DMT2023_HW2

April 20, 2023

0.1 Group composition:

————YOUR TEXT STARTS HERE————

Ben Kakitie, Joanna O., 1853548

Lauro, Francesco, 1706784

0.2 Homework 2

The homework consists of two parts:

1. PageRank

and

2. Recommendation System

   Ensure that the notebook can be faithfully reproduced by anyone (hint: pseudo random number generation).

   If you need to set a random seed, set it to 24.

1 Part 1

In this part of the homework, you have to deal with the PageRank algorithm.

[ ]:
```
#REMOVE_OUTPUT#
!pip install --upgrade --no-cache-dir gdown
from bs4 import BeautifulSoup
#YOUR CODE STARTS HERE#

!pip install networkx
import networkx as nx

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

1.1 Part 1.1

The data you need to process comes from the book *Le Morte D'Arthur* by Thomas Malory. The dataset you need to build should be an unweighted and undirected graph, where nodes represent characters from the book and an edge connects two characters in the graph if their names appeared at least one time in the same chapter.

Using this dataset, you must then run various PageRank algorithms.

1.1.1 1.1.1

Download the data from the Drive link (code already provided).

[ ]:
```
#REMOVE_OUTPUT#
!gdown 1zHgvidy9FvhZvE68SOmXWkoF-hHMpiUL
!gdown 1VjpTkFcbfaLIi4TXVafokW9e_bvGnfut
```

1.1.2 1.1.2

Parse the HTML. **Part** of code already provided: follow the comments to complete the code.

[3]:
```
with open('The Project Gutenberg eBook of Le Morte D'Arthur, Volume I (of II),␣
↪by Thomas Malory.html') as fp:
    vol1 = BeautifulSoup(fp, 'html.parser')
with open('The Project Gutenberg eBook of Le Morte D'Arthur, Volume II (of II),␣
↪by Thomas Malory.html') as fp:
    vol2 = BeautifulSoup(fp, 'html.parser')

def clean_text(txt):
    words_to_put_space_before = [".",",",";",":","'","'"]
    words_to_lowercase =␣
↪["First","How","Some","Yet","Of","A","The","What","Fifth"]

    app = txt.replace("\n"," ")
    for word in words_to_put_space_before:
        app = app.replace(word," "+word)
    for word in words_to_lowercase:
        app = app.replace(word+" ",word.lower()+" ")
    return app.strip()

def parse_html(soup):
    titles = []
    texts = []
    for chapter in soup.find_all("h3"):
        chapter_title = chapter.text
        if "CHAPTER" in chapter_title:
            chapter_title = clean_text("".join(chapter_title.split(".")[1:]))
            titles.append(chapter_title)

            chapter_text = [p.text for p in chapter.findNextSiblings("p")]
            chapter_text = clean_text(" ".join(chapter_text))
            texts.append(chapter_text)
    return titles, texts
```

[4]:
```
#YOUR CODE STARTS HERE#
#Extract all the chapters' titles and texts from the two volumes
title1, text1 = parse_html(vol1)
title2, text2 = parse_html(vol2)
title = [*title1, *title2]
text = [*text1, *text2]

docno1 = [str(l) for l in list(range(1, len(title1)+1))]
docno2 = [str(l) for l in list(range(len(title1)+1, len(title1) +␣
↪len(title2)+1))]
docno = [*docno1, *docno2]
```

```python
#Transform the list into a pandas DataFrame.


df = pd.DataFrame(list(zip(docno, title, text)), columns =['docno', 'title',␣
↪'text'])




#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[5]:
```python
#YOUR CODE STARTS HERE#

df.head(8)





#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[5]:
```
  docno                                              title  \
0     1  first , how Uther Pendragon sent for the duke …
1     2  how Uther Pendragon made war on the duke of Co…
2     3     of the birth of King Arthur and of his nurture
3     4                of the death of King Uther Pendragon
4     5  how Arthur was chosen king , and of wonders an…
5     6  how King Arthur pulled out the sword divers times
6     7  how King Arthur was crowned , and how he made …
7     8  how King Arthur held in Wales , at a Pentecost…

                                                text
0  It befell in the days of Uther Pendragon , whe…
1  Then Ulfius was glad , and rode on more than a…
2  Then Queen Igraine waxed daily greater and gre…
3  Then within two years King Uther fell sick of …
4  Then stood the realm in great jeopardy long wh…
5  Now assay , said Sir Ector unto Sir Kay . And …
6  And at the feast of Pentecost all manner of me…
7  Then the king removed into Wales , and let cry…
```

1.1.3 1.1.3

Extract character's names from the **titles** only. **Part** of code already provided: follow the comments to complete the code.

[6]:
```
all_characters = set()
def extract_character_names_from_string(string_to_parse):
    special_tokens = ["of","the","le","a","de"]

    remember = ""
    last_is_special_token = False

    tokens = string_to_parse.split(" ")
    characters_found = set()
    for i,word in enumerate(tokens):
        if word[0].isupper() or (remember!="" and word in special_tokens):
            #word = word.replace("'s","").replace("’s","")
            last_is_special_token = False
            if remember!="":
                if word in special_tokens:
                    last_is_special_token = True
                remember = remember+" "+word
            else: remember = word
        else:
            if remember!="":
                if last_is_special_token:
                    for tok in special_tokens:
                        remember = remember.replace(" "+tok,"")
                characters_found.add(remember)
            remember = ""
            last_is_special_token = False
    return characters_found

#all_characters = set([x for x in all_characters if x[-2:]!="'s"])
```

[7]:
```
#YOUR CODE STARTS HERE#
#Extract all characters' names

for i in (title):
  all_characters.update(extract_character_names_from_string(i)) #number of␣
 ↪characters = 225
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

[8]:

```
#YOUR CODE STARTS HERE#

for k in all_characters:
  if 'King' in k:
    print(k) #number of "King" characters = 25



#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

