria per lo storage compattato delle
10         informazioni"""
11 class Useful:
12         def __init__(self, k, h, l):
13                 self.k = k
14                 self.h = h
15                 self.l = l
16
```

```
k = 28 #Miglior valore dai test
h = 61 #Numero di vicini da considerare, miglior valore dai test
l = 20 #Numero di suggerimenti, a scelta

print('Recommendation system for CMS')
inputFile  = open("punteggi.txt", "r")
var = input('Press y for recommendations from desparsified matrix, \
press n for recommendations from the original one -> ')
myData = Useful(k, h, l)
if(var == 'y'):
        reduceToSVDk(inputFile, myData, 'yes')
else:
        reduceToSVDk(inputFile, myData, 'no')
suggestionsMatrix = np.empty([myData.p,myData.l], dtype = int)
userConceptProduct(myData)
for useri in range(0, myData.p):
        indexesTasks = recommendationsl(useri, myData)
        for task in range(0, myData.l):
                suggestionsMatrix[useri, task] = indexesTasks[task]
#Comunicazione risultati
var = input('Press a for all the results, press u for a specific user s \
suggestions -> ')
#Nota: solo nell'operazione di print sono applicate le funzioni inverse
#delle funzioni di hash
if(var == 'a'):
        printMatrix(suggestionsMatrix, myData)
else:
        user = input('Insert the user number -> ')
        printRow(suggestionsMatrix, user, myData)
```

### 4.2.2 functions.py

```
import math
import random
import sys
import numpy as np
from numpy import linalg as LA


"""Questa funzione prende l'oggetto myData e aggiorna:
        (1) M -> rendendola non sparsa"""
def deSparsify(myData):
        #ciclo sulle colonne (task)
        for task in range(0, myData.t):
                average = 0
```

```
14                  for counter in range(0, myData.p):
15                          average = average + myData.M[counter, task]
16                  average = average/myData.p
17                  for counter in range(0, myData.p):
18                          if (myData.M[counter, task] == 0):
19                                  myData.M[counter, task] = average
20
21  """Questa funzione prende il file di input e l'oggetto myData e
22          inizializza:
23          (1) t -> numero di task
24          (2) p -> numero di utenti
25          (3) hashPeople -> lista di interi che rappresentano gli utenti
26          (4) hashTasks -> lista di interi che rappresentano i task"""
27  def tasksAndPeopleCountAndHash(inputFile, myData):
28          tasksSet = set()
29          peopleSet = set()
30          data = inputFile.readlines()
31          lines = []
32          for line in data:
33                  words = list(map(int, line.split()))
34                  tasksSet.add(words[1])
35                  peopleSet.add(words[0])
36                  lines.append(words)
37
38          myData.t = len(tasksSet)
39          myData.p = len(peopleSet)
40          myData.hashPeople = sorted(list(peopleSet))
41          myData.hashTasks = sorted(list(tasksSet))
42          return lines
43
44  """Questa funzione prende il file di input e l'oggetto myData e
45          inizializza:
46          (1) M -> matrice che ha come righe gli utenti e per colonne i task
47          (2) t, p, hashPeople, hashTasks -> con tasksAndPeopleCountAndHash"""
48  def parseMatrix(inputFile, myData):
49          lines = tasksAndPeopleCountAndHash(inputFile, myData)
50          myData.M = np.zeros((myData.p,myData.t))
51          for words in lines:
52                  myData.M[myData.hashPeople.index(words[0]),
53                  myData.hashTasks.index(words[1])] = words[2]
54
55  """Questa funzione prende il file di input, l'oggetto myData ed un flag ed
56          inizializza:
57          (1) Sigmak -> matrice risultante da SVD, ma ridotta in taglia
58          (2) Sk -> matrice risultante da SVD, ma ridotta in taglia
59          (3) M, t, p, hashPeople, hashTasks -> mediante parseMatrix
```

