



**We Help  
You Sell**

## Modulo FIA

Prepared for: progetto IS

Prepared by: Alex

18 December 2021

---

---

# SOMMARIO

1 Introduzione

2 Descrizione dell'agente

    2.1 Obiettivi

    2.2 Specifica PEAS

    2.3 Analisi del problema

3 Big Data

4 Implementazione degli algoritmi

5 Integrazione con la piattaforma

---

---

## 1. INTRODUZIONE

Partiamo dall'inizio, Help Seller è una piattaforma che si occupa di digitalizzare il rapporto tra le **aziende**, ed i **distributori**.

L'azienda è l'attore protagonista che dispone di un prodotto ed ha la necessità di venderlo sul mercato.

Il distributore invece è quella figura che compra il prodotto direttamente dall'azienda ed ha il task di **distribuirla** ai vari punti vendita con cui interagisce la massa, stiamo parlando quindi di realtà come ristoranti, negozi, supermercati etc.

## 2. DESCRIZIONE DELL'AGENTE

### 2.1 Obiettivi

L'idea della proposta è quella di adottare algoritmi di machine learning data driven per guidare gli utenti all'esplorazione del catalogo fornito dalla piattaforma; aiutare l'utente a trovare prodotti interessanti così come dare la possibilità alle aziende di mettere in mostra i propri articoli.

Sviluppare un motore di raccomandazione in grado di gestire le seguenti situazioni:

1. **Utenti appena registrati**, e quindi riuscire a consigliare a qualcuno senza conoscerlo
2. **Utenti regolari**, tracciare un profilo degli utenti nel tempo, determinare le loro abitudini d'acquisto per fare previsioni sempre più efficaci
3. **Nuove aziende** e quindi **nuovi prodotti**, dare l'opportunità agli elementi del catalogo appena inseriti di poter compiere con quelli che invece sono già presenti

## 2.2 Specifica PEAS

PEAS	
<b>Performance</b>	La misura di performance dell'agente è direttamente proporzionale alla sua capacità nel riuscire a predire i prodotti che un utente è intenzionato a comprare, talvolta senza neanche che l'utente stesso ne sia consapevole.
<b>Environment</b>	L'ambiente in cui opererà l'agente è composto dall'insieme di tutti i prodotti presenti sulla piattaforma Help Seller, in congiunzione all'insieme di tutti gli utenti iscritti al sito con particolare enfasi sulla storia degli utenti, e quindi sulla loro cronologia d'acquisto. Si tratta di un ambiente: <ul style="list-style-type: none"><li>● <b>Dinamico</b> fin quando la piattaforma resta online, un utente potrebbe effettuare un ordine cambiando le proprie preferenze</li><li>● <b>Sequenziale</b> le azioni passate degli utenti influenzano le decisioni future dell'agente</li><li>● <b>Completamente Osservabile</b> si ha accesso a tutte le informazioni relative agli utenti ed ai prodotti in ogni momento</li><li>● <b>Discreto</b> il prezzo dei prodotti viene approssimato a due cifre decimali</li><li>● <b>Stocastico</b> lo stato dell'ambiente cambia indipendentemente dalle azioni dell'agente</li><li>● <b>Singolo agente</b> tecnicamente ci potrebbe essere una situazioni in cui più agenti operano in simultanea, tuttavia senza influenzarsi a vicenda.</li></ul>
<b>Actuators</b>	Gli attuatori consistono nella lista dei prodotti suggeriti sulla base del profilo dell'utente.
<b>Sensors</b>	I sensori dell'agente sono le view fatte al database per reperire informazioni necessarie al fine di prendere decisioni data driver

## 2.3 Analisi del problema

Il problema delle raccomandazioni può essere risolto con ciascuna delle tre tipologie di modello di machine Learning studiate nel corso. Potremmo avere un modello di classificazione andando ad associare una classe ad ogni utente o articolo, anche se è evidente l'inefficacia di tale risoluzione. Si potrebbe adottare un modello di Clustering e cercare di raggruppare tra loro gli users o gli items omogenei in clusters cercando di trovare pattern nascosti, ma non ne vedo la necessità in quanto tutte le informazioni necessarie ad una raccomandazione sono perfettamente visibili ed utilizzabili dal nostro eventuale modello. Pertanto ho deciso di utilizzare la tipologia di sistema di raccomandazione più diffusa da almeno due decenni a questa parte, ovvero quella che adopera modelli di regressione, nello specifico l'applicazione di metodi matematici come la correlazione di Pearson e la decomposizione ai valori singolari (tanto per dirne qualcuno) al fine di ottenere una variabile continua, indice del grado di affinità tra diversi utenti o diversi item.