```
King of the Land of Cameliard
King
King Bagdemagus
King Brandegore
King Ban
King Pelleas
King Anguish of Ireland
King Uriens
King Howel of Brittany
King Lot of Orkney
King Bors
King Pellam
King Arthur
King Lot
King Rience
King Pellinore
King Mordrains
King Solomon
King Mark of Cornwall
King Mark
King Leodegrance
King Pelles
King Evelake
King of England
Maimed King
```

1.1.4 1.1.4

Some names refer to the same characters (e.g. `'Arthur' = 'King Arthur'`). A function is provided to extract the disambiguation dictionary: each key represents a name and the value represents the true character name (e.g. `{'Arthur': 'King-Arthur', 'King': 'King-Arthur', 'Bedivere':'Sir Bedivere'}`). Disambiguation sets, i.e. a list with sets representing the multiple names of a single character, are also provided.

> There may be some mistakes, but it does not matter (e.g. `'Cornwall' = 'King of Cornwall'`)

[9]:
```
disambiguate_to = {}
for x in all_characters:
    for y in all_characters:
        if x in y and x!=y:
            if x in disambiguate_to:
                previous_y = disambiguate_to[x]
                if len(y)>len(previous_y): disambiguate_to[x] = y
            else:
                disambiguate_to[x] = y
disambiguate_to.update({"King": "King Arthur",
                        "King of England": "King Arthur",
                        "Queen": "Queen Guenever",
                        "Sir Lancelot": "Sir Launcelot"})

disambiguate_sets = []
for x,y in disambiguate_to.items():
    inserted = False
    for z in disambiguate_sets:
        if x in z or y in z:
            z.add(x); z.add(y)
            inserted = True
    if not inserted:
        disambiguate_sets.append(set([x,y]))

while True:
    to_remove,to_add = [],[]
    for i1,s1 in enumerate(disambiguate_sets[:-1]):
        for s2 in disambiguate_sets[i1+1:]:
            if len(s1.intersection(s2))>0:
                to_remove.append(s1)
                to_remove.append(s2)
                to_add.append(s1.union(s2))
    if len(to_add)>0:
        for rm in to_remove:
            disambiguate_sets.remove(rm)
        for ad in to_add:
            disambiguate_sets.append(ad)
```

```
    else: break
```

1.1.5 1.1.5

Prepare the dataset for the PageRank algorithm.

> It should be a Pandas DataFrame with two fields: `character_1`, `character_2`.
>
> Each row must contain two characters' names if they appear together in at least one chapter **text**.
>
> The relevant characters are only those extracted in Part 1.1.3.
>
> Keep in mind that some characters have alternative names, but they refer to the same character.
>
> The dataset must not contain repetitions.

[10]:
```
#YOUR CODE STARTS HERE#

characters = list(all_characters)
pr_dict = {}

for t in text:
  for c in range(0, len(characters)-1):
    # print(c)
    if characters[c] and characters[c+1] in t:
      pr_dict[characters[c]] = characters[c+1]

# print(pr_dict)







pr_df = pd.DataFrame(pr_dict.items(), columns =['character_1', 'character_2']).
 ↪drop_duplicates()











#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

[11]:
```
#YOUR CODE STARTS HERE#

pr_df.loc[(pr_df['character_1'] == 'Sir Lamorak') | (pr_df['character_2'] ==␣
↪'Sir Lamorak')]





#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[11]:

	character_1	character_2
116	Ettard	Sir Lamorak

1.1.6 1.1.6

Print the sorted list of all character names (without duplicates) in ascending alphabetical order.

Print also the length of this list.

[12]:
```
#YOUR CODE STARTS HERE#



char_names = sorted(list(all_characters)) #there are no duplicates since they␣
↪are not allowed in set() objects


print(char_names)
print(len(char_names))









#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

```
['Abbot', 'Accolon', 'Alice', 'Alisander', 'Alisander le Orphelin', 'Almaine',
'Almesbury', 'Andred', 'Anglides', 'Archbishop of Canterbury', 'Arthur',
'Astolat', 'Avoutres', 'Bagdemagus', 'Balan', 'Balin', 'Ban', 'Beale Isoud',
'Beale Pilgrim', 'Beaumains', 'Benwick', 'Bors', 'Boudwin', 'Bragwaine', 'Breuse
Saunce Pité', 'Camelot', 'Carbonek', 'Carlion', 'Castle Lonazep', 'Castle of
Maidens', 'Castle of Pendragon', 'Chapel Perilous', 'Christmas', 'Constantine',
'Cornwall', 'Corsabrin', 'Court', 'Dagonet', 'Dame Brisen', 'Dame Elaine',
'Damosel of the Lake', 'David', 'Dinadan', 'Dover', 'Elaine', 'Elias',
'England', 'Epinogris', 'Ettard', 'Excalibur', 'Fair Maid of Astolat', 'Feast of
Pentecost', 'Forest Perilous', 'France', 'Gaheris', 'Galahad', 'Gard', 'Garlon',
'Gawaine', 'God', 'Gouvernail', 'Great Royalty', 'Griflet', 'Guenever', 'Helin
le Blank', 'Holy Sangreal', 'Humber', 'Igraine', 'Ireland', 'Island', 'Isle',
'Isoud', 'Joseph', 'Joyous Gard', 'Joyous Isle', 'Kehydius', 'King', 'King
Anguish of Ireland', 'King Arthur', 'King Bagdemagus', 'King Ban', 'King Bors',
'King Brandegore', 'King Evelake', 'King Howel of Brittany', 'King Leodegrance',
'King Lot', 'King Lot of Orkney', 'King Mark', 'King Mark of Cornwall', 'King
Mordrains', 'King Pellam', 'King Pelleas', 'King Pelles', 'King Pellinore',
'King Rience', 'King Solomon', 'King Uriens', 'King of England', 'King of the
Land of Cameliard', 'Knight of the Black Launds', 'Knight of the Red Launds',
'Knights of the Round Table', 'La Beale Isoud', 'La Cote Male Taile', 'Lady
Ettard', 'Lady Lionesse', 'Lady of the Lake', 'Lambegus', 'Lanceor',
```

```
'Launcelot', 'Leodegrance', 'Lionel', 'Logris', 'Lonazep', 'Lucius', 'Maid of
Astolat', 'Maiden of the Lake', 'Maimed King', 'Maledisant', 'May-day',
'Melias', 'Merlin', 'Mordred', 'Morgan', 'Morgan le Fay', 'Nero', 'Our Lord',
'Palamides', 'Palomides', 'Pelles', 'Pentecost', 'Percivale', 'Pope', 'Queen',
'Queen Guenever', 'Queen Igraine', 'Queen Isoud', 'Queen Morgan le Fay', 'Queen
of Orkney', 'Questing Beast', 'Red Knight', 'Romans', 'Rome', 'Round Table',
'Sangreal', 'Saracen', 'Saracens', 'Siege Perilous', 'Sir Accolon', 'Sir Accolon
of Gaul', 'Sir Aglovale', 'Sir Agravaine', 'Sir Alisander', 'Sir Amant', 'Sir
Anguish', 'Sir Archade', 'Sir Beaumains', 'Sir Bedivere', 'Sir Belliance', 'Sir
Berluse', 'Sir Blamore', 'Sir Bleoberis', 'Sir Bliant', 'Sir Bors', 'Sir
Breunor', 'Sir Breuse Saunce Pité', 'Sir Brian', 'Sir Carados', 'Sir
Colgrevance', 'Sir Dagonet', 'Sir Dinadan', 'Sir Ector', 'Sir Elias', 'Sir
Epinogris', 'Sir Frol', 'Sir Gaheris', 'Sir Galahad', 'Sir Galahalt', 'Sir
Galihodin', 'Sir Gareth', 'Sir Gawaine', 'Sir Kay', 'Sir Lamorak', 'Sir Lamorak
de Galis', 'Sir Lancelot', 'Sir Lanceor', 'Sir Launcelot', 'Sir Lavaine', 'Sir
Lionel', 'Sir Mador', 'Sir Malgrin', 'Sir Marhaus', 'Sir Meliagaunce', 'Sir
Meliagrance', 'Sir Mordred', 'Sir Nabon', 'Sir Palomides', 'Sir Pedivere', 'Sir
Pelleas', 'Sir Percivale', 'Sir Persant', 'Sir Persant of Inde', 'Sir
Pervivale', 'Sir Sadok', 'Sir Safere', 'Sir Sagramore le Desirous', 'Sir
Segwarides', 'Sir Suppinabiles', 'Sir Tor', 'Sir Tristram', 'Sir Tristram de
Liones', 'Sir Turquine', 'Sir Uriens', 'Sir Urre', 'Sir Uwaine', 'Solomon',
'Surluse', 'Tintagil', 'Tristram', 'Ulfius', 'Uther Pendragon', 'Wales',
'Winchester', 'York']
225
```

