# Système de recommandations de livres

Anthony Moisan

February 14, 2020

## Contents

# 1 Description du projet et des jeux de données

L'objectif du projet est de mettre en place un système de recommandation.

Dans le cas présent, nous allons utiliser un dataset sur des livres avec trois tables : dataset.

**BX-Users** Contains the users. Note that user IDs (`User-ID`) have been anonymized and map to integers. Demographic data is provided (`Location`, `Age`) if available. Otherwise, these fields contain NULL-values.

**BX-Books** Books are identified by their respective ISBN. Invalid ISBNs have already been removed from the dataset. Moreover, some content-based information is given (`Book-Title`, `Book-Author`, `Year-Of-Publication`, `Publisher`), obtained from Amazon Web Services. Note that in case of several authors, only the first is provided. URLs linking to cover images are also given, appearing in three different flavours (`Image-URL-S`, `Image-URL-M`, `Image-URL-L`), i.e., small, medium, large. These URLs point to the Amazon web site.

**BX-Book-Ratings** Contains the book rating information. Ratings (`Book-Rating`) are either explicit, expressed on a scale from 1-10 (higher values denoting higher appreciation), or implicit, expressed by 0.

# 2 Chargement des fichiers et quelques statistiques

## 2.1 Lecture des fichiers

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
df_users = pd.read_csv("../input/BX-Users.csv", sep=";",delimiter=";", header=0,␣
 ↪encoding='ansi')
print("taille du jeu de donnees Users :", df_users.shape)
df_users.head(10)
```

```
taille du jeu de donnees Users : (278858, 3)
```

[2]:
```
   User-ID                            Location   Age
0        1                  nyc, new york, usa   NaN
1        2           stockton, california, usa  18.0
2        3     moscow, yukon territory, russia   NaN
3        4           porto, v.n.gaia, portugal  17.0
4        5  farnborough, hants, united kingdom   NaN
5        6       santa monica, california, usa  61.0
6        7                 washington, dc, usa   NaN
7        8            timmins, ontario, canada   NaN
8        9          germantown, tennessee, usa   NaN
9       10          albacete, wisconsin, spain  26.0
```

```
[3]: #certaines données sont mal formées avec des "" dans les champs. Utilisation␣
      ↪d'error_bad_lines
     df_books = pd.read_csv("../input/BX-Books.csv", sep=";",delimiter=";", header=0,␣
      ↪encoding='ansi', error_bad_lines=False)
     print("taille du jeu de donnees Books :", df_books.shape)
     df_books.head(10)
```

```
b'Skipping line 6452: expected 8 fields, saw 9\nSkipping line 43667: expected 8
fields, saw 10\nSkipping line 51751: expected 8 fields, saw 9\n'
b'Skipping line 92038: expected 8 fields, saw 9\nSkipping line 104319: expected
8 fields, saw 9\nSkipping line 121768: expected 8 fields, saw 9\n'
b'Skipping line 144058: expected 8 fields, saw 9\nSkipping line 150789: expected
8 fields, saw 9\nSkipping line 157128: expected 8 fields, saw 9\nSkipping line
180189: expected 8 fields, saw 9\nSkipping line 185738: expected 8 fields, saw
9\n'
b'Skipping line 209388: expected 8 fields, saw 9\nSkipping line 220626: expected
8 fields, saw 9\nSkipping line 227933: expected 8 fields, saw 11\nSkipping line
228957: expected 8 fields, saw 10\nSkipping line 245933: expected 8 fields, saw
9\nSkipping line 251296: expected 8 fields, saw 9\nSkipping line 259941:
expected 8 fields, saw 9\nSkipping line 261529: expected 8 fields, saw 9\n'

taille du jeu de donnees Books : (271360, 8)

C:\Users\bigdata\Anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3058: DtypeWarning: Columns (3) have
mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
[3]:          ISBN                                          Book-Title  \
     0  0195153448                                 Classical Mythology
     1  0002005018                                        Clara Callan
     2  0060973129                                Decision in Normandy
     3  0374157065  Flu: The Story of the Great Influenza Pandemic...
     4  0393045218                              The Mummies of Urumchi
     5  0399135782                              The Kitchen God's Wife
     6  0425176428  What If?: The World's Foremost Military Histor...
     7  0671870432                                     PLEADING GUILTY
     8  0679425608  Under the Black Flag: The Romance and the Real...
     9  074322678X            Where You'll Find Me: And Other Stories

                Book-Author Year-Of-Publication                   Publisher  \
     0   Mark P. O. Morford                2002     Oxford University Press
     1 Richard Bruce Wright                2001       HarperFlamingo Canada
     2         Carlo D'Este                1991             HarperPerennial
     3     Gina Bari Kolata                1999        Farrar Straus Giroux
     4      E. J. W. Barber                1999  W. W. Norton &amp; Company
     5              Amy Tan                1991            Putnam Pub Group
     6        Robert Cowley                2000     Berkley Publishing Group
```

```
7           Scott Turow                1993                Audioworks
8       David Cordingly                1996              Random House
9           Ann Beattie                2002                  Scribner

                                          Image-URL-S  \
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...
5  http://images.amazon.com/images/P/0399135782.0...
6  http://images.amazon.com/images/P/0425176428.0...
7  http://images.amazon.com/images/P/0671870432.0...
8  http://images.amazon.com/images/P/0679425608.0...
9  http://images.amazon.com/images/P/074322678X.0...

                                          Image-URL-M  \
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...
5  http://images.amazon.com/images/P/0399135782.0...
6  http://images.amazon.com/images/P/0425176428.0...
7  http://images.amazon.com/images/P/0671870432.0...
8  http://images.amazon.com/images/P/0679425608.0...
9  http://images.amazon.com/images/P/074322678X.0...

                                          Image-URL-L
0  http://images.amazon.com/images/P/0195153448.0...
1  http://images.amazon.com/images/P/0002005018.0...
2  http://images.amazon.com/images/P/0060973129.0...
3  http://images.amazon.com/images/P/0374157065.0...
4  http://images.amazon.com/images/P/0393045218.0...
5  http://images.amazon.com/images/P/0399135782.0...
6  http://images.amazon.com/images/P/0425176428.0...
7  http://images.amazon.com/images/P/0671870432.0...
8  http://images.amazon.com/images/P/0679425608.0...
9  http://images.amazon.com/images/P/074322678X.0...
```

[4]:

```
df_ratings = pd.read_csv("../input/BX-Book-Ratings.csv", sep=";",delimiter=";",␣
 ↪header=0, encoding='ansi')
print("taille du jeu de donnees Book Ratings :", df_ratings.shape)
df_ratings.head(10)
```

```
taille du jeu de donnees Book Ratings : (1149780, 3)
```

```
[4]:    User-ID        ISBN  Book-Rating
     0   276725  034545104X            0
     1   276726  0155061224            5
     2   276727  0446520802            0
     3   276729  052165615X            3
     4   276729  0521795028            6
     5   276733  2080674722            0
     6   276736  3257224281            8
     7   276737  0600570967            6
     8   276744  038550120X            7
     9   276745   342310538           10
```

## 2.2 Quelques statistiques descriptives

**Statistiques sur les Users**

```
[5]: nb_LinesUsers = df_users.shape[0]
     print("# Nb Lines File Users : ", nb_LinesUsers)
```

```
# Nb Lines File Users :  278858
```

```
[6]: nb_Users = len(df_users['User-ID'].unique())
     print("# Users : ", nb_Users)
```

```
# Users :  278858
```

Tous les users sont uniques.

**Statistiques sur les Livres**