---

Oggi i sistemi (o motori) di raccomandazione si suddividono in due tipologie:

## COLLABORATIVE FILTERING

In italiano “filtraggio collaborativo”, spesso abbreviato come “CF”;

Si intende una classe di strumenti e meccanismi che consentono il recupero di informazioni predittive relativamente agli interessi di un insieme dato di utenti. L'assunzione fondamentale dietro il concetto di collaborative filtering è che ogni singolo utente che ha mostrato un certo insieme di preferenze continuerà a mostrarle in futuro. Il sistema genererà raccomandazioni cercando relazioni tra utenti (tipologia denominata **user-based**) o tra gli item (tipologia detta **item-based**).

User / Item	Batman	Star Wars	Titanic
Bill	3	3	
Jane		2	4
Tom		5	

Esempio di matrice di una matrice di correlazione, tipica nei modelli CF. Tale matrice mette in relazione gli utenti (Bill, Jane e Tom) con gli items (in questo caso i film Batman, Star Wars e Titanic). Andando a confrontare la coppia di utenti Bill e Jane, notiamo che Bill non ha visto Titanic e Jane non ha visto Batman, pertanto la relazione tra i due utenti è rafforzata soltanto da Star Wars.

---

---

Una delle caratteristiche del CF è che non ha bisogno di capire gli item che dovrà andare a raccomandare, ciò però non implica che si tratti di un modello non supervisionato, semplicemente si concentra su altre caratteristiche label included quali ad esempio la valutazione che un utente ha fatto ad un determinato prodotto.

L'approccio tramite collaborative filtering soffre delle seguenti problematiche:

- **Cold start**

Quando un nuovo item o utente è aggiunto, esso non è associato a nessuna interazione, per tanto il modello non sarà in grado di gestirlo.

- **Scalability**

Potenzialmente nella piattaforma ci potrebbero essere milioni di utenti e prodotti, richiedendo un'enorme potenza di calcolo per il calcolo delle raccomandazioni.

- **Sparsity**

Il quantitativo di articoli venduti su un e-commerce è potenzialmente enorme; al tempo stesso persino gli utenti più attivi valuteranno solo una piccola porzione dell'insieme complessivo degli articoli, di conseguenza anche gli articoli più popolari avranno un quantitativo ristretto di valutazioni.

## User based

nell'andare a fare una raccomandazione, vengono individuati gli utenti dagli interessi simili analizzando il loro comportamento, - nel caso di Help Seller stiamo parlando di un ordine effettuato o di una recensione - e suggerire loro gli elementi che li distinguono.

Ad esempio abbiamo due utenti A e B tali che entrambi abbiano apprezzato gli item 1, 2 e 3. Un sistema user based noterà la relazione tra i due utenti in quanto è presente una congiunzione nell'insieme degli item d'interesse di A e B; pertanto, qualora l'utente A faccia un ordine dell'item 4, sulla base della relazione tra i due utenti, a B verrà suggerito l'articolo 4 perché se è piaciuto ad A è probabile che piaccia anche a B, ce lo fa pensare la loro relazione di affinità.

---

---

Limitazioni della tipologia user-based:

- Il sistema è poco performante se relativamente parlando ci sono molti item ma poche recensioni
- Calcolare la similarità tra tutte le coppie di utenti è costoso
- I profili degli utenti sono entità dinamiche che cambiano spesso e in breve tempo, costringendo l'intero modello del sistema ad essere ricalcolato.

## Item based

Inventata da Amazon nel 1998, questa tipologia è concettualmente identica a quella User based ma l'applicazione è eseguita al contrario: anziché cercare relazioni tra utenti, vengono cercate relazioni tra gli articoli del catalogo.

“Le persone che valutano positivamente l’item 1 come te, tendono a valutare positivamente anche l’item 2, siccome tu non hai ancora valutato l’item 2 dovrà provarlo”

Questa tipologia di collaborative filtering risolve le limitazioni appartenenti alla sua controparte, o almeno nei sistemi aventi più utenti di items; se questa condizione è vera ad ogni item saranno associate molteplici valutazioni, sicché il voto medio di un articolo non cambierà troppo frequentemente. Questo comporta una distribuzione più stabile nel modello minimizzando la necessità di andare a ricalcolarlo. Quando un utente valuta un nuovo item, verranno presi gli item aventi una relazione a quello appena acquistato e aggiunti nelle raccomandazioni dell’utente.