1.1.7 1.1.7

Create the adjacency matrix for the graph, assigning to each character a node identifier equal to the index that the character name has in ascending alphabetical order (remember that the first element of a list in Python has index 0).

[13]:
```
#YOUR CODE STARTS HERE#

sorted_pr_df = pr_df.sort_values(by=['character_1'])

G = nx.from_pandas_edgelist(sorted_pr_df, source='character_1',␣
↪target='character_2', create_using=nx.Graph())

G_adj = nx.adjacency_matrix(G)


# print(G_adj)  --> # (0, 1)        1
                    # (0, 168)        1
                    # (1, 0)        1
                    # (2, 3)        1
                    # (2, 198)        1
                    # ...



#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

1.1.8 1.1.8

Compute the PageRank vector for the obtained graph using a damping factor of 0.85.

[14]:
```
#YOUR CODE STARTS HERE#

damping_factor = 0.85
alpha = 1. - damping_factor

pr = nx.pagerank(G, alpha=alpha, max_iter=1000)


def return_top_scores(pr, topic, k):
  pr_dict = dict(sorted(pr.items()))
  topk = {}
  for key, value in pr_dict.items():
    if topic in key:
      topk[key] = value
      if len(topk) == k:
        break
```

```
  return list(topk.items())[:k]

#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[15]:
```
#YOUR CODE STARTS HERE#

k = 15
top15 = return_top_scores(pr, "", k)
# print(top15)

pd.DataFrame(top15, columns=["Name","PageRank score"])

#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[15]:

	Name	PageRank score
0	Abbot	0.004486
1	Accolon	0.004464
2	Alice	0.004464
3	Alisander	0.004464
4	Alisander le Orphelin	0.004466
5	Almaine	0.004464
6	Almesbury	0.004755
7	Andred	0.004464
8	Anglides	0.004464
9	Archbishop of Canterbury	0.004466
10	Arthur	0.004754
11	Astolat	0.004754
12	Avoutres	0.004464
13	Bagdemagus	0.004464
14	Balan	0.004464

1.1.9 1.1.9

Compute the Topic-specific PageRank vector for the obtained graph using a damping factor of 0.75, by considering as topic the *Queens*: a character belongs to the topic if its name starts with the string Queen.

[16]:

```
#YOUR CODE STARTS HERE#

damping_factor = 0.75
alpha = 1. - damping_factor

ts_pr = nx.pagerank(G, alpha=alpha, max_iter=1000)
# print(ts_pr)













#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[17]:

```
#YOUR CODE STARTS HERE#

k = 16
top16 = return_top_scores(ts_pr, "Queen", k)
# print(top16)

pd.DataFrame(top16, columns=["Name","PageRank score"])

#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[17]:

	Name	PageRank score
0	Queen	0.004904
1	Queen Guenever	0.004520
2	Queen Igraine	0.004465
3	Queen Isoud	0.004472
4	Queen Morgan le Fay	0.004472
5	Queen of Orkney	0.004520

1.1.10 1.1.10

Compute the Personalized PageRank vector for the obtained graph using a damping factor of 0.2 for each of the *Knights*: a character belongs to the topic if its name starts with the string `Sir`.

[18]:
```
#YOUR CODE STARTS HERE#

damping_factor = 0.2
alpha = 1. - damping_factor

# arg = [i for i in G.nodes() if dict(G.nodes()).items() == 'Sir']
p_pr = nx.pagerank(G, alpha=alpha)

# print(arg)


# node_dict = dict(G.nodes(data=True))

# # for el in node_dict:
# #   if 'Sir' in el:
# #     print(el)
# # node_dict['Sir']


# print([el for el in node_dict if 'Sir' in el])









#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

[19]:
```
#YOUR CODE STARTS HERE#



k = 2
top2 = return_top_scores(p_pr, "Sir", k)
# print(top2)

pd.DataFrame(top2, columns=["Name","PageRank score"])
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[19]:

	Name	PageRank score
0	Sir Accolon	0.004837
1	Sir Accolon of Gaul	0.002983

1.1.11 1.1.11

Compute Topic-specific PageRank for the graph using a damping factor of 0.2. Imagine you are in an **online** context.

The Topic is *Knights* (list of characters defined in step 1.1.7)

[20]:
```
#YOUR CODE STARTS HERE#