```
[7]: nb_LinesBooks = df_books.shape[0]
     print("# Nb Lines File Books : ", nb_LinesBooks)
```

```
# Nb Lines File Books :  271360
```

```
[8]: nb_Books = len(df_books['ISBN'].unique())
     print("# Books : ", nb_Books)
```

```
# Books :  271360
```

Tous les livres sont uniques.

**Statistiques sur les Rating**

```
[9]: nb_LinesRatings = df_ratings.shape[0]
     print("# Nb Lines File Ratings : ", nb_LinesRatings)
```

```
# Nb Lines File Ratings :  1149780
```

[10]:

```
set_scores = list(set(df_ratings["Book-Rating"]))
set_scores
```

[10]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[11]:

```
# Checking if the user has rated the same book twice, in that case we just take␣
 ↪max of them
df_validationRatings = df_ratings.groupby(['User-ID','ISBN']).aggregate(np.max)
print("Probleme de duplication dans ratings :", len(df_validationRatings) !=␣
 ↪nb_LinesRatings)
```

```
Probleme de duplication dans ratings : False
```

Plus d'un million de notations sur des livres compris entre 0 et 10 par des utilisateurs et il n'y a pas de duplication avec un utilisateur ayant noté deux fois le même livre.

[12]:

```
sparsity = round(1.0 - nb_LinesRatings/(1.0*(nb_Books*nb_Users)),5)
print("Sparsity : ", sparsity)
```

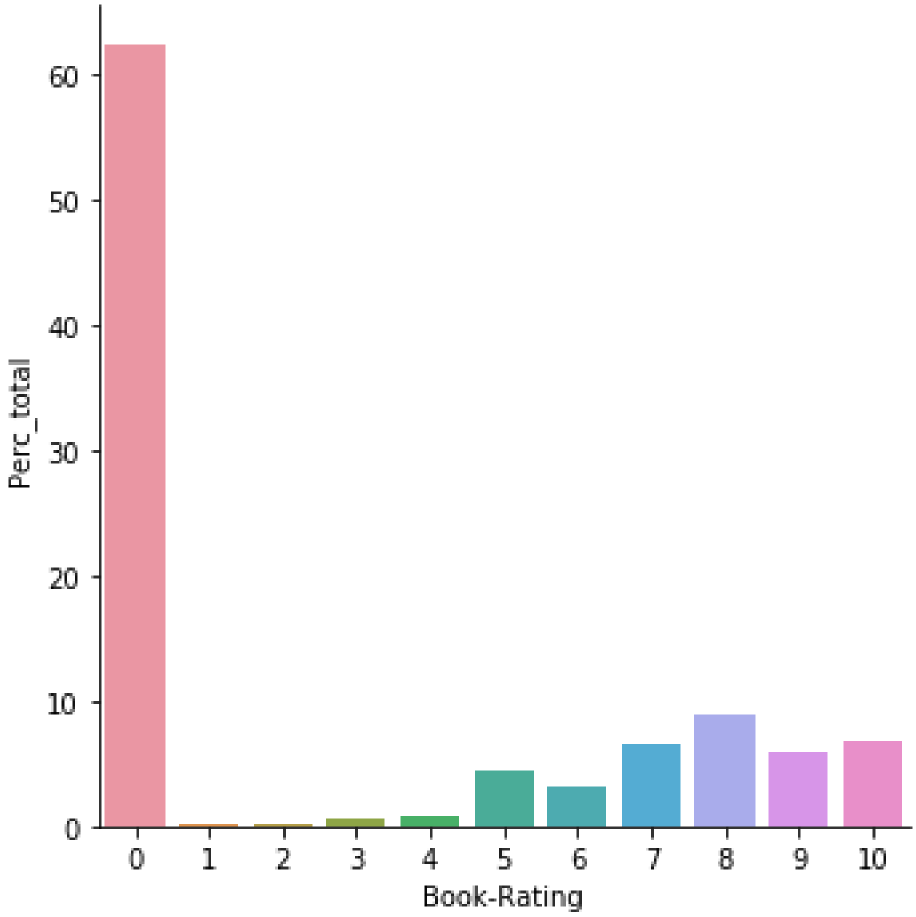
```
Sparsity :  0.99998
```

[13]:

```
count_ratings = df_ratings.groupby('Book-Rating').count()
count_ratings['Book-Rating'] = count_ratings.index
count_ratings['Perc_total']=round(count_ratings['User-ID']*100/
 ↪count_ratings['User-ID'].sum(),1)
sns.catplot(x= "Book-Rating", y="Perc_total",data=count_ratings,kind='bar')
```

[13]:

```
<seaborn.axisgrid.FacetGrid at 0x1bcf70a43c8>
```



Nous avons pu voir simplement : * Le nombre d'utilisateurs, le nombre de livres, la sparsité de la matrice users par livre * La distribution des notes sur les livres : la modalité 0 est la plus représentée mais correspond à une note implicite.

## 3 Construction du Dataset

1. Ne comprenant pas la note 0, on va se passer de l'ensemble des notes implicites ce qui va réduire considérablement le dataset initial de ratings

```
[14]: df_ratingsRed = df_ratings[df_ratings["Book-Rating"]>0]
      len(df_ratingsRed)
```

```
[14]: 433671
```

2. Faisons le merge avec les users présents dans la table User

```
[15]: rating_user = pd.merge(df_ratingsRed,df_users,on='User-ID')
      rating_user.head(5)
```

```
[15]:    User-ID        ISBN  Book-Rating                            Location   Age
      0   276726  0155061224            5           seattle, washington, usa   NaN
      1   276729  052165615X            3               rijeka, n/a, croatia  16.0
      2   276729  0521795028            6               rijeka, n/a, croatia  16.0
      3   276736  3257224281            8        salzburg, salzburg, austria   NaN
      4   276737  0600570967            6  sydney, new south wales, australia  14.0
```

```
[16]: len(rating_user)
```

```
[16]: 433671
```

Pas de perte, on retrouve tous les users.

```
[17]: df = pd.merge(rating_user, df_books, on="ISBN")
```

```
[18]: len(df)
```

```
[18]: 383842
```

Il y a des livres qui ne sont pas présents dans les deux tables.

```
[19]: df.head(5)
```

```
[19]:    User-ID        ISBN  Book-Rating                   Location   Age  \
      0   276726  0155061224            5   seattle, washington, usa   NaN
      1   276729  052165615X            3       rijeka, n/a, croatia  16.0
      2   276729  0521795028            6       rijeka, n/a, croatia  16.0
      3   276744  038550120X            7  torrance, california, usa   NaN
      4    11676  038550120X           10              n/a, n/a, n/a   NaN

                                               Book-Title    Book-Author  \
      0                                  Rites of Passage     Judith Rae
      1                                    Help!: Level 1  Philip Prowse
      2  The Amsterdam Connection : Level 4 (Cambridge ...    Sue Leather
      3                                  A Painted House   JOHN GRISHAM
      4                                  A Painted House   JOHN GRISHAM

        Year-Of-Publication                   Publisher  \
      0                2001                      Heinle
      1                1999  Cambridge University Press
      2                2001  Cambridge University Press
      3                2001                   Doubleday
      4                2001                   Doubleday

                                               Image-URL-S  \
      0  http://images.amazon.com/images/P/0155061224.0...
      1  http://images.amazon.com/images/P/052165615X.0...
      2  http://images.amazon.com/images/P/0521795028.0...
```

```
3  http://images.amazon.com/images/P/038550120X.0...
4  http://images.amazon.com/images/P/038550120X.0...

