DMT2023_HW2

April 20, 2023

0.1 Group composition:

————YOUR TEXT STARTS HERE————

Aur, Marina Iuliana, 1809715

Balestrucci, Sophia, 1713638

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#
#YOUR CODE STARTS HERE#
!pip install --upgrade --no-cache-dir gdown
from bs4 import BeautifulSoup
import pandas as pd
import numpy as np
!pip install scikit-network
from sknetwork.ranking import PageRank
import itertools




#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 1zHgvidy9FvhZvE68S0mXWkoF-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
vol1_titles, vol1_texts = parse_html(vol1)
vol2_titles, vol2_texts = parse_html(vol2)




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

```
d1 = {'title' : vol1_titles, 'text' : vol1_texts, 'vol_nr' : [1 for _ in␣
 ↪range(len(vol1_titles))]}
df_vol1 = pd.DataFrame(d1)
d2 = {'title' : vol2_titles, 'text' : vol2_texts, 'vol_nr' : [2 for _ in␣
 ↪range(len(vol2_titles))]}
df_vol2 = pd.DataFrame(d2)

df_book = pd.concat([df_vol1, df_vol2], ignore_index = True)
df_book['docno'] = [str(i) for i in range(len(df_book))]
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[5]:
```
#YOUR CODE STARTS HERE#
print("The first 8 rows of DataFrame 'df_book':")
df_book.head(8)




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

```
The first 8 rows of DataFrame 'df_book':
```

[5]:

	title	text	vol_nr	docno
0	first , how Uther Pendragon sent for the duke …	It befell in the days of Uther Pendragon , whe…	1	0
1	how Uther Pendragon made war on the duke of Co…	Then Ulfius was glad , and rode on more than a…	1	1
2	of the birth of King Arthur and of his nurture	Then Queen Igraine waxed daily greater and gre…	1	2
3	of the death of King Uther Pendragon	Then within two years King Uther fell sick of …	1	3
4	how Arthur was chosen king , and of wonders an…	Then stood the realm in great jeopardy long wh…	1	4
5	how King Arthur pulled out the sword divers times	Now assay , said Sir Ector unto Sir Kay . And …	1	5
6	how King Arthur was crowned , and how he made …	And at the feast of Pentecost all manner of me…	1	6
7	how King Arthur held in Wales , at a Pentecost…	Then the king removed into Wales , and let cry…	1	7

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 title in df_book['title']:
  all_characters.update(extract_character_names_from_string(title))
```

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

[8]:
```
#YOUR CODE STARTS HERE#
kings = []
print("The names of all the kings:\n")
for name in all_characters:
  if 'king' in name.lower() and name not in kings:
    kings.append(name)
    print(name)

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

```
The names of all the kings:

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

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_sets = disambiguate_sets.copy()
for name_set in characters_sets:
  for name in name_set:
    if name in all_characters:
        all_characters.remove(name)

for char in all_characters:
  characters_sets.append({char})

dic_of_characters_in_each_chapter = {}
for chapter in range(len(df_book)):
  for name_set in characters_sets:
    for name in name_set:
      # 'King' and 'Queen' are not valid names to determine presence of a␣
 ↪character in a document because they are not unique
      if name not in ['King', 'Queen'] and name in df_book['text'].loc[chapter]:
        char = ', '.join(name_set)
        if chapter in dic_of_characters_in_each_chapter:
          dic_of_characters_in_each_chapter[chapter].append(char)
          dic_of_characters_in_each_chapter[chapter] =␣
 ↪list(set(dic_of_characters_in_each_chapter[chapter]))
        else:
          dic_of_characters_in_each_chapter[chapter] = [char]

couples = []
for chapter in dic_of_characters_in_each_chapter:
    couples += list(itertools.
 ↪combinations(dic_of_characters_in_each_chapter[chapter], 2))
unique_couples = list(set(couples)) # to avoid duplicates
dataset = pd.DataFrame(unique_couples, columns = ['character_1', 'character_2'])
```