# char_names = sorted(list(all_characters))

damping_factor = 0.2
alpha = 1. - damping_factor

online_ts_pr = nx.pagerank(G, alpha=alpha)











#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[21]:
```
#YOUR CODE STARTS HERE#

k = 8
top8 = return_top_scores(online_ts_pr, "Sir", k)
# print(top8)

pd.DataFrame(top8, columns=["Name","PageRank score"])

#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[21]:

	Name	PageRank score
0	Sir Accolon	0.004837
1	Sir Accolon of Gaul	0.002983
2	Sir Aglovale	0.004476
3	Sir Agravaine	0.004470
4	Sir Alisander	0.004465
5	Sir Amant	0.004465
6	Sir Anguish	0.002983
7	Sir Archade	0.004473

1.2 Part 1.2

1.2.1 1.2.1

Given a graph with n nodes: * Node A is connected to all the other nodes. * There are no other edges.

What will be the PageRank of node A?

Does the result depend on the damping factor or number of nodes n? If yes, please describe the value of PageRank as both vary.

Use at most 3 sentences.

———-YOUR TEXT STARTS HERE———-

The PageRank of node A can be almost twice as much respect to the PageRank of the other nodes, while the other nodes will all have the same probability distribution (star graph).

The result depends on the damping factor, given that the PageRank increases as it decreases (and vice-versa).

The result also depends on the number of nodes, given that the PageRank increases as it increases, even though the change in probability distribution isn't too noticeable.

2 Part 2

In this part of the homework, you have to improve the performance of various recommendation-systems by using non-trivial algorithms and also by performing the tuning of the hyper-parameters.

[ ]:
```
#REMOVE_OUTPUT#
#YOUR CODE STARTS HERE#
import pandas as pd
import os

import multiprocessing

!pip install scikit-surprise
from surprise import Dataset
from surprise import Reader
from surprise import prediction_algorithms
from surprise.model_selection import KFold
from surprise.model_selection import cross_validate
#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

2.1 Part 2.1

Apply **all** algorithms for recommendation made available by "Surprise" libraries on the provided dataset: * **with their default configuration** * using **ALL** CPU-cores available on the remote machine by specifying the value in an **explicit** way with an integer number.

You also need to: * use Alternating Least Squares as baselines estimation method * use cosine similarity as similarity measure * use item-item similarity * if a number of iterations is to be set, it must be 25

> Not all options may be applicable to all algorithms

2.1.1 2.1.1

Prepare the dataset for the Recommendation algorithms.

> It should be a Pandas DataFrame with three fields: `Ruler`, `Knight`, `Rating`.
>
> Each row must contain two characters' names if they appear together in at least one chapter **text**.
>
> The relevant characters are only those extracted in Part 1.1.3.
>
> Keep in mind that some characters have alternative names, but they refer to the same character.
>
> The dataset must not contain repetitions.

Also:

> A `Ruler` is a character whose name starts with `King` or `Queen`.

A `Knight` is a character whose ame starts with `Knight` or `Sir`.

The `Rating` represents the number of chapters in which two characters appear together.

[23]:
```
#YOUR CODE STARTS HERE#

characters = list(all_characters)

Ruler, Knight, Rating =  [], [], []
for c in characters:
  if ("King" in c) or ("Queen" in c): Ruler.append(c)
  if ("Knight" in c) or ("Sir" in c): Knight.append(c)
# print(Ruler, len(Ruler), sep='\n')
# print(Knight, len(Knight), sep='\n')

r_dict = {}
for t in text:
  for r in range(0, len(Ruler)):
    for k in range(0, len(Knight)):
      if Ruler[r] and Knight[k] in t:
        if (Ruler[r], Knight[k]) in r_dict: r_dict[(Ruler[r], Knight[k])] += 1
        else: r_dict[(Ruler[r], Knight[k])] = 1
# print(r_dict)

Rating = list(r_dict.values())
# print(Rating, len(Rating), sep='\n')

Ruler2 = [e[0] for e in list(r_dict.keys())]
Knight2 = [e[1] for e in list(r_dict.keys())]
r_df = pd.DataFrame(list(zip(Ruler2, Knight2, Rating)), columns =['Ruler',␣
↪'Knight', 'Rating'])
r_df.head()

#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

[23]:

	Ruler	Knight	Rating
0	King of the Land of Cameliard	Sir Ector	62
1	King	Sir Ector	62
2	Queen	Sir Ector	62
3	King Bagdemagus	Sir Ector	62
4	Queen Isoud	Sir Ector	62

2.1.2 2.1.2

Inspect the dataset:

1. For each field, print the minimum and maximum values.

2. Print also the rows of the dataset where `Sir Accolon` appears.

[24]:
```
#YOUR CODE STARTS HERE#



print(r_df.agg(['min', 'max']))
print('\n')


print(r_df.loc[(r_df['Knight'] == 'Sir Accolon')])





#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

```
                Ruler                        Knight  Rating
min              King  Knight of the Red Launds       1
max   Queen of Orkney                Sir Uwaine     273



                            Ruler       Knight  Rating
217  King of the Land of Cameliard  Sir Accolon       6
219                           King  Sir Accolon       6
221                          Queen  Sir Accolon       6
223                King Bagdemagus  Sir Accolon       6
225                    Queen Isoud  Sir Accolon       6
227                King Brandegore  Sir Accolon       6
229                       King Ban  Sir Accolon       6
231                   King Pelleas  Sir Accolon       6
233        King Anguish of Ireland  Sir Accolon       6
235                    King Uriens  Sir Accolon       6
237         King Howel of Brittany  Sir Accolon       6
239             King Lot of Orkney  Sir Accolon       6
241                 Queen Guenever  Sir Accolon       6
243                      King Bors  Sir Accolon       6
245                    King Pellam  Sir Accolon       6
247                    King Arthur  Sir Accolon       6
249                       King Lot  Sir Accolon       6
251                    King Rience  Sir Accolon       6
253                Queen of Orkney  Sir Accolon       6
255                  Queen Igraine  Sir Accolon       6
257                 King Pellinore  Sir Accolon       6
259                 King Mordrains  Sir Accolon       6
261                   King Solomon  Sir Accolon       6
263           King Mark of Cornwall  Sir Accolon       6
```

265	King Mark	Sir Accolon	6
267	King Leodegrance	Sir Accolon	6
269	Queen Morgan le Fay	Sir Accolon	6
271	King Pelles	Sir Accolon	6
273	King Evelake	Sir Accolon	6
275	King of England	Sir Accolon	6
277	Maimed King	Sir Accolon	6

2.1.3 2.1.3

Load the dataset into the appropriate scikit-surprise structure.