                                         Image-URL-M  \
0  http://images.amazon.com/images/P/0155061224.0...
1  http://images.amazon.com/images/P/052165615X.0...
2  http://images.amazon.com/images/P/0521795028.0...
3  http://images.amazon.com/images/P/038550120X.0...
4  http://images.amazon.com/images/P/038550120X.0...

                                         Image-URL-L
0  http://images.amazon.com/images/P/0155061224.0...
1  http://images.amazon.com/images/P/052165615X.0...
2  http://images.amazon.com/images/P/0521795028.0...
3  http://images.amazon.com/images/P/038550120X.0...
4  http://images.amazon.com/images/P/038550120X.0...
```

On va recalculer les statistiques sur base des données présentes sur le nombre d'Users, de books, sparsite

[20]:
```
nb_Users = len(df['User-ID'].unique())
print("# Users : ", nb_Users)
```

```
# Users :  68091
```

[21]:
```
nb_Books = len(df['ISBN'].unique())
print("# Books : ", nb_Books)
```

```
# Books :  149836
```

[22]:
```
nb_Ratings = df.shape[0]
print("# Nb Ratings : ", nb_Ratings)
```

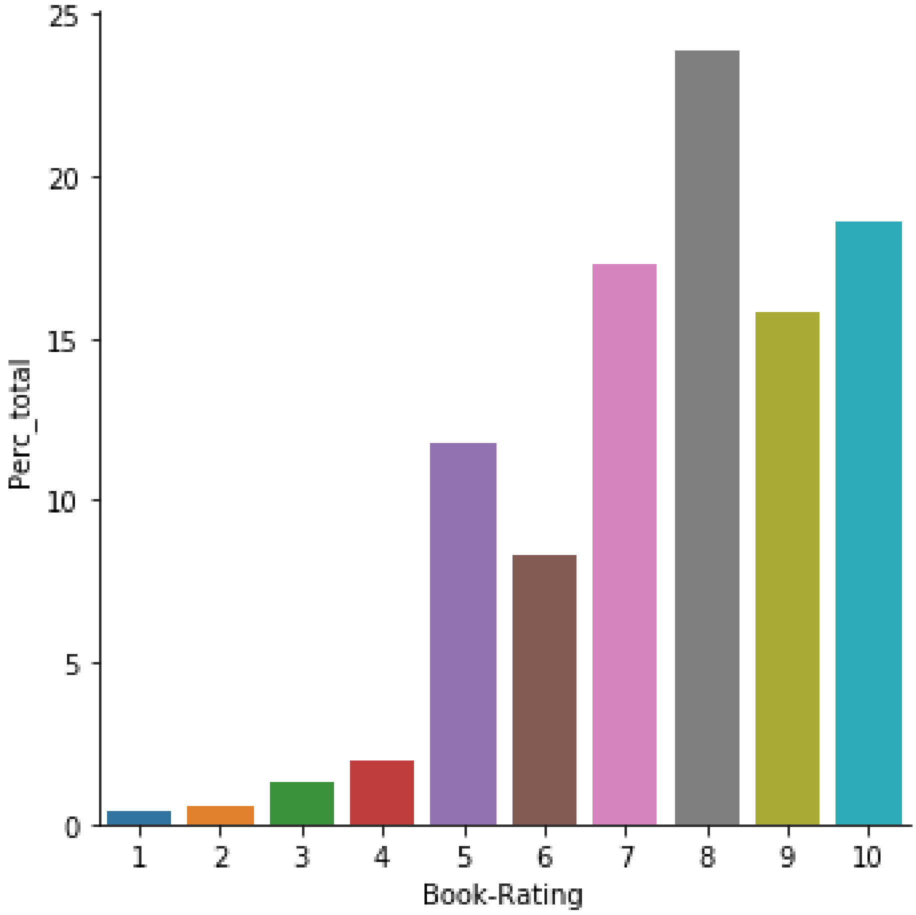
```
# Nb Ratings :  383842
```

[23]:
```
sparsity = round(1.0 - nb_Ratings/(1.0*(nb_Books*nb_Users)),5)
print("Sparsity : ", sparsity)
```

```
Sparsity :  0.99996
```

[24]:
```
count_ratings = df.groupby('Book-Rating').count()
count_ratings['Book-Rating'] = count_ratings.index
count_ratings['Perc_total']=round(count_ratings['User-ID']*100/
 ↪count_ratings['User-ID'].sum(),1)
sns.catplot(x= "Book-Rating", y="Perc_total",data=count_ratings,kind='bar')
```

[24]:
```
<seaborn.axisgrid.FacetGrid at 0x1bc8015ca88>
```



Sans surprise, on retrouve la forme de la distribution en enlevant la notation implicite.

# 4 Recommandations

## 4.1 Recommandations basées sur l'indicateur IMDB

Visualisation du top 10 des livres les plus commentées avec calcul de la moyenne des scores des utilisateurs.

[25]:
```
# Finding the average rating for book and the number of ratings for each book
avg_book_rating = pd.DataFrame(df.groupby('ISBN')['Book-Rating'].
 ↪agg(['mean','count']))
avg_book_rating.sort_values(['count'],ascending=False)[:10]
```

[25]:
```
                mean  count
ISBN
0316666343  8.185290    707
0971880107  4.390706    581
```

```
0385504209  8.435318    487
0312195516  8.182768    383
0060928336  7.887500    320
059035342X  8.939297    313
0142001740  8.452769    307
0446672211  8.142373    295
044023722X  7.338078    281
0452282152  7.982014    278
```

[26]:
```
avg_rating_all = df['Book-Rating'].mean()
print("La moyenne des notes sur l'ensemble des livres ", avg_rating_all)
```

```
La moyenne des notes sur l'ensemble des livres  7.626700569505161
```

[27]:
```
#calculate the percentile count. It gives the no of ratings at least 70% of the␣
↪books have
min_reviews = np.percentile(avg_book_rating['count'],70)
print("Min reviews pour 70% percentile des livres ", min_reviews)
```

```
Min reviews pour 70% percentile des livres  2.0
```

Dans notre DataSet, il y a beaucoup de livres qui ont très peu de notations.

[28]:
```
#Pondération entre le rating du book et le rating de tous les books et je tire␣
↪vers C (la moyenne des scores) si pas assez de notations et sinon vers R (la␣
↪moyenne des scores des utilisateurs pour ce livre)
def weighted_rating(x, m=min_reviews, C=avg_rating_all):
    v = x['count']
    R = x['mean']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

[29]:
```
#On réduit le dataset initial par rapport au nombre minimum de reviews␣
↪correspondant au quantile 70 dans le cas présent
book_score = avg_book_rating[avg_book_rating['count']>min_reviews]
#On applique le score IMDB
book_score['Score IMDB'] = book_score.apply(weighted_rating, axis=1)
book_score.head()
```

```
C:\Users\bigdata\Anaconda3\lib\site-packages\ipykernel_launcher.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

```
[29]:                 mean  count  Score IMDB
      ISBN
      0002005018  7.666667      9    7.659400
      0002116286  7.250000      4    7.375567
      0002239183  7.333333      3    7.450680
      0002240114  6.750000      4    7.042234
      0002243962  5.750000      4    6.375567
```

```
[30]: #Petite bidouille pour la jointure
      book_score["ISBN"] = book_score.index
      book_score.index.name = None
      #On garde uniquement les champs qui nous intéressent
      df_books_util = df_books.drop(['Book-Author', 'Publisher', 'Image-URL-S',␣
       ↪'Image-URL-M', 'Image-URL-L'],axis=1)
      book_score = pd.merge(book_score,df_books_util,on='ISBN')
      book_score.head(5)
```