## CONTENT-BASED FILTERING

Come detto prima, una delle caratteristiche del CF è il non preoccuparsi del contenuto effettivo degli articoli che si vanno a raccomandare.

Caso contrario è quello che troviamo nel content-based filtering, infatti la metodologia di filtering caratteristica dei content-based recommenders si basa sulla descrizione tramite keywords degli items e su un profilo delle preferenze dell’utente. Tale metodologia è quindi preferibile in situazioni in cui siamo a conoscenza delle caratteristiche del prodotto ma non dell’utente.

---

---

I content-based raccomanders trattano i suggerimenti come problemi di **classificazione** specifici al singolo user. Ad ogni utente è associato un profilo dove vengono descritte le sue preferenze e avversioni relative alle caratteristiche degli items.

In poche parole, il modello cercherà di suggerire all'utente articoli dalle caratteristiche simili a quelle degli articoli precedentemente acquistati all'utente in questione.

## MATRICE DI DECOMPOSIZIONE

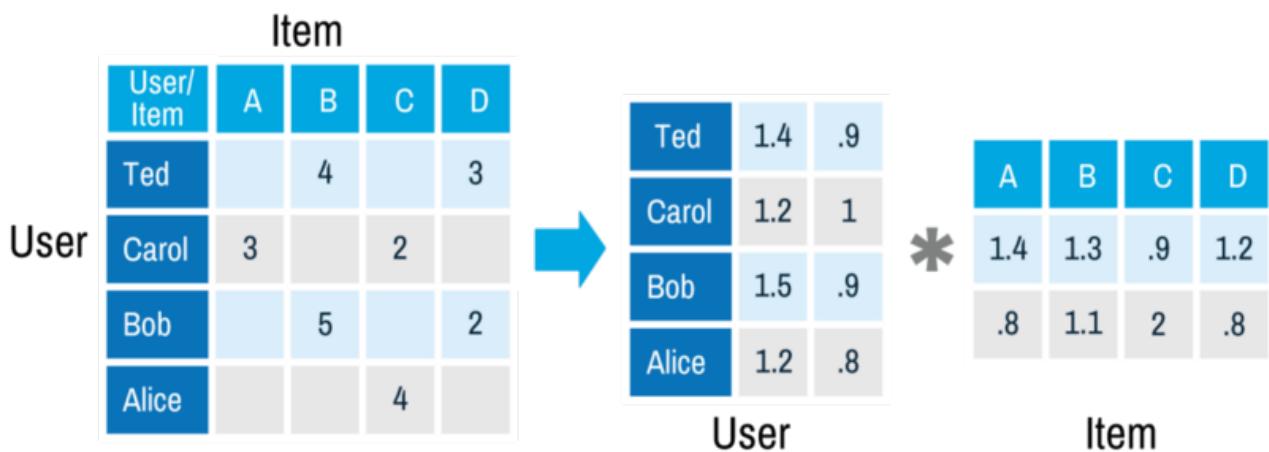
Considerato da alcuni come uno degli approcci più eleganti per le raccomandazioni, utilizza il metodo della decomposizione a valore singolare per determinare l'affinità tra user e items.

$$x_{ij} \approx \langle u_i, v_j \rangle$$

$$\sum_{i,j} (\langle u_i, v_j \rangle - x_{ij})^2 \rightarrow \min$$

È possibile approssimare  $x$ , ovvero il valore della valutazione dell'i-esimo utente per il j-esimo film applicando il prodotto scalare ad  $u$  e  $v$ . Tali vettori verranno costruiti sulla base delle valutazioni disponibili e grazie a queste verranno determinate le valutazioni sconosciute (perché non ancora effettuate dall'utente e quindi non presenti nel database).

---



Per esempio, dopo aver calcolato la matrice abbiamo i vettori  $(1.4; .9)$  per l'utente 'Ted' ed il vettore  $(1.4; .8)$  per il film A. Ora possiamo stimare una possibile valutazione di Ted per il film A calcolando il prodotto scalare di  $(1.4; .9)$  e  $(1.4; .8)$  ottenendo **2.68**.