[25]:
```
#YOUR CODE STARTS HERE#
#r_df['Timestamp'] = 0
file_name = 'HW_2__Ruler__Knight__Rating.csv'
cwd = os.getcwd()        # get current working directory
file_path = cwd + file_name
r_df.to_csv(file_path, index = False)
reader = Reader(line_format='user item rating', sep=',', rating_scale=[1, 273],␣
 ↪skip_lines=1)
data = Dataset.load_from_file(file_path, reader=reader)
#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

2.1.4 2.1.4

Initialize a `scikit-surprise` `KFold` object with 3-folds.

[26]:
```
#YOUR CODE STARTS HERE#

kf_3 = KFold(n_splits=3, random_state=42)






#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

2.1.5 2.1.5

Define **all** the algorithms you are going to use

[27]:
```
#YOUR CODE STARTS HERE#

NormalPredictor = prediction_algorithms.random_pred.NormalPredictor()
BaselineOnly = prediction_algorithms.baseline_only.BaselineOnly()
KNNBasic = prediction_algorithms.knns.KNNBasic()
KNNWithMeans = prediction_algorithms.knns.KNNWithMeans()
KNNWithZScore = prediction_algorithms.knns.KNNWithZScore()
KNNBaseline = prediction_algorithms.knns.KNNBaseline()
SVD = prediction_algorithms.matrix_factorization.SVD()
SVDpp = prediction_algorithms.matrix_factorization.SVDpp()
NMF = prediction_algorithms.matrix_factorization.NMF()
SlopeOne = prediction_algorithms.slope_one.SlopeOne()
CoClustering = prediction_algorithms.co_clustering.CoClustering()
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

2.1.6 2.1.6

Define the parameter configurations for each selected algorithm.

Each configuration must be a python `dict`.

Ensure that the definition meets the requirements of Part 2, but is also as minimal as possible (the fewer parameters you define, the better).

> Tip: dictionaries can be passed to methods using `**`. Example:

```
def method_name(param1, param2):
    return param1+param2
py_dict = {param1: 4, param2:2}
method_name(**py_dict) #gives 6
```

[28]:

```
#YOUR CODE STARTS HERE#
NormalPredictor_config_def, SlopeOne_config_def = {}, {}
BaselineOnly_config_def = {"bsl_options": {"method": "als", "n_epochs": 10,␣
 ↪"reg_u": 15, "reg_i": 10}, "verbose": True}
BaselineOnly_config_2 = {"bsl_options": {"method": "als", "n_epochs": 25,␣
 ↪"reg_u": 15, "reg_i": 10}, "verbose": True}
KNNBasic_config_def = {"k": 40, "min_k": 1, "sim_options": {'name': 'MSD',␣
 ↪"user_based": True, 'min_support': 0}, "verbose": True}
KNNBasic_config_2 = {"k": 40, "min_k": 1, "sim_options": {'name': 'cosine',␣
 ↪"user_based": False, 'min_support': 0}, "verbose": True}
KNNWithMeans_config_def = {"k": 40, "min_k": 1, "sim_options": {'name': 'MSD',␣
 ↪"user_based": True, 'min_support': 0}, "verbose": True}
KNNWithMeans_config_2 = {"k": 40, "min_k": 1, "sim_options": {'name': 'cosine',␣
 ↪"user_based": False, 'min_support': 0}, "verbose": True}
KNNWithZScore_config_def = {"k": 40, "min_k": 1, "sim_options": {'name': 'MSD',␣
 ↪"user_based": True, 'min_support': 0}, "verbose": True}
KNNWithZScore_config_2 = {"k": 40, "min_k": 1, "sim_options": {'name':␣
 ↪'cosine', "user_based": False, 'min_support': 0}, "verbose": True}
KNNBaseline_config_def = {"k": 40, "min_k": 1, "sim_options": {'name': 'MSD',␣
 ↪"user_based": True, 'min_support': 0},
                          "bsl_options": {"method": "als", "n_epochs": 10,␣
 ↪"reg_u": 15, "reg_i": 10}, "verbose": True}
KNNBaseline_config_2 = {"k": 40, "min_k": 1, "sim_options": {'name': 'cosine',␣
 ↪"user_based": False, 'min_support': 0},
```

```
                            "bsl_options": {"method": "als", "n_epochs": 25,␣
 ↪"reg_u": 15, "reg_i": 10}, "verbose": True}
SVD_config_def = {"n_factors": 100, "n_epochs": 20, "biased": True, "init_mean":
 ↪ 0, "init_std_dev": 0.1, "lr_all": 0.005, "reg_all": 0.02, "lr_bu": None,
                  "lr_bi": None, "lr_pu": None, "lr_qi": None, "reg_bu": None,␣
 ↪"reg_bi": None, "reg_pu": None, "reg_qi": None, "random_state": None,␣
 ↪"verbose": False}
SVD_config_2 = {"n_factors": 100, "n_epochs": 25, "biased": True, "init_mean":␣
 ↪0, "init_std_dev": 0.1, "lr_all": 0.005, "reg_all": 0.02, "lr_bu": None,
                "lr_bi": None, "lr_pu": None, "lr_qi": None, "reg_bu": None,␣
 ↪"reg_bi": None, "reg_pu": None, "reg_qi": None, "random_state": None,␣
 ↪"verbose": False}
SVDpp_config_def = {"n_factors": 20, "n_epochs": 20, "init_mean": 0,␣
 ↪"init_std_dev": 0.1, "lr_all": 0.007, "reg_all": 0.02, "lr_bu": None,␣
 ↪"lr_bi": None, "lr_pu": None,
      "lr_qi": None, "lr_yj": None, "reg_bu": None, "reg_bi": None, "reg_pu":␣
 ↪None, "reg_qi": None, "reg_yj": None, "random_state": None, "verbose":␣
 ↪False, "cache_ratings": False}
SVDpp_config_2 = {"n_factors": 20, "n_epochs": 25, "init_mean": 0,␣
 ↪"init_std_dev": 0.1, "lr_all": 0.007, "reg_all": 0.02, "lr_bu": None,␣
 ↪"lr_bi": None, "lr_pu": None,
      "lr_qi": None, "lr_yj": None, "reg_bu": None, "reg_bi": None, "reg_pu":␣
 ↪None, "reg_qi": None, "reg_yj": None, "random_state": None, "verbose":␣
 ↪False, "cache_ratings": False}
NMF_config_def = {"n_factors": 15, "n_epochs": 50, "biased": False, "reg_pu": 0.
 ↪06, "reg_qi": 0.06, "reg_bu": 0.02, "reg_bi": 0.02, "lr_bu": 0.005, "lr_bi":␣
 ↪0.005, "init_low": 0,
                  "init_high": 1, "random_state": None, "verbose": False}
NMF_config_2 = {"n_factors": 15, "n_epochs": 25, "biased": False, "reg_pu": 0.
 ↪06, "reg_qi": 0.06, "reg_bu": 0.02, "reg_bi": 0.02, "lr_bu": 0.005, "lr_bi":␣
 ↪0.005, "init_low": 0,
                "init_high": 1, "random_state": None, "verbose": False}
CoClustering_config_def = {"n_cltr_u": 3, "n_cltr_i": 3, "n_epochs": 20,␣
 ↪"random_state": None, "verbose": False}
CoClustering_config_2 = {"n_cltr_u": 3, "n_cltr_i": 3, "n_epochs": 25,␣
 ↪"random_state": None, "verbose": False}
#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