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

[11]:
```
#YOUR CODE STARTS HERE#
print("The rows of the dataset where 'Sir Lamorak' appears: \n")
dataset.loc[(dataset['character_1'].str.contains('Sir Lamorak')) |␣
 ↪(dataset['character_2'].str.contains('Sir Lamorak'))]




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

```
The rows of the dataset where 'Sir Lamorak' appears:
```

[11]:
```
                                      character_1  \
62                              Sir Gaheris, Gaheris
64                                  Sir Suppinabiles
136                                        Kehydius
140                Sir Lamorak, Sir Lamorak de Galis
150                Sir Lamorak, Sir Lamorak de Galis
…                                                …
4844               Sir Lamorak, Sir Lamorak de Galis
4847                                     Sir Mador
4946                                     Sir Sadok
5067                                    Gouvernail
5082  Sir Tristram de Liones, Tristram, Sir Tristram

                            character_2
62    Sir Lamorak, Sir Lamorak de Galis
64    Sir Lamorak, Sir Lamorak de Galis
136   Sir Lamorak, Sir Lamorak de Galis
140                               Wales
150                 Bors, King Bors
…                                 …
4844                         Sir Mador
4847  Sir Lamorak, Sir Lamorak de Galis
4946  Sir Lamorak, Sir Lamorak de Galis
5067  Sir Lamorak, Sir Lamorak de Galis
5082  Sir Lamorak, Sir Lamorak de Galis

[134 rows x 2 columns]
```

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#
all_characters_names = [', '.join(names) for names in characters_sets]
all_characters_names_sorted = sorted(all_characters_names)
print("List of all character names (without duplicates) in ascending␣
 ↪alphabetical order:\n")
for name in all_characters_names_sorted:
  print(name)

number_of_characters = len(all_characters_names)
print("\nLenght of the list: ", number_of_characters)









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

```
List of all character names (without duplicates) in ascending alphabetical
order:

Abbot
Alice
Alisander le Orphelin, Alisander
Almaine
Almesbury
Andred
Anglides
Archbishop of Canterbury
Astolat, Fair Maid of Astolat, Maid of Astolat
Avoutres
Bagdemagus, King Bagdemagus
Balan
Balin
Ban, King Ban
Beale Pilgrim
Benwick
Bors, King Bors
Boudwin
```

Bragwaine
Breuse Saunce Pité, Sir Breuse Saunce Pité
Camelot
Carbonek
Carlion
Castle of Maidens
Castle of Pendragon
Chapel Perilous
Christmas
Constantine
Cornwall, King Mark, King Mark of Cornwall
Corsabrin
Court
Dagonet, Sir Dagonet
Dame Brisen
Damosel of the Lake
David
Dinadan, Sir Dinadan
Dover
Elaine, Dame Elaine
Excalibur
Forest Perilous
France
Galahad, Sir Galahad
Gard, Joyous Gard
Garlon
God
Gouvernail
Great Royalty
Griflet
Helin le Blank
Humber
Igraine, Queen Igraine
Ireland, King Anguish of Ireland
Island
Joseph
Joyous Isle, Isle
Kehydius
King Arthur, King of England, Arthur, England, King
King Brandegore
King Evelake
King Howel of Brittany
King Leodegrance, Leodegrance
King Lot, King Lot of Orkney
King Mordrains
King Pellam
King Pelleas
King Pellinore

King Rience
King Solomon, Solomon
King Uriens
King of the Land of Cameliard
Knight of the Black Launds
Knight of the Red Launds
Knights of the Round Table, Round Table
La Beale Isoud, Isoud, Beale Isoud
La Cote Male Taile
Lady Ettard, Ettard
Lady Lionesse
Lady of the Lake
Lambegus
Lanceor, Sir Lanceor
Lionel, Sir Lionel
Logris
Lonazep, Castle Lonazep
Lucius
Maiden of the Lake
Maimed King
Maledisant
May-day
Melias
Merlin
Nero
Our Lord
Palamides
Pelles, King Pelles
Pentecost, Feast of Pentecost
Percivale, Sir Percivale
Pope
Queen Guenever, Guenever, Queen
Queen Isoud
Queen Morgan le Fay, Morgan le Fay, Morgan
Queen of Orkney
Questing Beast
Red Knight
Romans
Rome
Sangreal, Holy Sangreal
Saracens, Saracen
Siege Perilous
Sir Accolon, Accolon, Sir Accolon of Gaul
Sir Aglovale
Sir Agravaine
Sir Alisander
Sir Amant
Sir Anguish

Sir Archade
Sir Beaumains, Beaumains
Sir Bedivere
Sir Belliance
Sir Berluse
Sir Blamore
Sir Bleoberis
Sir Bliant
Sir Bors
Sir Breunor
Sir Brian
Sir Carados
Sir Colgrevance
Sir Ector
Sir Elias, Elias
Sir Epinogris, Epinogris
Sir Frol
Sir Gaheris, Gaheris
Sir Galahalt
Sir Galihodin
Sir Gareth
Sir Gawaine, Gawaine
Sir Kay
Sir Lamorak, Sir Lamorak de Galis
Sir Launcelot, Launcelot, Sir Lancelot
Sir Lavaine
Sir Mador
Sir Malgrin
Sir Marhaus
Sir Meliagaunce
Sir Meliagrance
Sir Mordred, Mordred
Sir Nabon
Sir Palomides, Palomides
Sir Pedivere
Sir Pelleas
Sir Persant, Sir Persant of Inde
Sir Pervivale
Sir Sadok
Sir Safere
Sir Sagramore le Desirous
Sir Segwarides
Sir Suppinabiles
Sir Tor
Sir Tristram de Liones, Tristram, Sir Tristram
Sir Turquine
Sir Uriens
Sir Urre