```
C:\Users\bigdata\Anaconda3\lib\site-packages\ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
[30]:       mean  count  Score IMDB        ISBN  \
      0  7.666667      9    7.659400  0002005018
      1  7.250000      4    7.375567  0002116286
      2  7.333333      3    7.450680  0002239183
      3  6.750000      4    7.042234  0002240114
      4  5.750000      4    6.375567  0002243962

                                               Book-Title Year-Of-Publication
      0                                       Clara Callan                2001
      1                   There's A Seal in my Sleeping Bag                1992
      2                                         Affliction                   0
      3  The Dixon Cornbelt League and other baseball s...                1993
      4                                Girlfriend In a Coma                   0
```

```
[31]: # Gives the best books according to year based on weighted score which is␣
       ↪calculated using IMDB formula
      def best_books_by_year(year,top_n):
          return pd.DataFrame(book_score.
       ↪loc[(book_score["Year-Of-Publication"]==year)].sort_values(['Score␣
       ↪IMDB'],ascending=False)[['Book-Title','count','mean','Score IMDB']][:top_n])
```

[32]:

```
best_books_by_year("1999",10)
```

[32]:

```
                                              Book-Title  count       mean  \
8746         The Annotated Alice: The Definitive Edition      4  10.000000
25010       Buffy the Vampire Slayer: Remaining Sunlight      3   9.333333
25842                                Farmhouse Christmas      3   9.000000
27259  Die 13 1/2 Leben des KÃ¤pt'n BlaubÃ¤r: Die hal...      3   8.666667
20096                                      Mutts Sundays      3   8.666667
23253  Working from Home: Everything You Need to Know...      3   8.666667
18831  Running to the Mountain: A Journey of Faith an...      3   8.666667
9978                                           Lindbergh      3   8.333333
8098   A Woman Like That: Lesbian and Bisexual Writer...      3   8.000000
23436  Always Believe in Yourself and Your Dreams: A ...      3   8.000000

       Score IMDB
8746      9.20890
25010     8.65068
25842     8.45068
27259     8.25068
20096     8.25068
23253     8.25068
18831     8.25068
9978      8.05068
8098      7.85068
23436     7.85068
```

Dans le cas présent, il aurait été plus intéressant d'avoir les genres du livre mais nous n'avons pas beaucoup d'informations dans la description des livres. Le filtre a été établi sur la date de publication l'ouvrage qui n'est pas spécialement un élément pertinent dans un système de recommandation.

### 4.2 Recommandations basées sur la similarité entre les livres

On utilise uniquement le rating et la matrice entre les utilisateurs et les livres pour mesurer une similarité entre les livres basées sur l'algorithme des K plus proches voisins et d'une métrique Cosinus.

[33]:

```
#only include books with more than 15 ratings
books_plus_15_ratings = avg_book_rating.loc[avg_book_rating['count']>=15]
print("Nombre de livres ayant plus de 15 ratings :", len(books_plus_15_ratings))
```

```
Nombre de livres ayant plus de 15 ratings : 3151
```

[34]:

```
books_plus_15_ratings.head(5)
```

[34]:

```
                mean  count
ISBN
000649840X  7.756098     41
```

```
0006547834  8.066667     15
0006550576  7.266667     15
0007110928  7.703704     27
0020198817  7.933333     15
```

[35]:
```
books_plus_15_ratings["ISBN"] = books_plus_15_ratings.index
books_plus_15_ratings.index.name=None
filtered_ratings = pd.merge(books_plus_15_ratings, df, on="ISBN")
print("Nombre d'enregistrements avec uniquement les livres ayant plus de 15␣
 ↪commentaires :", len(filtered_ratings))
```

```
C:\Users\bigdata\Anaconda3\lib\site-packages\ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

```
Nombre d'enregistrements avec uniquement les livres ayant plus de 15
commentaires : 112631
```

[36]:
```
filtered_ratings.head()
```

[36]:
```
       mean  count        ISBN  User-ID  Book-Rating  \
0  7.756098     41  000649840X      901            9
1  7.756098     41  000649840X    11676            8
2  7.756098     41  000649840X    16383           10
3  7.756098     41  000649840X    24922            9
4  7.756098     41  000649840X    32440            8

                                   Location   Age     Book-Title  \
0  worcester, worcestershire, united kingdom  21.0  Angelas Ashes
1                             n/a, n/a, n/a   NaN  Angelas Ashes
2         sheffield, england, united kingdom  24.0  Angelas Ashes
3      wolverhampton, england, united kingdom  45.0  Angelas Ashes
4                 dunedin, otago, new zealand   NaN  Angelas Ashes

     Book-Author Year-Of-Publication         Publisher  \
0  Frank Mccourt                   0  Harpercollins Uk
1  Frank Mccourt                   0  Harpercollins Uk
2  Frank Mccourt                   0  Harpercollins Uk
3  Frank Mccourt                   0  Harpercollins Uk
4  Frank Mccourt                   0  Harpercollins Uk

                                         Image-URL-S  \
0  http://images.amazon.com/images/P/000649840X.0...
```

```
1  http://images.amazon.com/images/P/000649840X.0...
2  http://images.amazon.com/images/P/000649840X.0...
3  http://images.amazon.com/images/P/000649840X.0...
4  http://images.amazon.com/images/P/000649840X.0...

                                         Image-URL-M  \
0  http://images.amazon.com/images/P/000649840X.0...
1  http://images.amazon.com/images/P/000649840X.0...
2  http://images.amazon.com/images/P/000649840X.0...
3  http://images.amazon.com/images/P/000649840X.0...
4  http://images.amazon.com/images/P/000649840X.0...

                                         Image-URL-L
0  http://images.amazon.com/images/P/000649840X.0...
1  http://images.amazon.com/images/P/000649840X.0...
2  http://images.amazon.com/images/P/000649840X.0...
3  http://images.amazon.com/images/P/000649840X.0...
4  http://images.amazon.com/images/P/000649840X.0...
```

[37]:

```
print("# Users :", len(filtered_ratings["User-ID"].unique()))
```

```
# Users : 34573
```

[38]:

```
print("# Books :", len(filtered_ratings["ISBN"].unique()))
```

```
# Books : 3151
```

[39]:

```
#create a matrix table with ISBN on the rows and User-ID in the columns.
#replace NAN values with 0
matrix_books_user = filtered_ratings.pivot(index = 'ISBN', columns = 'User-ID',␣
 ↪values = 'Book-Rating').fillna(0)
matrix_books_user.head()
```

[39]:

```
User-ID     9       16      17      26      32      39      42      44      \
ISBN
000649840X     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0006547834     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0006550576     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0007110928     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0020198817     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0

User-ID     51      56      ...  278800  278807  278828  278832  278836  \
ISBN                        ...
000649840X     0.0     0.0  ...     0.0     0.0     0.0     0.0     0.0
0006547834     0.0     0.0  ...     0.0     0.0     0.0     0.0     0.0
0006550576     0.0     0.0  ...     0.0     0.0     0.0     0.0     0.0
0007110928     0.0     0.0  ...     0.0     0.0     0.0     0.0     0.0
```

```
0020198817     0.0     0.0  ...     0.0     0.0     0.0     0.0     0.0

User-ID     278843  278844  278846  278851  278854
ISBN
000649840X     0.0     0.0     0.0     0.0     0.0
0006547834     0.0     0.0     0.0     0.0     0.0
0006550576     0.0     0.0     0.0     0.0     0.0
0007110928     0.0     0.0     0.0     0.0     0.0
0020198817     0.0     0.0     0.0     0.0     0.0

[5 rows x 34573 columns]
```

On va utiliser les K plus proches voisins avec la similarité cosinus comme mesure de ressemblance entre deux livres.