### 3. BIG DATA

Lo sviluppo del motore di raccomandazione si può dire che è diviso in due parti; La prima parte è quella di progettazione, in cui ho studiato le diverse tecnologie applicabili al problema alla ricerca della soluzione più adatta. Questa fase è antecedente all'implementazione del codice del progetto Help Seller, pertanto nel testare il modello ho fatto ricorso a diversi dataset trovati su un'eccellente piattaforma: [kaggle.com](http://kaggle.com). Un dataset in particolare che mi è tornato utile nella fase di sviluppo dell'algoritmo è composto da due milioni di articoli di cosmetica venduti su [amazon.com](http://amazon.com)

Nella seconda parte dello sviluppo del progetto il dataset coincide al database della piattaforma HelpSeller, difatti tutto ciò che è necessario al modello per poter fare un suggerimento è contenuto all'interno della table “recensione”:

	idrecensione	testo	voto	data	idProdotto	idDistributore
▶	1	adoro il piccante	5	NULL	9	1
	2	adoro il piccante	5	NULL	8	1
	3	adoro il piccante	5	NULL	10	1
	4	buona la cola	5	NULL	1	2
	5	buona la cola	5	NULL	2	2
	6	buona la cola	5	NULL	3	2
	7	ottimo surgelato	5	NULL	4	3
	8	ottimo surgelato	5	NULL	5	3
	9	ottimo surgelato	5	NULL	6	3
	10	anche a me piace il piccante	5	NULL	9	4
	11	sono un patito della cola	5	NULL	1	5
	12	molto comodi da cucinare	5	NULL	4	6
	13	pessimo sapore	1	NULL	3	4
	14	troppo piccante	1	NULL	10	5
*	NULL	NULL	NULL	NULL	NULL	NULL

## 4. IMPLEMENTAZIONE

### UTENTI APPENA REGISTRATI

Stiamo trattando il problema del **cold start** definito in precedenza, ovvero introducendo un nuovo elemento nella piattaforma, il modello non è in grado di gestirlo per carenza di informazioni.

Nella soluzione implementata verranno individuati quelli che risultano essere i prodotti più popolari tra gli altri utenti della piattaforma e suggeriti ai nuovi utenti. L'idea di base è che da un punto di vista statistico, se una cosa piace ad n persone, è probabile che possa piacere anche alla persona n+1