```
Sir Uwaine
Surluse
Tintagil
Ulfius
Uther Pendragon
Wales
Winchester
York

Lenght of the list:  170
```

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#
map__character_id__character_name = {}
for i in range(number_of_characters):
  map__character_id__character_name[i] = all_characters_names_sorted[i]


adj_matrix = np.zeros((number_of_characters, number_of_characters))
for i in range(0,number_of_characters):
  for j in range(0,number_of_characters):
    if (map__character_id__character_name[i],␣
 ↪map__character_id__character_name[j]) in unique_couples:
      adj_matrix[i][j] = 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
pagerank = PageRank(damping_factor=damping_factor, solver="piteration",␣
 ↪n_iter=1000, tol=10**-6)
pagerank_vector = pagerank.fit_transform(adj_matrix)
```

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

[15]:
```
#YOUR CODE STARTS HERE#
k = 15
list__character_id__score = []
for id, value in enumerate(pagerank_vector):
    list__character_id__score.append((id, value))
list__character_id__score.sort(key=lambda x: (x[1], x[0]), reverse=True)
top_k_pages_according_to_pagerank = [(map__character_id__character_name[id],␣
 ↪value) for id, value in list__character_id__score][:k]
top_k_pages_according_to_pagerank
#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[15]:
```
[('Sir Launcelot, Launcelot, Sir Lancelot', 0.037067394476893105),
 ('Sir Gawaine, Gawaine', 0.023020746553237652),
 ('King Arthur, King of England, Arthur, England, King', 0.02151240893048154),
 ('Queen Guenever, Guenever, Queen', 0.021426694080208954),
 ('Bors, King Bors', 0.019194417269595674),
 ('Sir Bors', 0.019006677446600628),
 ('Sir Ector', 0.01785950765806844),
 ('Sir Mordred, Mordred', 0.01760698120072515),
 ('Sir Agravaine', 0.017070483226020394),
 ('Galahad, Sir Galahad', 0.01698381702020407),
 ('King Lot, King Lot of Orkney', 0.016578583037906033),
 ('Sir Kay', 0.01637040568500791),
 ('Knights of the Round Table, Round Table', 0.015850555788311033),
 ('God', 0.014993785892697826),
 ('Sir Tristram de Liones, Tristram, Sir Tristram', 0.014578407412861523)]
```

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
pagerank = PageRank(damping_factor=damping_factor, solver="piteration",␣
 ↪n_iter=1000, tol=10 ** -6)
queens_id = []
landing_probability_queens = {}
for character_id in map__character_id__character_name:
  if 'Queen' in map__character_id__character_name[character_id]:
    queens_id.append(character_id)

print("Number of characters belonging to the topic:", len(queens_id), "\n")
for id in queens_id:
  landing_probability_queens[id] = 1. / len(queens_id)

topic_specific_queens_pagerank_vector = pagerank.fit_transform(adj_matrix,␣
 ↪weights = landing_probability_queens)




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