[40]:

```python
from sklearn.neighbors import NearestNeighbors
#specify model parameters
model_knn = NearestNeighbors(metric='cosine',algorithm='brute')
#fit model to the data set
model_knn.fit(matrix_books_user)
```

[40]:

```
NearestNeighbors(algorithm='brute', leaf_size=30, metric='cosine',
                 metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                 radius=1.0)
```

[41]:

```python
#Gets the top 10 nearest neighbours got the book
def print_similar_books(query_index) :
    #get the list of user ratings for a specific userId
    query_index_books_ratings = matrix_books_user.loc[query_index,:].values.
 ↪reshape(1,-1)
    #get the closest 10 books and their distances from the book specified
    distances,indices = model_knn.
 ↪kneighbors(query_index_books_ratings,n_neighbors = 11)
    #write a lopp that prints the similar books for a specified book.
    for i in range(0,len(distances.flatten())):
        #get the title of the random book that was chosen
        get_book = df_books.loc[df_books['ISBN']==query_index]['Book-Title']
        #for the first movie in the list i.e closest print the title
        if i==0:
            print('Recommendations for {0}:\n'.format(get_book))
        else :
            #get the indiciees for the closest movies
            indices_flat = indices.flatten()[i]
            #get the title of the movie
            get_book = df_books.loc[df_books['ISBN']==matrix_books_user.
 ↪iloc[indices_flat,:].name]['Book-Title']
            #print the movie
```

```
            print('{0}: {1}, with distance of {2}:'.format(i,get_book,distances.
↪flatten()[i]))
```

On va chercher les livres similaires par rapport au livre le plus noté dans le dataset qui correspond au numéro ISBN 0316666343

```
[42]: print_similar_books("0316666343")
```

```
Recommendations for 408    The Lovely Bones: A Novel
Name: Book-Title, dtype: object:

1: 3939    Lucky : A Memoir
Name: Book-Title, dtype: object, with distance of 0.8907765344442606:
2: 706    Where the Heart Is (Oprah's Book Club (Paperba...
Name: Book-Title, dtype: object, with distance of 0.9095981950904724:
3: 748    The Da Vinci Code
Name: Book-Title, dtype: object, with distance of 0.9170468045457768:
4: 3511    Lucky
Name: Book-Title, dtype: object, with distance of 0.9185733092426993:
5: 776    Nights in Rodanthe
Name: Book-Title, dtype: object, with distance of 0.922072755930089:
6: 1496    Good in Bed
Name: Book-Title, dtype: object, with distance of 0.9227391845889488:
7: 12920    The Pact: A Love Story
Name: Book-Title, dtype: object, with distance of 0.9244091289737372:
8: 6038    The Lake House
Name: Book-Title, dtype: object, with distance of 0.9253117176778205:
9: 2536    The Book of Ruth (Oprah's Book Club (Paperback))
Name: Book-Title, dtype: object, with distance of 0.9286916905396203:
10: 4824    The Pilot's Wife : A Novel
Name: Book-Title, dtype: object, with distance of 0.9323735033791939:
```

A noter que cette méthode n'a pas pu être mise en oeuvre sur le dataset df liée à l'impossibilité de construire la matrice books_user. La réduction en prenant uniquement des livres notés a minima 15 fois a permis de réduire la dimension du nombre d'utilisateurs et de livres permettant son initialisation.

## 4.3 Recommandations collababoratives basées sur la factorisation de matrices

Un système de recommadation populaire est basé sur la factorisation de matrice. Elle est basée sur le principe de représentation dans une moindre dimension des livres et des utilisateurs (embedding). Etant donné une matrice (A [M X N]) avec M le nombre d'utilisateurs et N le nombre de livres, nous souhaitons estimer en moindre dimension (W [M X k] et H [M X k]), tel que: AW.HT

### 4.3.1 SVD et sparsité de la matrice

Pour des problèmes de mémoire, on va aussi travailler avec le dataset où les livres sont notés a minima 15 fois.

```
[43]: df_red = filtered_ratings.drop(['mean', 'count','Location', 'Age', 'Book-Title',
             'Book-Author', 'Year-Of-Publication', 'Publisher', 'Image-URL-S',
             'Image-URL-M', 'Image-URL-L'], axis=1)
```

```
[44]: df_red.head(5)
```

```
[44]:          ISBN  User-ID  Book-Rating
      0  000649840X      901            9
      1  000649840X    11676            8
      2  000649840X    16383           10
      3  000649840X    24922            9
      4  000649840X    32440            8
```

On va créer des index en lieu et place des User-ID et ISBN

```
[45]: #get ordered list of userIds
      user_indices = pd.
       ↪DataFrame(sorted(list(set(df_red['User-ID']))),columns=['User-ID'])
      #add in data frame index value to data frame
      user_indices['User_index']=user_indices.index
      #inspect data frame
      user_indices.head()
```

```
[45]:    User-ID  User_index
      0        9           0
      1       16           1
      2       17           2
      3       26           3
      4       32           4
```

```
[46]: #get ordered list of ISBN
      isbn_indices = pd.DataFrame(sorted(list(set(df_red['ISBN']))),columns=['ISBN'])
      #add in data frame index value to data frame
      isbn_indices['ISBN_index']=isbn_indices.index
      #inspect data frame
      isbn_indices.head()
```

```
[46]:          ISBN  ISBN_index
      0  000649840X           0
      1  0006547834           1
      2  0006550576           2
      3  0007110928           3
      4  0020198817           4
```

```
[47]: #join the ISBN indices
      df_with_index = pd.merge(df_red,isbn_indices,on='ISBN')
      #join the user indices
```

```
df_with_index=pd.merge(df_with_index,user_indices,on='User-ID')
#inspec the data frame
df_with_index.head()
```

[47]:

```
         ISBN  User-ID  Book-Rating  ISBN_index  User_index
0  000649840X      901            9           0         108
1  0099771519      901            9         242         108
2  000649840X    11676            8           0        1362
3  0006547834    11676            5           1        1362
4  0007110928    11676            7           3        1362
```

Pour le modèle, on va séparer notre dataset avec 80% pour fitter le modèle et 20% pour tester

[48]:

```
#import train_test_split module
from sklearn.model_selection import train_test_split
#take 80% as the training set and 20% as the test set
df_train, df_test= train_test_split(df_with_index,test_size=0.2)
print(len(df_train))
print(len(df_test))
```

```
90104
22527
```

[49]:

```
n_users = len(df_with_index["User-ID"].unique())
print("# users :", n_users)
n_books = len(df_with_index["ISBN"].unique())
print("# books :", n_books)
```

```
# users : 34573
# books : 3151
```

[50]:

```
df_train.head(5)
```

[50]:

```
             ISBN  User-ID  Book-Rating  ISBN_index  User_index
4072   0452269571   202963            8        1856       25029
79117  0380792745    65183            8         897        8245
95367  0553572377   262271           10        2184       32575
58291  0142001740   136234           10         329       17031
55180  0446670251    64429            7        1649        8144
```

On crée la matrice d'entrainement et de tests n_user par n_books basés sur les index avec comme valeur le rating