2.1.7 2.1.7

Print the number of CPU cores belonging to the machine on which Colab is running.

[29]:
```
#YOUR CODE STARTS HERE#
```

```
cores = multiprocessing.cpu_count() # Count the number of cores in a computer
cores


#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[29]: 2

[30]:

```
#YOUR CODE STARTS HERE#

algorithms = {prediction_algorithms.random_pred.NormalPredictor:␣
↪[NormalPredictor_config_def],
              prediction_algorithms.baseline_only.BaselineOnly:␣
↪[BaselineOnly_config_def, BaselineOnly_config_2],
              prediction_algorithms.knns.KNNBasic: [KNNBasic_config_def,␣
↪KNNBasic_config_2],
              prediction_algorithms.knns.KNNWithMeans:␣
↪[KNNWithMeans_config_def, KNNWithMeans_config_2],
              prediction_algorithms.knns.KNNWithZScore:␣
↪[KNNWithZScore_config_def, KNNWithZScore_config_2],
              prediction_algorithms.knns.KNNBaseline: [KNNBaseline_config_def,␣
↪KNNBaseline_config_2],
              prediction_algorithms.matrix_factorization.SVD: [SVD_config_def,␣
↪SVD_config_2],
              prediction_algorithms.matrix_factorization.SVDpp:␣
↪[SVDpp_config_def, SVDpp_config_2],
              prediction_algorithms.matrix_factorization.NMF: [NMF_config_def,␣
↪NMF_config_2], prediction_algorithms.slope_one.SlopeOne:␣
↪[SlopeOne_config_def],
              prediction_algorithms.co_clustering.CoClustering:␣
↪[CoClustering_config_def, CoClustering_config_2]}

results = []
for current_algo in algorithms.keys():
  for current_config in algorithms[current_algo]:
    result = cross_validate(current_algo(**current_config), data,␣
↪measures=['RMSE'], cv=kf_3, verbose=True)
    results.append([str(current_algo), str(current_config), result])
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

```
Evaluating RMSE of algorithm NormalPredictor on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    55.9051 58.4040 53.3012 55.8701 2.0834
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.00    0.00    0.01    0.00
Estimating biases using als…
Estimating biases using als…
Estimating biases using als…
Evaluating RMSE of algorithm BaselineOnly on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    14.1440 18.4249 11.8261 14.7983 2.7334
Fit time          0.00    0.00    0.00    0.00    0.00
```

```
Test time         0.00    0.00    0.00    0.00    0.00
Estimating biases using als…
Estimating biases using als…
Estimating biases using als…
Evaluating RMSE of algorithm BaselineOnly on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    14.1440 18.4249 11.8261 14.7983 2.7334
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBasic on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.02    0.02    0.02    0.00
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBasic on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    43.2801 50.0231 38.2378 43.8470 4.8280
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.04    0.08    0.04    0.05    0.02
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    3.4306  4.3348  3.2460  3.6705  0.4758
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.02    0.02    0.02    0.00
Computing the cosine similarity matrix…
Done computing similarity matrix.
```

```
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.05    0.04    0.04    0.04    0.01
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithZScore on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    13.8617 20.3216 12.3810 15.5214 3.4476
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.02    0.02    0.02    0.00
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithZScore on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.01    0.00    0.01    0.01    0.00
Test time         0.04    0.05    0.04    0.05    0.00
Estimating biases using als…
Computing the msd similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the msd similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBaseline on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.8668  1.2247  0.7560  0.9492  0.2000
Fit time          0.00    0.00    0.00    0.00    0.00
```

```
Test time         0.02    0.02    0.05    0.03    0.01
Estimating biases using als…
Computing the cosine similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the cosine similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBaseline on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    14.2504 18.5180 11.8376 14.8687 2.7621
Fit time          0.00    0.01    0.00    0.00    0.00
Test time         0.06    0.05    0.05    0.06    0.00
Evaluating RMSE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    249.94301.2996  250.5683167.2703117.3593
Fit time          0.01    0.01    0.01    0.01    0.00
Test time         0.00    0.00    0.00    0.00    0.00
Evaluating RMSE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    249.94301.2955  250.5683167.2689117.3612
Fit time          0.02    0.02    0.02    0.02    0.00
Test time         0.01    0.00    0.00    0.01    0.00
Evaluating RMSE of algorithm SVDpp on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    249.9430246.9805250.5683249.16391.5649
Fit time          0.13    0.11    0.10    0.11    0.01
Test time         0.03    0.05    0.06    0.05    0.01
Evaluating RMSE of algorithm SVDpp on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    249.9430246.9805250.5683249.16391.5649
Fit time          0.13    0.13    0.14    0.13    0.00
Test time         0.03    0.05    0.04    0.04    0.01
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    40.5957 45.3295 36.4602 40.7951 3.6236
Fit time          0.02    0.02    0.02    0.02    0.00
Test time         0.01    0.00    0.00    0.00    0.00
Evaluating RMSE of algorithm NMF on 3 split(s).
```

```
                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    105.4793103.0791109.7387106.09902.7539
Fit time          0.01    0.01    0.01    0.01    0.00
Test time         0.00    0.00    0.00    0.00    0.00
Evaluating RMSE of algorithm SlopeOne on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.02    0.02    0.02    0.00
Evaluating RMSE of algorithm CoClustering on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    3.3754  7.5824  3.1014  4.6864  2.0508
Fit time          0.03    0.02    0.02    0.02    0.01
Test time         0.00    0.00    0.00    0.00    0.00
Evaluating RMSE of algorithm CoClustering on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    3.7933  4.0221  3.1014  3.6390  0.3914
Fit time          0.02    0.03    0.02    0.02    0.00
Test time         0.00    0.00    0.00    0.00    0.00
```

2.1.8 2.1.8

Rank all recommendation algorithms you tested according to the mean of the Mean Squared Error metric value: from the worst to the best algorithm.