```
Number of characters belonging to the topic: 5
```

[17]:
```
#YOUR CODE STARTS HERE#
k = 16
list__character_id__score_queens = []
for id, value in enumerate(topic_specific_queens_pagerank_vector):
     list__character_id__score_queens.append((id, value))
list__character_id__score_queens.sort(key=lambda x: (x[1], x[0]), reverse=True)
top_k_pages_according_to_pagerank_queens =␣
 ↪[(map__character_id__character_name[id], value) for id, value in␣
 ↪list__character_id__score_queens][:k]
top_k_pages_according_to_pagerank_queens
#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[17]:
```
[('Queen Guenever, Guenever, Queen', 0.06775553161438061),
 ('Igraine, Queen Igraine', 0.05579661438958508),
 ('Queen of Orkney', 0.053772915169463184),
```

```
('Queen Morgan le Fay, Morgan le Fay, Morgan', 0.05304476751260935),
('Queen Isoud', 0.051689080006639486),
('Sir Launcelot, Launcelot, Sir Lancelot', 0.036817656692497175),
('King Lot, King Lot of Orkney', 0.02569247558428348),
('Sir Mordred, Mordred', 0.023375142117843112),
('Sir Gawaine, Gawaine', 0.01928115998983105),
('King Arthur, King of England, Arthur, England, King', 0.018041716861151812),
('Sir Ector', 0.016845332160653113),
('Bors, King Bors', 0.015692605956299857),
('Knights of the Round Table, Round Table', 0.014981513526825314),
('Sir Agravaine', 0.014907693746238633),
('Sir Tristram de Liones, Tristram, Sir Tristram', 0.014240243260470662),
('Ulfius', 0.014150781459426892)]
```

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
pagerank = PageRank(damping_factor=damping_factor, solver="piteration",␣
 ↪n_iter=1000, tol=10 ** -6)

knights_id = []
for character_id in map__character_id__character_name:
  if 'Sir' in map__character_id__character_name[character_id]:
    knights_id.append(character_id)

map__knights_id__landing_probability = {}
for id in knights_id:
  map__knights_id__landing_probability[id] = 1.

personalized_pagerank_vector_knights = pagerank.fit_transform(adj_matrix,␣
 ↪weights= map__knights_id__landing_probability)









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

[19]:
```
#YOUR CODE STARTS HERE#
for k_id in knights_id:
  page_rank_associated_nodes = [] # list of associated nodes for each knight
  for associated_node_id in range(0, len(adj_matrix[k_id])):
    if adj_matrix[k_id][associated_node_id] == 1:
      page_rank_associated_nodes.
 ↪append((map__character_id__character_name[associated_node_id],␣
 ↪personalized_pagerank_vector_knights[associated_node_id]))
```

```
 page_rank_associated_nodes = sorted(list(set(page_rank_associated_nodes)),␣
↪key = lambda x: x[1], reverse = True) # sorting
 # top-2 characters (if the knight has no neighbors, we will have an empty␣
↪list)
 print(map__character_id__character_name[k_id] , ":",␣
↪page_rank_associated_nodes[:2])









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