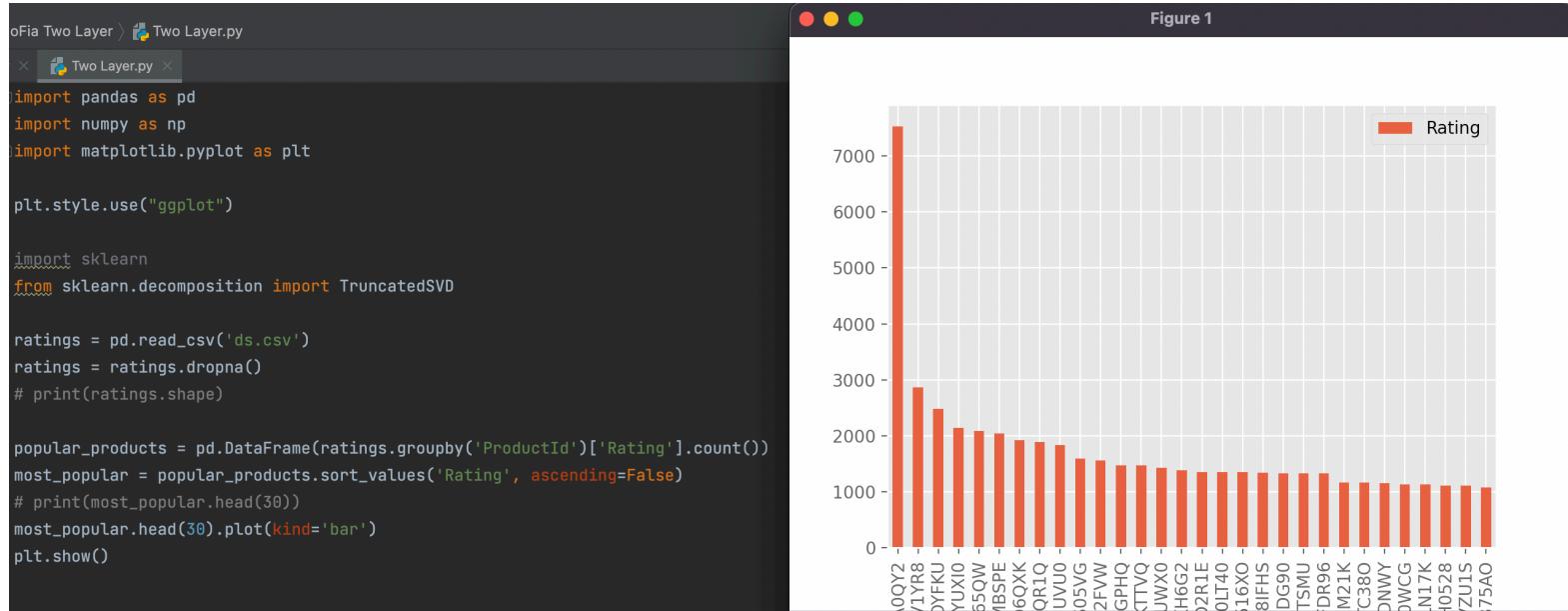


Immagine di una view con grafico a barre degli articoli più popolari (con più recensioni) del dataset “2 milioni”. Parecchi dei grafici che mi sono tornati utili durante lo sviluppo del progetto e presenti in questa documentazione sono stati creati tramite la libreria *matplotlib* di Python.

## UTENTI REGOLARI

Il nostro motore di raccomandazione sarà grado di riconoscere le abitudini d’acquisto degli utenti analizzando i prodotti che ha ordinato e confrontandolo con altri utenti che hanno ordinato prodotti simili

1. Tracciare un profilo dell’user (il distributore)
2. Trovare correlazioni o affinità tra diversi profili
3. Predire quali prodotti interessano a quali utenti sulla base del loro profilo

Tutti questi obiettivi sono stati raggiunti con l’implementazione di un motore di raccomandazione basato sul collaborative filtering item based.

## Tracciare un profilo Utente

Come già detto sopra, all'interno del nostro database abbiamo tutti i dati che ci servono. Verrà mappata una matrice avente per indici gli utenti e per colonne i prodotti. Il valore di tale matrice sarà il rating che l'utente ha dato al prodotto in seguito ad un acquisto.

```
X = ratings.pivot_table(values='voto', index='idProdotto', columns='idDistributore', fill_value=0)
```

	1	2	3	4	5	6
1	0	5	0	0	5	0
2	0	5	0	0	0	0
3	0	5	0	1	0	0
4	0	0	5	0	0	5
5	0	0	5	0	0	0
6	0	0	5	0	0	0
8	5	0	0	0	0	0
9	5	0	0	5	0	0
10	5	0	0	0	1	0