```
60          (4) aggiorna M (forse) -> mediante deSparsify"""
61  def reduceToSVDk(inputFile, myData, flag):
62          #Parsa il file in una matrice
63          parseMatrix(inputFile, myData)
64          #Valuta il flag
65          if (flag == 'yes'):
66                  deSparsify(myData)
67          #Calcola la SVD di M
68          S, Sigma, U = np.linalg.svd(myData.M, full_matrices=False)
69          #Approssima M
70          myData.Sigmak = np.diag(Sigma[0:myData.k])
71          myData.Sk = S[...,0:myData.k]
72
73  """Questa funzione prende l'utente user e l'oggetto myData e restituisce:
74          (1) Il vettore di dimensione k, che rappresenta il concetto di
75                  quell'utente"""
76  def userConceptNormalized(user, myData):
77          rowOfSk = myData.Sk[user]
78          userCon = np.matmul(rowOfSk, myData.Sigmak)
79          norm = LA.norm(userCon)
80          userConNorm = userCon/norm
81          return userConNorm
82
83  """Questa funzione prende l'oggetto myData ed inizializza:
84          (1) MatRes -> matrice della somiglianza tra utenti"""
85  def userConceptProduct(myData):
86          Un = np.empty([myData.p, myData.k], dtype = float)
87          for user in range(0, myData.p):
88                  Un[user] = userConceptNormalized(user, myData)
89          myData.MatRes = np.matmul(Un, Un.transpose())
90
91  """Questa funzione prende un utente useri e l'oggetto myData e ritorna:
92          (1) neighboursi -> vettore delle somiglianze tra useri e gli
93                  altri utenti"""
94  def neighboursUi(useri, myData):
95          neighboursi = np.empty([myData.p], dtype = float)
96          neighboursi = myData.MatRes[useri]
97          #A neigbours[useri] è assegnato un valore che non disturba
98          #la somiglianza
99          neighboursi[useri] = 0
100         return neighboursi
101
102 """Questa funzione prende un utente useri e l'oggetto myData e ritorna:
103         (1) indexes -> vettore degli interi che rappresentano gli h
104                 utenti più simili a useri, ordinati in senso decrescente"""
105 def closesth(useri, myData):
```

```
106         indexes = np.empty([myData.h], dtype = int)
107         neighboursi = neighboursUi(useri, myData)
108         minimum = min(neighboursi)
109         for counter in range(0, myData.h):
110                 indexes[counter] = np.argmax(neighboursi)
111                 neighboursi[np.argmax(neighboursi)] = minimum -1
112         return indexes
113
114 """Questa funzione prende l'oggetto myData, un task e indexes e
115         restituisce:
116         (1) frequency -> frequenza di un task su tutti gli h utenti
117                 di indexes"""
118 def taskFrequency(myData, task, indexes):
119         frequency = 0
120         for counter in range(0, myData.h):
121                 frequency = frequency + myData.M[indexes[counter], task]
122         return frequency
123
124 """Questa funzione prende un utente useri e l'oggetto myData e ritorna:
125         (1) frequencies -> array delle frequenze di ogni task per
126                 quell'utente"""
127 def tasksFrequency(useri, myData):
128         frequencies = np.empty([myData.t], dtype = int)
129         indexes = closesth(useri, myData)
130         for task in range(0, myData.t):
131                 frequencies[task] = taskFrequency(myData, task, indexes)
132         return frequencies
133
134 """Questa funzione prende un utente useri e l'oggetto myData e
135         restituisce:
136         (1) indexesTasks -> vettore dei primi l suggerimenti
137                 (tra i task non risolti completamente) per l'utente useri"""
138 def recommendationsl(useri, myData):
139         indexesTasks = np.empty([myData.l], dtype = int)
140         frequencies = tasksFrequency(useri, myData)
141         minimum = min(frequencies)
142         counter = 0
143         while counter < myData.l:
144                 currentTask = np.argmax(frequencies)
145                 frequencies[currentTask] = minimum -1
146                 if (myData.M[useri, currentTask] != 100):
147                         indexesTasks[counter] = currentTask
148                         counter = counter + 1
149         return indexesTasks
150
151 """Questa funzione prende la matrice suggestionsMat e l'oggetto myData
```