Print out the ranking: algorithm name and MSE value.

[31]:
```
#YOUR CODE STARTS HERE#

tested_r_df = pd.DataFrame(columns=['Algorithm Name', 'MSE value'])

for e in results:
  algorithm_name = (e[0].split('.')[-1][0:-2])
  mean_RSE = sum(e[2]['test_rmse'])/3
  tested_r_df.loc[len(tested_r_df)] = {'Algorithm Name': algorithm_name, 'MSE␣
↪value': mean_RSE}

# print(tested_r_df)

tested_r_df2 = tested_r_df.groupby('Algorithm Name', as_index=False).mean()

# type(tested_r_df2)

print(tested_r_df2.sort_values("MSE value").to_string(index=False))











#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

```
 Algorithm Name     MSE value
       SlopeOne 5.787804e-16
   KNNWithMeans 1.835228e+00
   CoClustering 4.162698e+00
  KNNWithZScore 7.760709e+00
    KNNBaseline 7.908908e+00
   BaselineOnly 1.479830e+01
       KNNBasic 2.192351e+01
NormalPredictor 5.587013e+01
            NMF 7.344708e+01
```

```
  SVD 1.672696e+02
SVDpp 2.491639e+02
```

2.1.9 2.1.9

Select the algorithm with the best result in the previous test.

You must test a maximum of **31** possible configurations for the selected recommendation algorithm. The number of parameters specified for the various configurations must be at least 2* and no more than 5*. Also, disregard configuration limitations described at the start of Part 2.

You must obtain the best configuration among all configurations, considering the Root Mean Squared Error metric calculated on a cross-validation of **5** folds.

1. Define the configuration dictionary that will be used for parameter optimisation.
2. Find a model configuration that offers the best possible performance within the given constraints. Print this configuration.

The resulting solution must exceed the default configuration according to the Mean Absolute Error metric.

> **If a parameter is itself composed of several parameters (e.g. if it is a dictionary), each will be counted separately when calculating the total number of attributes to be optimised.

[32]:
```
#YOUR CODE STARTS HERE#
kf_5 = KFold(n_splits=5, random_state=42)
# Since the SlopeOne algorithm has 0 parameters, we'll use the second best␣
↪algorithm in the ranking, which is KNNWithMeans.
# The parameters specified for the various configurations will be k, min_k,␣
↪verbose, name and user_based (the last 2 will be
# sub-arguments of the sim_option argument)
KNNWithMeans_config_def = {"k": 40, "min_k": 1, "sim_options": {'name': 'MSD',␣
↪"user_based": True}, "verbose": True}
config_dict = {"k": [20,40], "min_k": [1,2], 'name': ['cosine', 'MSD',␣
↪'pearson', 'pearson_baseline'], "user_based": [True, False]}
results = []
for k in config_dict['k']:
  for min_k in config_dict['min_k']:
    for name in config_dict['name']:
      for user_based in config_dict['user_based']:
        if (k != 40 or min_k != 1 or name != 'MSD' or user_based != True):
          current_config = {"k": k, "min_k": min_k, "sim_options": {'name':␣
↪name, "user_based": user_based}, "verbose": True}
          result = cross_validate(prediction_algorithms.knns.
↪KNNWithMeans(**current_config), data, measures=['RMSE'], cv=kf_5,␣
↪verbose=True)
          results.append([str(current_config), result])
config_df = pd.DataFrame(columns=['Configuration', 'Mean RSE'])
for e in results:
  configuration, mean_RSE = e[0], sum(e[1]['test_rmse'])/5
  config_df.loc[len(config_df)] = {'Configuration': configuration, 'Mean RSE':␣
↪mean_RSE}
```