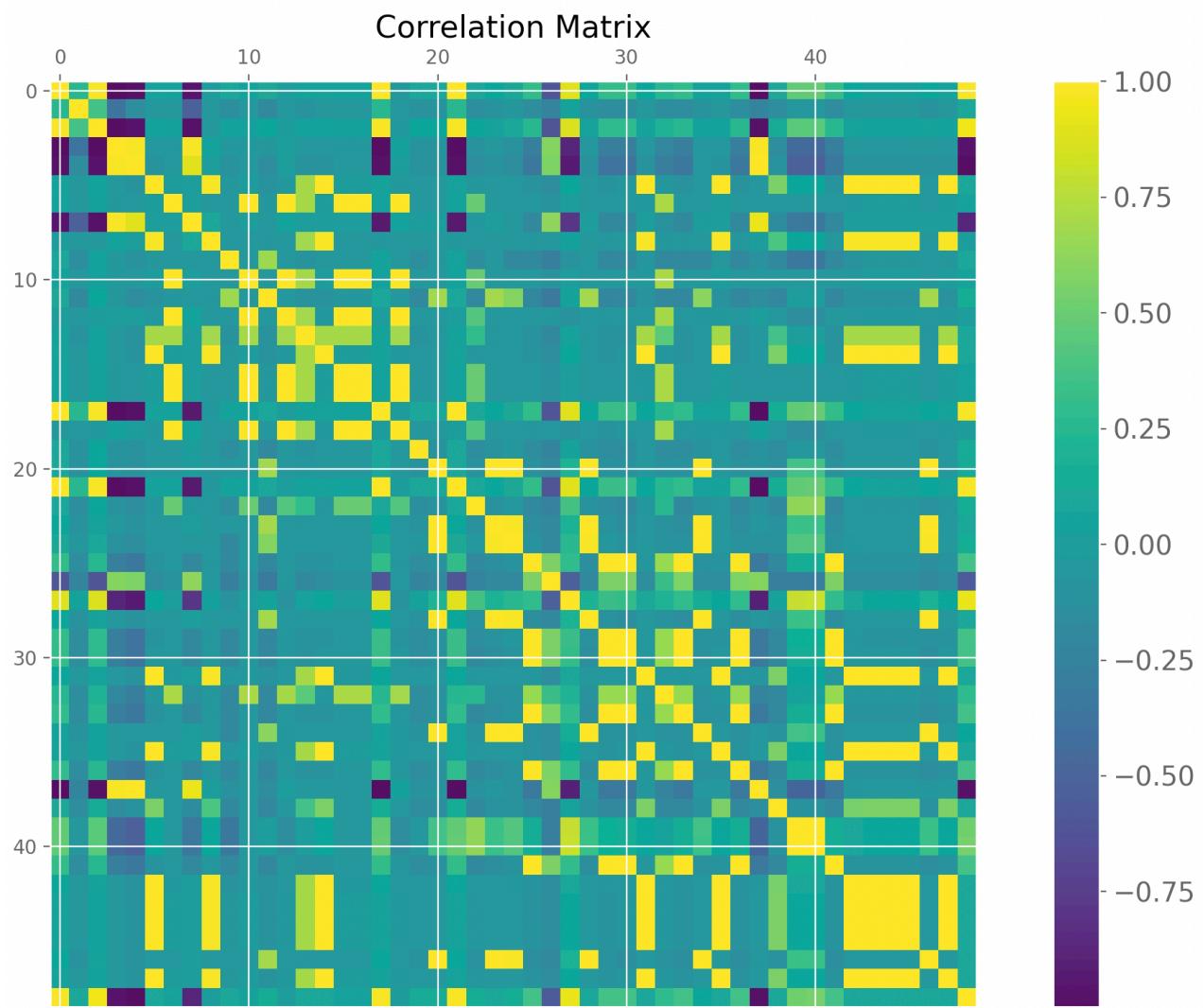
La matrice ottenuta mappando i dati presenti nella table “recensione” mostrata sopra.

## Trovare affinità tra utenti

```
SVD = TruncatedSVD(n_components=5)
decomposed_matrix = SVD.fit_transform(X)
# print(decomposed_matrix.shape)

correlation_matrix = np.corrcoef(decomposed_matrix)
# print(correlation_matrix.shape)
```

È possibile determinare il grado di affinità tra users & items applicando il metodo della decomposizione ai valori singolari, il tutto è già implementato nel package sklearn.decomposition di Python.



Nella matrice di correlazione è evidente la diagonale che intercorre nel mezzo, tale diagonale rappresenta il fattore di correlazione tra l'utente e se stesso, per tanto assume valore massimo. Più il valore contenuto nella cella si avvicina ad 1, maggiore sarà l'affinità tra i due utenti.

La matrice dell'immagine è applicata al dataset ‘2 milioni’ prendendo in considerazione i primi 50 elementi

	0	1	2	3	4	5	6	7
0	1.00000	-0.40578	0.68452	0.73799	0.02826	-0.26825	-0.27955	0.51209
1	-0.40578	1.00000	-0.88110	0.19059	0.88513	0.94811	0.96573	-0.99039
2	0.68452	-0.88110	1.00000	0.12718	-0.57556	-0.85841	-0.84484	0.93373
3	0.73799	0.19059	0.12718	1.00000	0.56644	0.37853	0.35867	-0.09742
4	0.02826	0.88513	-0.57556	0.56644	1.00000	0.86976	0.89906	-0.81786
5	-0.26825	0.94811	-0.85841	0.37853	0.86976	1.00000	0.99327	-0.93988
6	-0.27955	0.96573	-0.84484	0.35867	0.89906	0.99327	1.00000	-0.95047
7	0.51209	-0.99039	0.93373	-0.09742	-0.81786	-0.93988	-0.95047	1.00000
8	-0.26822	0.94806	-0.85843	0.37858	0.86969	1.00000	0.99324	-0.93984
9	0.25920	-0.88369	0.82712	-0.37914	-0.80178	-0.97670	-0.96961	0.88073
10	-0.27955	0.96573	-0.84484	0.35867	0.89906	0.99327	1.00000	-0.95047
11	0.25776	-0.87412	0.82389	-0.37852	-0.79045	-0.97242	-0.96413	0.87238
12	-0.27991	0.96605	-0.84438	0.35794	0.89965	0.99290	1.00000	-0.95062
13	-0.27057	0.95225	-0.85681	0.37531	0.87595	0.99978	0.99549	-0.94274
14	-0.26825	0.94811	-0.85841	0.37853	0.86976	1.00000	0.99327	-0.93988
15	-0.27955	0.96573	-0.84484	0.35867	0.89906	0.99327	1.00000	-0.95047
16	-0.27955	0.96573	-0.84484	0.35867	0.89906	0.99327	1.00000	-0.95047
17	0.45162	-0.13937	0.53909	0.22376	0.18698	-0.31289	-0.23859	0.24063
18	-0.27991	0.96605	-0.84438	0.35794	0.89965	0.99290	1.00000	-0.95062
19	0.35610	-0.93627	0.91171	-0.28523	-0.79957	-0.99034	-0.97453	0.94542