```
Breuse Saunce Pité, Sir Breuse Saunce Pité : [('Sir Launcelot, Launcelot, Sir
Lancelot', 0.025991029107237176), ('Sir Gawaine, Gawaine',
0.019578800660911423)]
Dagonet, Sir Dagonet : []
Dinadan, Sir Dinadan : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Galahad, Sir Galahad : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Lanceor, Sir Lanceor : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Lionel, Sir Lionel : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Percivale, Sir Percivale : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Accolon, Accolon, Sir Accolon of Gaul : [('Sir Gawaine, Gawaine',
0.019578800660911423), ('Sir Mordred, Mordred', 0.016183969448433932)]
Sir Aglovale : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Agravaine : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Alisander : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Kay', 0.016357839631001247)]
Sir Amant : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176)]
Sir Anguish : []
Sir Archade : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Galahalt', 0.014404535830197364)]
Sir Beaumains, Beaumains : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Bedivere : [('Sir Launcelot, Launcelot, Sir Lancelot',
```

0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Belliance : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Berluse : []
Sir Blamore : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Bleoberis : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Bliant : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('God', 0.004804506500823638)]
Sir Bors : [('Sir Gawaine, Gawaine', 0.019578800660911423), ('Sir Kay',
0.016357839631001247)]
Sir Breunor : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Tristram de Liones, Tristram, Sir Tristram', 0.01613936310711862)]
Sir Brian : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Carados : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Colgrevance : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Ector : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Elias, Elias : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Tristram de Liones, Tristram, Sir Tristram',
0.01613936310711862)]
Sir Epinogris, Epinogris : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Frol : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Gaheris, Gaheris : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Galahalt : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Galihodin : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Gareth : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Gawaine, Gawaine : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Dagonet, Sir Dagonet', 0.017069455222900695)]
Sir Kay : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Lamorak, Sir Lamorak de Galis : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Launcelot, Launcelot, Sir Lancelot : [('Sir Gawaine, Gawaine',
0.019578800660911423), ('Sir Nabon', 0.018028284362324135)]
Sir Lavaine : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Mador : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),

('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Malgrin : []
Sir Marhaus : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Meliagaunce : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Meliagrance : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Mordred, Mordred : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Nabon : []
Sir Palomides, Palomides : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Pedivere : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Bors', 0.01633760675947295)]
Sir Pelleas : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Persant, Sir Persant of Inde : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Pervivale : []
Sir Sadok : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Safere : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Sagramore le Desirous : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Segwarides : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Suppinabiles : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Nabon', 0.018028284362324135)]
Sir Tor : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Tristram de Liones, Tristram, Sir Tristram : [('Sir Launcelot, Launcelot,
Sir Lancelot', 0.025991029107237176), ('Sir Gawaine, Gawaine',
0.019578800660911423)]
Sir Turquine : [('Sir Launcelot, Launcelot, Sir Lancelot',
0.025991029107237176), ('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Uriens : []
Sir Urre : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]
Sir Uwaine : [('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
('Sir Gawaine, Gawaine', 0.019578800660911423)]

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#
damping_factor = 0.2
pagerank = PageRank(damping_factor=damping_factor, solver="piteration",␣
 ↪n_iter=1000, tol=10 ** -6)
landing_probability_knights = {}
print("Number of characters belonging to the topic:", len(knights_id), "\n")
for id in knights_id:
  landing_probability_knights[id] = 1. / len(knights_id)

topic_specific_knights_pagerank_vector = pagerank.fit_transform(adj_matrix,␣
 ↪weights = landing_probability_knights)









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

```
Number of characters belonging to the topic: 62
```

[21]:
```
#YOUR CODE STARTS HERE#
k = 8
list__character_id__score_knights = []
for id, value in enumerate(topic_specific_knights_pagerank_vector):
    list__character_id__score_knights.append((id, value))
list__character_id__score_knights.sort(key=lambda x: (x[1], x[0]), reverse=True)
top_k_pages_according_to_pagerank_knights =␣
 ↪[(map__character_id__character_name[id], value) for id, value in␣
 ↪list__character_id__score_knights][:k]
top_k_pages_according_to_pagerank_knights
#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[21]:
```
[('Sir Launcelot, Launcelot, Sir Lancelot', 0.025991029107237176),
 ('Sir Gawaine, Gawaine', 0.019578800660911423),
```

```
('Sir Nabon', 0.018028284362324135),
('Dagonet, Sir Dagonet', 0.017069455222900695),
('Sir Berluse', 0.016693484834301966),
('Sir Kay', 0.016357839631001247),
('Sir Bors', 0.01633760675947295),
('Sir Agravaine', 0.0162717972824295)]
```

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———-

Node A will have the **highest PageRank value** since it has incoming links from all nodes of the graph.