[51]:

```
#Create one user-book matrices for training
train_data_matrix = np.zeros((n_users, n_books))
    #for every line in the data
for line in df_train.itertuples():
    #set the value in the column and row to
```

```
    #line[1] is User-Id, line[2] is ISBN and line[3] is Book-Rating, line[4] is␣
  ↪ISBN_index and line[5] is User_index
    train_data_matrix[line[5], line[4]] = line[3]
train_data_matrix.shape
```

[51]: (34573, 3151)

[52]:

```
#Create one user-book matrices for testing
test_data_matrix = np.zeros((n_users, n_books))
    #for every line in the data
for line in df_test[:1].itertuples():
    #set the value in the column and row to
    #line[1] is User-Id, line[2] is ISBN and line[3] is Book-Rating, line[4] is␣
  ↪ISBN_index and line[5] is User_index
    test_data_matrix[line[5], line[4]] = line[3]
test_data_matrix.shape
```

[52]: (34573, 3151)

La métrique utilisée sera la Mean Squared Error entre la prédiction et la vraie valeur

[53]:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
def rmse(prediction, ground_truth):
    #select prediction values that are non-zero and flatten into 1 array
    prediction = prediction[ground_truth.nonzero()]
    #select test values that are non-zero and flatten into 1 array
    ground_truth = ground_truth[ground_truth.nonzero()]
    #return RMSE between values
    return sqrt(mean_squared_error(prediction, ground_truth))
```

On réalise une décomposition de la matrice suivant k valeurs singulières en fittant sur le train et en mesurant l'erreur sur le test. Cette décomposition utilise une méthode se basant sur le caractère très sparse des matrices.

[54]:

```
from scipy.sparse.linalg import svds
#Calculate the rmse sscore of SVD using different values of k (latent features)
rmse_list = []
for i in [1,2,5,20,40,60,100]:
    #apply svd to the test data
    u,s,vt = svds(train_data_matrix,k=i)
    #get diagonal matrix
    s_diag_matrix=np.diag(s)
    #predict x with dot product of u s_diag and vt
    X_pred = np.dot(np.dot(u,s_diag_matrix),vt)
    #calculate rmse score of matrix factorisation predictions
    rmse_score = rmse(X_pred,test_data_matrix)
```

```
    rmse_list.append(rmse_score)
    print("Matrix Factorisation with " + str(i) +" latent features has a RMSE of␣
 ↪" + str(rmse_score))
```

```
Matrix Factorisation with 1 latent features has a RMSE of 8.99950139596305
Matrix Factorisation with 2 latent features has a RMSE of 8.99933081960599
Matrix Factorisation with 5 latent features has a RMSE of 8.998673822240121
Matrix Factorisation with 20 latent features has a RMSE of 8.99542708130528
Matrix Factorisation with 40 latent features has a RMSE of 8.994186340373284
Matrix Factorisation with 60 latent features has a RMSE of 8.991454106696285
Matrix Factorisation with 100 latent features has a RMSE of 8.982204948143098
```

[55]:
```
#Convert predictions to a DataFrame
mf_pred = pd.DataFrame(X_pred)
mf_pred.head()
```

[55]:
```
          0         1         2         3         4         5         6     \
0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1 -0.000665 -0.002979 -0.000914 -0.001585 -0.002521 -0.000441  0.024705
2 -0.001473 -0.000294  0.000097 -0.005037 -0.000678  0.004825  0.008514
3 -0.004279 -0.001372  0.003622  0.063260  0.108750 -0.037414  0.010315
4  0.001148 -0.000421 -0.000266 -0.002045 -0.000751  0.001440  0.004110

          7         8         9     ...      3141      3142      3143      3144  \
0  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000
1  0.011418  0.028709 -0.019144  ... -0.000384 -0.000612 -0.001324 -0.000641
2 -0.005740  0.011503  0.004819  ... -0.001043 -0.000263 -0.000944 -0.000110
3 -0.032633  0.009358 -0.108584  ... -0.003453 -0.002397 -0.010451 -0.003640
4  0.001547  0.006475  0.002359  ... -0.000487  0.000326 -0.000262  0.000115

       3145      3146      3147      3148      3149          3150
0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000e+00
1 -0.000505 -0.000448  0.000005 -0.000157 -0.000198  5.752579e-18
2 -0.001610 -0.001433 -0.000003 -0.000140  0.000021 -1.742706e-18
3 -0.005333 -0.004772 -0.000009  0.000010 -0.001209  2.721052e-16
4 -0.000752 -0.000669 -0.000002 -0.000143  0.000025 -1.851055e-18

[5 rows x 3151 columns]
```

[56]:
```
user_id = 32440
user_index = user_indices.loc[user_indices["User-ID"]==user_id]['User_index'][:
 ↪1].values[0]
#get book ratings predicted for this user and sort by highest rating prediction
sorted_user_predictions = pd.DataFrame(mf_pred.iloc[user_index].
 ↪sort_values(ascending=False))
#rename the columns
sorted_user_predictions.columns=['Book-Rating']
```

```
#save the index values as ISBN
sorted_user_predictions['ISBN_index']=sorted_user_predictions.index
print("Top 10 predictions for User " + str(user_id))
#display the top 10 predictions for this user
top_10 = pd.merge(sorted_user_predictions,isbn_indices, on = 'ISBN_index')[:10]
pd.merge(top_10, df_books, on = 'ISBN')[["Book-Title", "Book-Author",␣
 ↪"Book-Rating"]]
```

```
Top 10 predictions for User 32440
```

[56]:

```
                                          Book-Title         Book-Author  \
0                                         Life of Pi         Yann Martel
1                            The Secret Life of Bees       Sue Monk Kidd
2                          Girl with a Pearl Earring     Tracy Chevalier
3                                      ANGELA'S ASHES       Frank McCourt
4  A Walk in the Woods: Rediscovering America on ...         Bill Bryson
5                                Message in a Bottle     Nicholas Sparks
6                                 STONES FROM THE RIVER         Ursula Hegi
7                                        The Chamber        John Grisham
8                        The Handmaid's Tale : A Novel     Margaret Atwood
9                                The Poisonwood Bible  Barbara Kingsolver

   Book-Rating
0    10.196463
1     9.331762
2     8.919113
3     7.642815
4     1.775618
5     1.748854
6     1.528020
7     1.159126
8     1.094987
9     1.047042
```

Aux vues des résultats, il y a certainement un problème de méthodologie aux vues des Book-rating retournés

### 4.3.2 Deep Learning avec contrainte de non-négativité

[57]:

```
import keras
from keras.layers import Embedding, Reshape  #Merge
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.constraints import non_neg
```

```
Using TensorFlow backend.
```

[58]:

```
# Returns a neural network model which performs matrix factorisation with␣
 ↪additional constraint on embeddings(that they can't be negative)
def␣
 ↪matrix_factorisation_model_with_n_latent_factors_and_non_negative_embedding(n_latent_factors)
 ↪:
    book_input = keras.layers.Input(shape=[1],name='Book')
    book_embedding = keras.layers.Embedding(n_books + 1, n_latent_factors,␣
 ↪name='Non-Negative-Book-Embedding',embeddings_constraint=non_neg())(book_input)
    book_vec = keras.layers.Flatten(name='FlattenBooks')(book_embedding)

    user_input = keras.layers.Input(shape=[1],name='User')
    user_embedding = keras.layers.Embedding(n_users + 1,␣
 ↪n_latent_factors,name='Non-Negative-User-Embedding',embeddings_constraint=non_neg())(user_inp
    user_vec = keras.layers.Flatten(name='FlattenUsers')(user_embedding)

    prod = keras.layers.dot([book_vec, user_vec],axes=1)


    model = keras.Model([user_input, book_input], prod)
    model.compile('adam', 'mean_squared_error')

    return model
```

[59]:

```
model =␣
 ↪matrix_factorisation_model_with_n_latent_factors_and_non_negative_embedding(5)
```

```
WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:517: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:4138: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:74: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\optimizers.py:790: The name tf.train.Optimizer is deprecated.
Please use tf.compat.v1.train.Optimizer instead.
```

[60]:

```
model.summary()
```

```
--------------------------------------------------------------------------------
------------------
```

```
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
Book (InputLayer)               (None, 1)            0

--------------------------------------------------------------------------------
------------------
User (InputLayer)               (None, 1)            0

--------------------------------------------------------------------------------
------------------
Non-Negative-Book-Embedding (Em (None, 1, 5)         15760       Book[0][0]

--------------------------------------------------------------------------------
------------------
Non-Negative-User-Embedding (Em (None, 1, 5)         172870      User[0][0]

--------------------------------------------------------------------------------
------------------
FlattenBooks (Flatten)          (None, 5)            0           Non-Negative-
Book-Embedding[0][0]

--------------------------------------------------------------------------------
------------------
FlattenUsers (Flatten)          (None, 5)            0           Non-Negative-
User-Embedding[0][0]

--------------------------------------------------------------------------------
------------------
dot_1 (Dot)                     (None, 1)            0
FlattenBooks[0][0]
FlattenUsers[0][0]
================================================================================
==================
Total params: 188,630
Trainable params: 188,630
Non-trainable params: 0

--------------------------------------------------------------------------------
------------------
```

[61]:

```
history_nonneg = model.fit([df_train["User_index"], df_train["ISBN_index"]],␣
↪df_train["Book-Rating"], epochs=100, verbose=1)
```

```
WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:986: The name tf.assign_add is
deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:973: The name tf.assign is
deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:2741: The name tf.Session is
deprecated. Please use tf.compat.v1.Session instead.
```

```
Epoch 1/100
WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:174: The name
tf.get_default_session is deprecated. Please use
tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:181: The name tf.ConfigProto is
deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:190: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:199: The name
tf.is_variable_initialized is deprecated. Please use
tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From C:\Users\bigdata\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:206: The name
tf.variables_initializer is deprecated. Please use
tf.compat.v1.variables_initializer instead.

90104/90104 [==============================] - 7s 81us/step - loss: 62.0140
Epoch 2/100
90104/90104 [==============================] - 8s 84us/step - loss: 54.3369
Epoch 3/100
90104/90104 [==============================] - 8s 87us/step - loss: 43.9714
Epoch 4/100
90104/90104 [==============================] - 8s 92us/step - loss: 34.6128
Epoch 5/100
90104/90104 [==============================] - 9s 95us/step - loss: 27.3761
Epoch 6/100
90104/90104 [==============================] - 8s 86us/step - loss: 21.9239
Epoch 7/100
90104/90104 [==============================] - 7s 81us/step - loss: 17.7625
Epoch 8/100
90104/90104 [==============================] - 7s 80us/step - loss: 14.5462
Epoch 9/100
90104/90104 [==============================] - 7s 80us/step - loss: 12.0314
Epoch 10/100
90104/90104 [==============================] - 8s 88us/step - loss: 10.0348
Epoch 11/100
90104/90104 [==============================] - 8s 86us/step - loss: 8.4274
Epoch 12/100
90104/90104 [==============================] - 6s 67us/step - loss: 7.1186
```

```
Epoch 13/100
90104/90104 [==============================] - 6s 69us/step - loss: 6.0446
Epoch 14/100
90104/90104 [==============================] - 6s 68us/step - loss: 5.1587
Epoch 15/100
90104/90104 [==============================] - 6s 70us/step - loss: 4.4243
Epoch 16/100
90104/90104 [==============================] - 7s 81us/step - loss: 3.8200
Epoch 17/100
90104/90104 [==============================] - 7s 79us/step - loss: 3.3164
Epoch 18/100
90104/90104 [==============================] - 7s 78us/step - loss: 2.8995
Epoch 19/100
90104/90104 [==============================] - 7s 78us/step - loss: 2.5572
Epoch 20/100
90104/90104 [==============================] - 7s 78us/step - loss: 2.2823
Epoch 21/100
90104/90104 [==============================] - 7s 79us/step - loss: 2.0657
Epoch 22/100
90104/90104 [==============================] - 7s 79us/step - loss: 1.9024
Epoch 23/100
90104/90104 [==============================] - 7s 79us/step - loss: 1.7843
Epoch 24/100
90104/90104 [==============================] - 7s 79us/step - loss: 1.7001
Epoch 25/100
90104/90104 [==============================] - 7s 79us/step - loss: 1.6422
Epoch 26/100
90104/90104 [==============================] - 7s 83us/step - loss: 1.6034
Epoch 27/100
90104/90104 [==============================] - 8s 91us/step - loss: 1.5786
Epoch 28/100
90104/90104 [==============================] - 8s 88us/step - loss: 1.5635
Epoch 29/100
90104/90104 [==============================] - 8s 87us/step - loss: 1.5540
Epoch 30/100
90104/90104 [==============================] - 8s 91us/step - loss: 1.5497
Epoch 31/100
90104/90104 [==============================] - 8s 90us/step - loss: 1.5458
Epoch 32/100
90104/90104 [==============================] - 9s 104us/step - loss: 1.5437
Epoch 33/100
90104/90104 [==============================] - 9s 95us/step - loss: 1.5427
Epoch 34/100
90104/90104 [==============================] - 8s 84us/step - loss: 1.5409
Epoch 35/100
90104/90104 [==============================] - 8s 84us/step - loss: 1.5401
Epoch 36/100
90104/90104 [==============================] - 8s 84us/step - loss: 1.5391
```

```
Epoch 37/100
90104/90104 [==============================] - 7s 82us/step - loss: 1.5386
Epoch 38/100
90104/90104 [==============================] - 8s 84us/step - loss: 1.5372
Epoch 39/100
90104/90104 [==============================] - 7s 82us/step - loss: 1.5366
Epoch 40/100
90104/90104 [==============================] - 7s 81us/step - loss: 1.5358
Epoch 41/100
90104/90104 [==============================] - 7s 81us/step - loss: 1.5351
Epoch 42/100
90104/90104 [==============================] - 7s 80us/step - loss: 1.5352
Epoch 43/100
90104/90104 [==============================] - 7s 79us/step - loss: 1.5313
Epoch 44/100
90104/90104 [==============================] - 7s 79us/step - loss: 1.5314
Epoch 45/100
90104/90104 [==============================] - 8s 87us/step - loss: 1.5302
Epoch 46/100
90104/90104 [==============================] - 9s 102us/step - loss: 1.5289
Epoch 47/100
90104/90104 [==============================] - 8s 85us/step - loss: 1.5282
Epoch 48/100
90104/90104 [==============================] - 6s 72us/step - loss: 1.5261
Epoch 49/100
90104/90104 [==============================] - 7s 75us/step - loss: 1.5238
Epoch 50/100
90104/90104 [==============================] - 6s 70us/step - loss: 1.5223
Epoch 51/100
90104/90104 [==============================] - 6s 64us/step - loss: 1.5201
Epoch 52/100
90104/90104 [==============================] - 6s 65us/step - loss: 1.5174
Epoch 53/100
90104/90104 [==============================] - 6s 65us/step - loss: 1.5152
Epoch 54/100
90104/90104 [==============================] - 6s 69us/step - loss: 1.5119
Epoch 55/100
90104/90104 [==============================] - 6s 65us/step - loss: 1.5088
Epoch 56/100
90104/90104 [==============================] - 6s 66us/step - loss: 1.5050
Epoch 57/100
90104/90104 [==============================] - 6s 65us/step - loss: 1.5015
Epoch 58/100
90104/90104 [==============================] - 6s 64us/step - loss: 1.4961
Epoch 59/100
90104/90104 [==============================] - 6s 65us/step - loss: 1.4921
Epoch 60/100
90104/90104 [==============================] - 6s 66us/step - loss: 1.4861
```