```
152         e chiama printRow per ogni riga, NOTA: deve passare per la
153         funzione di hash"""
154 def printMatrix(suggestionsMat, myData):
155         for user in range(0, myData.p):
156                 printRow(suggestionsMat, myData.hashPeople[user], myData)
157
158 """Questa funzione prende la matrice suggestionsMat, un utente
159         e l'oggetto myData e stampa a video i suggerimenti per
160         quell'utente applicando la funzione la funzione inversa
161         della funzione di hash sull'utente e sul task"""
162 def printRow(suggestionsMat, user, myData):
163         print(user, end=' ')
164         hashedUser = myData.hashPeople.index(int(user))
165         for counter in range(0, myData.l):
166                 print(
167                 myData.hashTasks[suggestionsMat[hashedUser, counter]],
168                 end=' ')
169         print('')
```

# 5 Risultati

Dopo aver provato tutte le coppie $(k, h)$, per valori interi di $k \in [0, \cdots, 30]$ e $h \in [0, \cdots, 300]$ è stata individuata la coppia $(28, 61)$, come migliore e da qui si è svolto il vero e proprio *testing* del programma. I risultati di questa parte sono discussi in maniera più dettagliata in appendice B.

Per valutare l'affidabilità dell'algoritmo ai fini di guidare l'apprendimento degli studenti si è ritenuto un valido approccio quello di richiedere proprio a loro un *feedback* del prodotto.

Sono stati selezionati 5 utenti di CMS, di età compresa tra i 16 ed i 19 anni, che si sono proposti di affrontare un *task* per una durata di tre giorni. Il loro obiettivo è stato quello di confrontarsi col problema non tanto con lo scopo di risolverlo, quanto con lo scopo di "valutarlo". Si è trovato, infatti, più interessante conoscere (oltre al punteggio totalizzato) le impressioni sul problema.

Di seguito i risultati:

RAGAZZO - 19 ANNI ha risolto il *task* assegnatogli molto velocemente. L'ha ritenuto inferiore alle sue capacità (e quindi inutile nel processo di apprendimento), ma ottimo come ripasso, poichè "era una vita che non mi allenavo su un problema greedy!". Si è prestato per un secondo tentativo, che è risultato molto più felice, in particolare ha impiegato più tempo per risolverlo, la ricerca della soluzione lo ha coinvolto intellettualmente e "allo stesso tempo non era esageratamente difficile, quindi direi che è stato azzeccato in pieno".

RAGAZZO - 16 ANNI ha risolto il primo suggerimento in poco tempo, ma lo ha gradito molto, in quanto aveva già guardato il testo in passato e aveva iniziato a pensarci, senza però lasciare traccia di tale movimento nel sistema. Il secondo suggerimento,

ricalcolato dopo la soluzione del primo, ha avuto un esito meno roseo, infatti è stato recensito come "decisamente troppo facile".

Ragazzo - 19 anni ha risolto il *task* assegnatogli in un tempo medio. Ritiene che il problema sia stato un buon suggerimento "in quanto non era scontato".

Ragazzo - 19 anni ha risolto il problema in un tempo breve e lo ha ritenuto inferiore alle sue capacità, ma afferma che il suo profilo su CMS non corrisponde alle sue reali capacità di solutore, poichè tende a confrontarsi con problemi troppo facili.

Ragazzo - 18 anni ha risolto il secondo problema assegnato con punteggio pieno, ha invece completamente ignorato il primo, per motivi imprecisati, cimentandosi su altri problemi. Non ha espresso alcun parere in merito, quindi non è possibile estrapolare ulteriori informazioni.

È ora necessario commentare un minimo questi risultati: la piattaforma CMS al momento attuale è popolata da un numero notevole di problemi di media difficoltà, quindi anche i solutori bravi possono essere vittima di una "fluttuazione statistica" e vedersi assegnato un *task* al di sotto delle loro capacità.

Inoltre, è stato possibile verificare che in alcuni casi le capacità del solutore non sono in accordo con la difficoltà dei *task* con i quali si cimenta. Come detto da alcuni dei *tester*, infatti, non necessariamente i problemi che una persona svolge su CMS sono di difficoltà adeguata alle proprie capacità di soluzione. Dato che l'algoritmo ottiene informazioni solo dalla piattaforma, è naturalmente impossibile richiedere un comportamente migliore.