View dal debugger della medesima matrice, anche qui è facile notare la diagonale di affinità tra l'utente e se stesso.

---

## Determinare i prodotti da suggerire

```
31     i = X.index[8] #id del prodotto con indice n
32     product_names = list(X.index)
33     product_ID = product_names.index(i)
34     correlation_product_ID = correlation_matrix[product_ID]
35     Recommend = list(X.index[correlation_product_ID > 0.5])
36     # Removes the item already bought by the customer
37     Recommend.remove(i)
38     print("Il prodotto {} ha una relazione con: ".format(i))
39     print(Recommend)
40     print(correlation_product_ID)
```

Ipotizzando che l'utente abbia effettuato l'acquisto dell'item con index 8, viene ricercato un riscontro tra tale item e gli altri sulla base delle recensioni di utenti che hanno comprato lo stesso prodotto.

```
C:\Users\xlits\PycharmProjects\ML1\venv\Scripts\python.exe "C:/Users/xlits/PycharmProjects/ML1/moduloFIA/test/Two Layer HS.py"
Il prodotto 10 ha una relazione con:
[8, 9]
[ 0.11234319  0.05780858  0.1016665 -0.82070469 -0.28433023 -0.28433023
  0.97777491  0.71869026  1.         ]
```

Process finished with exit code 0

Facendo riferimento alla table “recensioni” di sopra, si vedono tre recensioni con voto 5 ai prodotti 7, 8, 9 effettuati dallo stesso user. Al modello non interessa riconoscere il perché vi è una correlazione tra questi tre items, è sufficiente che riesca a trovarla e che riesca a restituirla in output.

---

## 5. INTEGRAZIONE CON LA PIATTAFORMA

L'integrazione con la piattaforma è scomponibile in due parti; la prima consiste nella lettura tramite query delle istanze delle recensioni presenti all'interno del DB.

A tale fine ho in un primo momento scritto uno script in Python che si connette al database, recupera le istanze dalla table “recensioni” e le memorizza in un file .csv

Dopodiché ho convertito tale script in un eseguibile .exe sicché possa essere lanciato dal backend della piattaforma.

```
try:
    connection = mysql.connector.connect(host='localhost',
                                         database='helpseller',
                                         user='root',
                                         password='root')

    sql_select_Query = "select * from recensione"
    cursor = connection.cursor()
    cursor.execute(sql_select_Query)
    # get all records
    records = cursor.fetchall()
    print("Total number of rows in table: ", cursor.rowcount)

    print("\nPrinting each row")
    for row in records:
        print("Id = ", row[0], )
        print("testo = ", row[1])
        print("voto = ", row[2])
        print("idProdotto = ", row[4])
        print("idDistributore = ", row[5], "\n")
```

```
except mysql.connector.Error as e:
    print("Error reading data from MySQL table", e)
finally:
    if connection.is_connected():
        connection.close()
        cursor.close()
        print("MySQL connection is closed")

    # Create the csv file
    with open('bigData.csv', 'w', newline='') as f_handle:
        writer = csv.writer(f_handle)
        # Add the header/column names
        header = ['id', 'testo', 'voto', 'data', 'idProdotto', 'idDistributore']
        writer.writerow(header)
        # Iterate over `data` and write to the csv file
        for row in records:
            writer.writerow(row)
```

La seconda parte dell'integrazione consiste nel modello vero e proprio, per funzionare necessita del file .csv appena generato, prende in input un prodotto e restituisce quelli affini.