```
Epoch 61/100
90104/90104 [==============================] - 7s 75us/step - loss: 1.4779
Epoch 62/100
90104/90104 [==============================] - 6s 71us/step - loss: 1.4725
Epoch 63/100
90104/90104 [==============================] - 6s 69us/step - loss: 1.4635
Epoch 64/100
90104/90104 [==============================] - 6s 69us/step - loss: 1.4574
Epoch 65/100
90104/90104 [==============================] - 6s 69us/step - loss: 1.4474
Epoch 66/100
90104/90104 [==============================] - 6s 69us/step - loss: 1.4401
Epoch 67/100
90104/90104 [==============================] - 6s 71us/step - loss: 1.4305
Epoch 68/100
90104/90104 [==============================] - 8s 87us/step - loss: 1.4199
Epoch 69/100
90104/90104 [==============================] - 8s 90us/step - loss: 1.4103
Epoch 70/100
90104/90104 [==============================] - 8s 85us/step - loss: 1.3991
Epoch 71/100
90104/90104 [==============================] - 8s 88us/step - loss: 1.3892
Epoch 72/100
90104/90104 [==============================] - 7s 75us/step - loss: 1.3765
Epoch 73/100
90104/90104 [==============================] - 7s 78us/step - loss: 1.3640
Epoch 74/100
90104/90104 [==============================] - 7s 75us/step - loss: 1.3508
Epoch 75/100
90104/90104 [==============================] - 6s 63us/step - loss: 1.3389
Epoch 76/100
90104/90104 [==============================] - 6s 63us/step - loss: 1.3250
Epoch 77/100
90104/90104 [==============================] - 6s 64us/step - loss: 1.3111
Epoch 78/100
90104/90104 [==============================] - 6s 64us/step - loss: 1.2973
Epoch 79/100
90104/90104 [==============================] - 6s 63us/step - loss: 1.2817
Epoch 80/100
90104/90104 [==============================] - 6s 63us/step - loss: 1.2675
Epoch 81/100
90104/90104 [==============================] - 6s 63us/step - loss: 1.2519
Epoch 82/100
90104/90104 [==============================] - 6s 70us/step - loss: 1.2363
Epoch 83/100
90104/90104 [==============================] - 6s 65us/step - loss: 1.2215
Epoch 84/100
90104/90104 [==============================] - 6s 66us/step - loss: 1.2045
```

```
Epoch 85/100
90104/90104 [==============================] - 7s 72us/step - loss: 1.1899
Epoch 86/100
90104/90104 [==============================] - 7s 83us/step - loss: 1.1735
Epoch 87/100
90104/90104 [==============================] - 8s 94us/step - loss: 1.1584
Epoch 88/100
90104/90104 [==============================] - 7s 77us/step - loss: 1.1421
Epoch 89/100
90104/90104 [==============================] - 7s 78us/step - loss: 1.1263
Epoch 90/100
90104/90104 [==============================] - 7s 79us/step - loss: 1.1112
Epoch 91/100
90104/90104 [==============================] - 7s 73us/step - loss: 1.0948
Epoch 92/100
90104/90104 [==============================] - 7s 78us/step - loss: 1.0788
Epoch 93/100
90104/90104 [==============================] - 10s 111us/step - loss: 1.0639
Epoch 94/100
90104/90104 [==============================] - 8s 89us/step - loss: 1.0484
Epoch 95/100
90104/90104 [==============================] - 7s 80us/step - loss: 1.0338
Epoch 96/100
90104/90104 [==============================] - 7s 75us/step - loss: 1.0186
Epoch 97/100
90104/90104 [==============================] - 8s 85us/step - loss: 1.0040
Epoch 98/100
90104/90104 [==============================] - 6s 70us/step - loss: 0.9889
Epoch 99/100
90104/90104 [==============================] - 7s 75us/step - loss: 0.9746
Epoch 100/100
90104/90104 [==============================] - 8s 90us/step - loss: 0.9603
```

[62]:

```
book_embedding_learnt = model.get_layer(name='Non-Negative-Book-Embedding').
 ↪get_weights()[0]
pd.DataFrame(book_embedding_learnt).describe()
```

[62]:

```
                 0            1            2            3            4
count  3152.000000  3152.000000  3152.000000  3152.000000  3152.000000
mean      2.170481     2.187744     2.195058     2.183846     2.182026
std       0.618222     0.598144     0.622132     0.603437     0.606997
min       0.001234    -0.000000    -0.000000    -0.000000    -0.000000
25%       1.782540     1.816230     1.825028     1.816823     1.818164
50%       2.176932     2.211349     2.208902     2.182009     2.201387
75%       2.564777     2.552419     2.580720     2.546772     2.569596
max       5.508662     5.021390     4.844610     5.321653     4.839131
```

```
[63]: user_embedding_learnt = model.get_layer(name='Non-Negative-User-Embedding').
       ↪get_weights()[0]
      pd.DataFrame(user_embedding_learnt).describe()
```

```
[63]:                  0             1             2             3             4
      count 34574.000000  34574.000000  34574.000000  34574.000000  34574.000000
      mean      0.610343      0.610357      0.611139      0.609801      0.610174
      std       0.291488      0.292264      0.294033      0.291579      0.289319
      min      -0.000000     -0.000000     -0.000000     -0.000000     -0.000000
      25%       0.497563      0.498050      0.497417      0.496602      0.500090
      50%       0.680115      0.679097      0.679868      0.680402      0.679900
      75%       0.798750      0.799705      0.800458      0.798680      0.798616
      max       2.440540      2.510997      2.089963      2.239646      2.464717
```

```
[64]: y_hat = np.round(model.predict([df_test["User_index"], df_test["ISBN_index"]]),0)
      y_true = df_test["Book-Rating"]
```

```
[65]: from sklearn.metrics import mean_absolute_error
      mean_absolute_error(y_true, y_hat)
```

```
[65]: 2.6270253473609446
```

L’erreur moyenne absolue est conséquente et les résultats sont décevants.

## 5 Conclusion

Différentes approches ont été mises en oeuvre : * Approche basée sur l’indicateur IMDB qui est relativement simple, * Approche basée sur la similarité des livres en utilisant les K plus proches voisins et la métrique cosinus * 2 approches collaboratives basées sur la factorisation de matrice en utilisant SVD et deep learning.

Deux sentiments : * Certaines méthodes ne peuvent pas passer à l’échelle ou peut-être faut-il utiliser des structures de données adéquates pour gérer le caractère sparse des matrices * Il n’a pas été possible de développer des approches prenant en compte des informations sur le contenu, faute de données dans le dataset permettant d’avoir par exemple le genre du livre. Plus on a de données et plus le système de recommandation pourra être efficace ce qui apparait assez limitant dans le cas présent.