# 6 Conclusioni

Alla luce dei dati raccolti dalla sperimentazione si può affermare che l'algoritmo è corretto e che potrebbe rappresentare un notevole miglioramento nell'ambito di quello che è di fatto un apprendimento di tipo autodidatta.

Non sono tuttavia noti risultati a lungo termine.

# A Cenni di SVD

Si supponga di avere i dati della sezione 2.1.1, senza interesse all'interpretazione che è stata data nell'ambito della piattaforma CMS. Ossia i seguenti

- $M \in M(p, t, \mathbb{N})$, con $p > t$
- $B = M^t M \in S(p, \mathbb{N})$
- $C = MM^t \in S(t, \mathbb{N})$

Valgono i seguenti lemmi:

**Lemma A.1**
*$B$ e $C$ sono simmetriche.*

*Dimostrazione.*
$B^t = (M^t M)^t = M^t M = B$
$C^t = (MM^t)^t = MM^t = C$ □

**Lemma A.2**
*Se $N = R^t R$ allora $N$ è semi-definita positiva.*

*Dimostrazione.* La tesi equivale a $x^t R^t R x \geq 0$, ma $x^t R^t R x = (Rx)^t (Rx) \geq 0$, perchè il prodotto scalare standard è definito positivo. □

Valgono inoltre le ipotesi del seguente teorema:

**Teorema A.3 (Teorema spettrale)**
*Se $Q \in S(n, \mathbb{R})$ esistono $x_1, x_2, \cdots, x_n$ autovettori ortonormali di $Q$, con autovalori $\lambda_1, \lambda_2, \cdots, \lambda_n$ reali.*

**Corollario A.4**
*Sia $B$ che $C$ hanno autovalori reali non negativi.*

Tali autovalori sono dunque quadrati di numeri reali non negativi, ordinati in senso decrescente $\sigma_1^2 \geq \sigma_1^2 \geq \cdots \geq \sigma_t^2$, tali che $\sigma_1, \sigma_2, \cdots, \sigma_r \in \mathbb{R}^+$ e $\sigma_{r+1}, \cdots, \sigma_t = 0$.

Quindi sia

$$U = \left( \begin{array}{c|c|c|c} & & & \\ x_1 & x_2 & \cdots & x_r \\ & & & \end{array} \right) \in M(t, r, \mathbb{R})$$

la matrice che ha per colonne gli autovettori ortonormali di $B$ relativi ad autovalori positivi.

Siano $y_i = \frac{1}{\sigma_i} M x_i \ \forall i = 1, \cdots, r$, allora vale il seguente lemma:

**Lemma A.5**
*Gli $y_i \ \forall i \in \{1, \cdots, r\}$ sono autovettori ortonormali per $C$.*

*Dimostrazione.*

AUTOVETTORI È possibile riscrivere la tesi come $Cy_i = \lambda_i y_i$, ovvero $MM^t y_i = \lambda_i y_i$.

Si ha

$$MM^t y_i = MM^t \left(\frac{1}{\sigma_i} M x_i\right) = M\left(\frac{1}{\sigma_i} M^t M x_i\right) = M\left(\frac{1}{\sigma_i}\sigma_i^2 x_i\right) = \sigma_i^2 \frac{1}{\sigma_i} M x_i = \sigma_i^2 y_i$$

che corrisponde alla prima parte della tesi scegliendo $\lambda_i = \sigma_i^2$.

ORTONORMALI Vale la seguente catena di uguaglianze

$$\begin{aligned} y_i^t y_j &= \left(\frac{1}{\sigma_i} M x_i\right)^t \frac{1}{\sigma_j} M x_j \\ &= \frac{1}{\sigma_i \sigma_j} x_i^t M^t M x_j \\ &= \frac{1}{\sigma_i \sigma_j} x_i^t B x_j \\ &= \frac{1}{\sigma_i \sigma_j} x_i^t \sigma_j^2 x_j \\ &= \frac{\sigma_j}{\sigma_i} x_i^t x_j \end{aligned}$$

Quindi, poichè $x_i$ e $x_j$ sono ortonormali, si ha la tesi.

□

Sia

$$S = \left(\begin{array}{c|c|c|c} & & & \\ y_1 & y_2 & \cdots & y_r \\ & & & \end{array}\right) \in M(p, r, \mathbb{R})$$

la matrice che ha per colonne gli autovettori ortonormali relativi ad autovalori non nulli di $C$ e si consideri la matrice $\Sigma = S^t M U$. Un suo generico elemento $(i, j)$ vale $(S^t M U)_{ij} = y_j^t M x_i = y_j^t \sigma_i y_i = \sigma_i y_j^t y_i$, quindi, poichè gli $y_i$ sono ortonormali, tale matrice è diagonale con elementi $\sigma_1 \dots \sigma_r$.

Inoltre, poichè $S$ e $U$ hanno per colonne vettori ortonormali, $SS^t = I_p$ e $UU^t = I_d$, quindi è possibile moltiplicare l'uguaglianza $S^t M U = \Sigma$ a sinistra per $S$ e a destra per $U^t$, ottenendo il seguente teorema:

**Teorema A.6**
*Sia $M \in M(p, t, \mathbb{R})$ e siano $B = M^t M$, $C = MM^t$, $U \in M(t, r, \mathbb{R})$ matrice che ha per colonne gli autovettori ortonormali relativi ad autovalori non nulli di $B$ e $S \in M(p, r, \mathbb{R})$ matrice che ha per colonne gli autovettori ortonormali relativi ad autovalori non nulli di*

*C. Allora la matrice* $\Sigma = S^t MU$ *è diagonale e ha per elementi le radici quadrate positive degli autovalori della matrice* $B$*, ossia*

$$S^t MU = \Sigma = \begin{pmatrix} \sigma_1 & & & & 0 \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_{r-1} & \\ 0 & & & & \sigma_r \end{pmatrix}$$

*Inoltre vale che* $M = S\Sigma U^t$*.*

# B Calcolo dei parametri

## B.1 Il codice

### B.1.1 Lo script bash