```python
best_config = config_df.sort_values("Mean RSE", ignore_index=True).
 ↪loc[0]['Configuration']
result = cross_validate(prediction_algorithms.knns.
 ↪KNNWithMeans(**eval(best_config)), data, measures=['MAE'], cv=kf_5,␣
 ↪verbose=True)
best_config_mae = sum(result['test_mae'])/5
result = cross_validate(prediction_algorithms.knns.
 ↪KNNWithMeans(**KNNWithMeans_config_def), data, measures=['MAE'], cv=kf_5,␣
 ↪verbose=True)
default_config_mae = sum(result['test_mae'])/5
print("\nThe best configuration according to the RMSE is:\n" + best_config)
print("Its MAE is: " + str(best_config_mae))
print("The MAE of the default configuration is: " + str(default_config_mae))
#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

```
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0588  3.1461  2.9849  2.9050  2.2739  2.8737  0.3103
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.02    0.02    0.03    0.03    0.02    0.01
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.01    0.00    0.01    0.01    0.00    0.01    0.00
```

```
Test time         0.04    0.04    0.04    0.04    0.04    0.04    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0588  3.1461  2.9849  2.9050  2.2739  2.8737  0.3103
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.03    0.04    0.03    0.03    0.03    0.03    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.01    0.00    0.00    0.00    0.00
Test time         0.05    0.05    0.05    0.04    0.05    0.05    0.00
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0588  3.1461  2.9849  2.9050  2.2739  2.8737  0.3103
Fit time          0.00    0.01    0.00    0.00    0.00    0.00    0.00
```

```
Test time         0.03    0.03    0.03    0.03    0.03    0.03    0.00
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.01    0.01    0.01    0.01    0.01    0.00
Test time         0.03    0.04    0.03    0.04    0.03    0.03    0.00
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    2.8668  2.8710  2.7881  2.8209  2.2111  2.7116  0.2521
Fit time          0.01    0.01    0.01    0.01    0.01    0.01    0.00
Test time         0.03    0.04    0.03    0.04    0.03    0.03    0.00
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
```

```
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.01    0.01    0.01    0.01    0.01    0.01    0.00
Test time         0.05    0.05    0.05    0.05    0.05    0.05    0.00
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0588  3.1461  2.9849  2.9050  2.2739  2.8737  0.3103
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.04    0.03    0.03    0.03    0.03    0.03    0.00
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.01    0.01    0.00    0.01    0.01    0.00
Test time         0.04    0.04    0.04    0.04    0.04    0.04    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
```

```
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0588  3.1461  2.9849  2.9050  2.2739  2.8737  0.3103
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.03    0.03    0.03    0.03    0.03    0.03    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.05    0.05    0.05    0.05    0.05    0.05    0.00
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0588  3.1461  2.9849  2.9050  2.2739  2.8737  0.3103
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.03    0.03    0.03    0.03    0.03    0.03    0.00
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
```

```
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.01    0.01    0.01    0.00    0.00    0.00    0.00
Test time         0.04    0.03    0.03    0.02    0.02    0.02    0.01
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    2.8668  2.8710  2.7881  2.8209  2.2111  2.7116  0.2521
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.02    0.02    0.02    0.02    0.02    0.00
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).
```

```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.01    0.01    0.01    0.00    0.01    0.01    0.00
Test time         0.02    0.02    0.02    0.02    0.02    0.02    0.00
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0605  3.1376  2.9518  2.8877  2.2598  2.8595  0.3120
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.02    0.01    0.02    0.01    0.01    0.00
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.03    0.03    0.04    0.03    0.03    0.03    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).
```

```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.01    0.00    0.00    0.00    0.00    0.00
Test time         0.03    0.03    0.03    0.03    0.03    0.03    0.00
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0605  3.1376  2.9518  2.8877  2.2598  2.8595  0.3120
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.02    0.01    0.01    0.01    0.01    0.00
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.02    0.03    0.02    0.02    0.02    0.00
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
```

```
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0289  3.1020  2.9202  2.8716  2.2534  2.8352  0.3019
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.01    0.01    0.01    0.01    0.01    0.00
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.01    0.01    0.00    0.01    0.00    0.01    0.00
Test time         0.03    0.03    0.03    0.04    0.03    0.03    0.01
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0605  3.1376  2.9518  2.8877  2.2598  2.8595  0.3120
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.01    0.01    0.01    0.01    0.01    0.00
Computing the cosine similarity matrix…
```

```
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.01    0.00    0.00    0.00
Test time         0.03    0.03    0.03    0.03    0.03    0.03    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0605  3.1376  2.9518  2.8877  2.2598  2.8595  0.3120
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.01    0.01    0.01    0.01    0.01    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.03    0.03    0.03    0.03    0.04    0.03    0.00
Computing the pearson similarity matrix…
```

```
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0605  3.1376  2.9518  2.8877  2.2598  2.8595  0.3120
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.01    0.01    0.01    0.01    0.01    0.00
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.02    0.02    0.02    0.02    0.02    0.02    0.00
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).
```

```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    3.0289  3.1020  2.9202  2.8716  2.2534  2.8352  0.3019
Fit time          0.00    0.00    0.00    0.01    0.01    0.00    0.00
Test time         0.02    0.01    0.01    0.02    0.01    0.02    0.00
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.01    0.01    0.01    0.00    0.01    0.01    0.00
Test time         0.03    0.03    0.03    0.03    0.04    0.03    0.00
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Estimating biases using als…
Computing the pearson_baseline similarity matrix…
Done computing similarity matrix.
Evaluating MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MAE (testset)     0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
Fit time          0.00    0.00    0.00    0.00    0.01    0.01    0.00
Test time         0.02    0.02    0.03    0.02    0.03    0.03    0.00
Computing the msd similarity matrix…
Done computing similarity matrix.
```

```
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Computing the msd similarity matrix…
Done computing similarity matrix.
Evaluating MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MAE (testset)     2.5022  2.4251  2.2994  2.3681  1.7817  2.2753  0.2557
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.01    0.01    0.01    0.01    0.01    0.00

The best configuration according to the RMSE is:
{'k': 20, 'min_k': 2, 'sim_options': {'name': 'pearson_baseline', 'user_based':
False}, 'verbose': True}
Its MAE is: 0.0
The MAE of the default configuration is: 2.275316807445151
```

2.2 Part 2.2

2.2.1 2.2.1

Consider this scenario:

- There are n users and m items.
- The items are divided into two groups I_A and I_B.
- Users can like (rating 1) all items in group I_A and dislike (rating 0) those in group I_B, or vice versa, but no intermediate case; thus users can also be divided into users in group U_A and users in group U_B.
- Suppose we have all n x m ratings.

Now, consider this:

- A new user u is added and we record his preference of an item i from group I_A (rating 1).

  What will be the estimated rating of an item $a \in I_A, a \neq i$ for user u if we use user-based collaborative filtering? What will be the rating of item $b \in I_B$ instead?

  If the user adds that they do not like an item j belonging to group B, how would the above ratings change ($b \neq j$)?

Use at most 3 sentences.

———-YOUR TEXT STARTS HERE———-

Given a suitable similarity metric, the estimated rating of item a (or b) for user u in a user-based collaborative filtering can be, for example, the aritmetic average of the ratings of a (or b) made by the k most similar users to u for some k, or the weighted average of the ratings of a (or b) made by all other users, with the similarities between each one of these users and u as weights.

If we use 0 to indicate missing values, the Pearson correlation coefficient similarity is always equal to 0, while the Jaccard similarity and the cosine similarity are greater than 0 only when the other users' ratings for i are 1, which means they belong to the group that can only give a 0 rating to items of group I_B, which means that if we use a weighted average as explained above the estimated rating of b for u will always be 0, and if we use the aritmetic average of the k most similar users' ratings, the only way to have an estimated rating of b for u greater than 0 is to use a value of k greater than the number of users that gave a 1 rating to i.

In the same assumption where we use 0 to indicate missing values, when the user adds that they do not like j it doesn't make any difference in terms of the similarity scores with other users, and therefore the above ratings of a and b don't change in any way.