```
#!bin/bash

rm -f goodness.txt

for h in $(seq 1 300)
do
        for k in $(seq 1 30)
        do
                echo -n "h = $h, k = $k, goodness = " >> goodness.txt
                echo -e "$h\n$k\ny\na\n" | python algorithm.py | python test.py >> \
                        goodness.txt
        done
done
```

### B.1.2 Il programma Python

```
import math
import random
import sys
import readline

goodness = 0
fullFile  = open("full.txt", "r")
data = fullFile.readlines()
couples = set()
for line in data:
        words = list(map(int, line.split()))
```

```
12          couples.add((words[0], words[1]))
13
14 data = sys.stdin.readlines()
15 lines = []
16 count = 0
17 for line in data:
18          words = list(map(int, line.split()))
19          for i in range(1, 21):
20                  if (words[0], words[i]) in couples:
21                          count = count + 1
22                          goodness = goodness + 21 - i
23
24 F = len(couples)
25 Inter = count
26 goodness = goodness + (- F + Inter) * 20
27
28 print(goodness)
```

### B.1.3 L'algoritmo senza interfaccia grafica

```
1  import math
2  import random
3  import sys
4  import numpy as np
5  import readline
6  from numpy import linalg as LA
7  from functions import *
8
9  """Struttura dati necessaria per lo storage compattato delle informazioni"""
10 class Useful:
11          def __init__(self, k, h, l):
12                  self.k = k
13                  self.h = h
14                  self.l = l
15
16 l = 20 #Numero di suggerimenti
17
18 inputFile  = open("punteggi.txt", "r")
19 h = input('')
20 k = input('')
21 var = input('')
22 myData = Useful(int(k), int(h), l)
23 if(var == 'y'):
24          reduceToSVDk(inputFile, myData, 'yes')
25 else:
```

```
26          reduceToSVDk(inputFile, myData, 'no')
27  suggestionsMatrix = np.empty([myData.p,myData.l], dtype = int)
28  userConceptProduct(myData)
29  for useri in range(0, myData.p):
30          indexesTasks = recommendationsl(useri, myData)
31          for task in range(0, myData.l):
32                  suggestionsMatrix[useri, task] = indexesTasks[task]
33  #Comunicazione risultati
34  var = input('')
35  #Nota: solo nell'operazione di print sono applicate le funzioni inverse delle
36  #funzioni di hash
37  if(var == 'a'):
38          printMatrix(suggestionsMatrix, myData)
39  else:
40          user = input('Insert the user number -> ')
41          printRow(suggestionsMatrix, user, myData)
```

## B.2 I risultati

Il file `goodness.txt`, ottenuto dall'esecuzione completa dello script *bash* conta centinaia di righe, quindi si è preferito non inserirlo in appendice, bensì in allegato. Di seguito il grafico tridimensionale della bontà dell'algoritmo in funzione di $k$ e $h$.

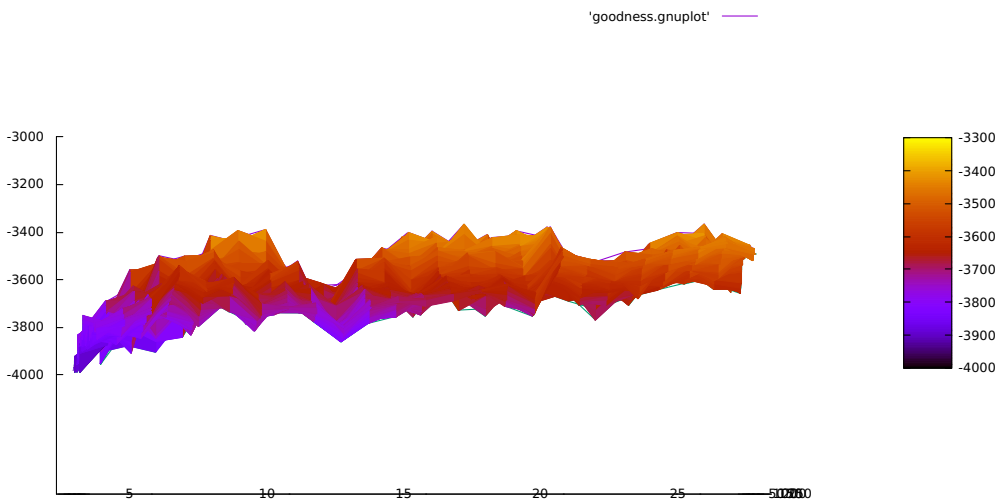



Figura 1: Bontà in funzione di $k$ e $h$, con $k$ in ascissa

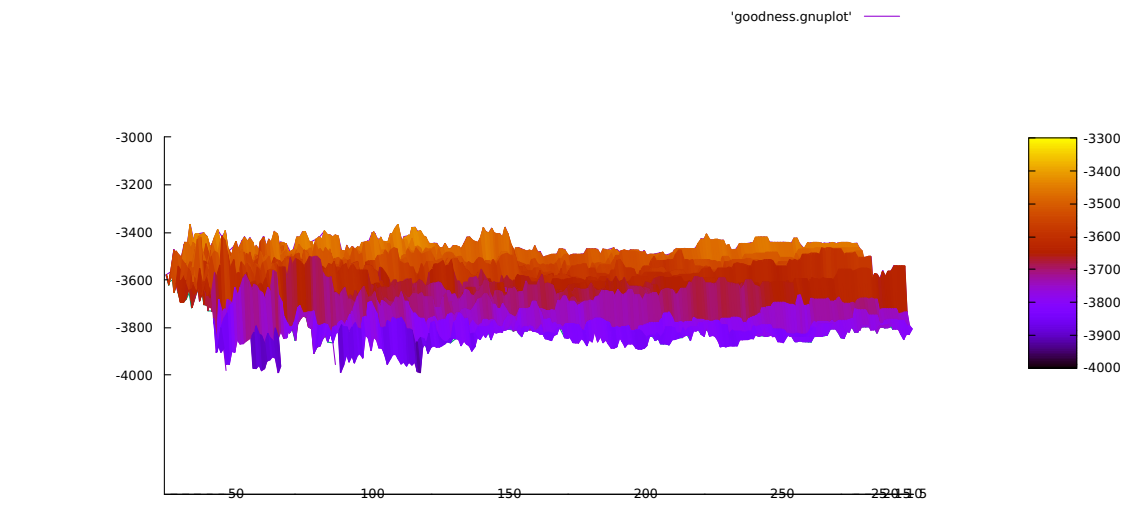



Figura 2: Bontà in funzione di $k$ e $h$, con $h$ in ascissa

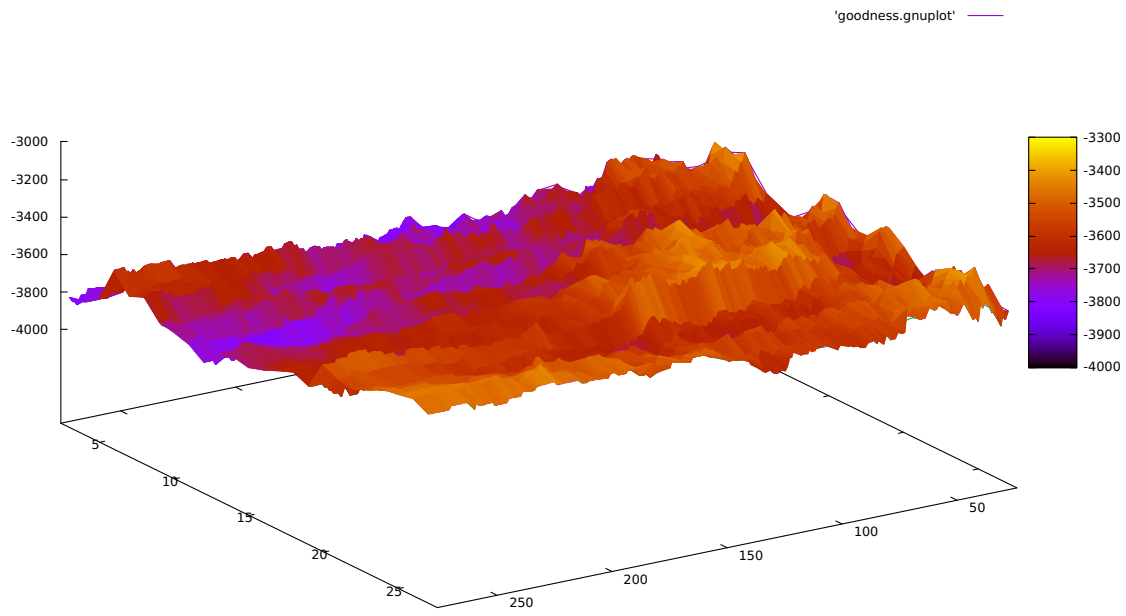



Figura 3: Bontà in funzione di $k$ e $h$