The PageRank value of node A *will depend on* the damping factor: decreasing the damping factor gives less weight to the likelihood of continuing to browse in node A, so it will cause the PageRank of other nodes to increase, decreasing the PageRank of node A (and viceversa).

Also the number of nodes in the graph *will affect* the PageRank value of node A: increasing the number of nodes n, also the PageRank value of node A will be higher (and viceversa) because it will have more incoming links (so also more relevance in the graph).

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#
!pip install scikit-surprise
from surprise import Reader, Dataset
from surprise.model_selection import KFold, cross_validate
from surprise.prediction_algorithms.random_pred import NormalPredictor
from surprise.prediction_algorithms.baseline_only import BaselineOnly
from surprise.prediction_algorithms.knns import KNNBasic, KNNWithMeans,␣
 ↪KNNWithZScore, KNNBaseline
from surprise.prediction_algorithms.matrix_factorization import SVD, SVDpp, NMF
from surprise.model_selection import GridSearchCV, RandomizedSearchCV
import multiprocessing
from surprise.prediction_algorithms.slope_one import SlopeOne
from surprise.prediction_algorithms.co_clustering import CoClustering
#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#
counter = {}
for couple in couples:
  if couple in counter:
    counter[couple] +=1
  else:
    counter[couple] = 1

dic = {'Ruler': [], 'Knight': [], 'Rating': []}
for couple in counter:
  if 'King' in couple[0] or 'Queen' in couple[0]:
    if 'Knight' in couple[1] or 'Sir' in couple[1]:
      dic['Ruler'].append(couple[0])
      dic['Knight'].append(couple[1])
      dic['Rating'].append(counter[couple])

rec_dataset = pd.DataFrame(dic)
rec_dataset









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

[23]:
```
                                                 Ruler  \
0           Queen Morgan le Fay, Morgan le Fay, Morgan
1                                          King Uriens
2    King Arthur, King of England, Arthur, England,…
3    King Arthur, King of England, Arthur, England,…
4                                       Ban, King Ban
..                                                 …
338                       King Lot, King Lot of Orkney
339         Cornwall, King Mark, King Mark of Cornwall
340         Queen Morgan le Fay, Morgan le Fay, Morgan
341                                    Bors, King Bors
342         Cornwall, King Mark, King Mark of Cornwall
```

```
                                    Knight  Rating
0                       Sir Gawaine, Gawaine       8
1                       Sir Gawaine, Gawaine       9
2                       Sir Gawaine, Gawaine     110
3                                 Sir Ector      40
4                                   Sir Kay       6
..                                      …     …
338  Sir Launcelot, Launcelot, Sir Lancelot       1
339                           Sir Bedivere       2
340                           Sir Bedivere       1
341                          Sir Galihodin       2
342                          Sir Galihodin       1

[343 rows x 3 columns]
```

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("The minimum and maximum values in Ruler filed:")
print(rec_dataset['Ruler'].agg(['min', 'max']))
print(" ")
print("The minimum and maximum values in Knight filed:")
print(rec_dataset['Knight'].agg(['min', 'max']))
print(" ")
print("The minimum and maximum values in Rating filed: ")
print(rec_dataset['Rating'].agg(['min', 'max']))
print(" ")
print("The rows of the dataset where 'Sir Accolon' appears:")
rec_dataset.loc[rec_dataset['Knight'].str.contains('Sir Accolon')]

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

```
The minimum and maximum values in Ruler filed:
min    Bagdemagus, King Bagdemagus
max                 Queen of Orkney
Name: Ruler, dtype: object

The minimum and maximum values in Knight filed:
min    Breuse Saunce Pité, Sir Breuse Saunce Pité
max                                    Sir Uwaine
Name: Knight, dtype: object
```

```
The minimum and maximum values in Rating filed:
min      1
max    179
Name: Rating, dtype: int64

The rows of the dataset where 'Sir Accolon' appears:
```

[24]:

	Ruler	Knight	Rating
27	Queen Morgan le Fay, Morgan le Fay, Morgan	Sir Accolon, Accolon, Sir Accolon of Gaul	9

2.1.3 2.1.3

Load the dataset into the appropriate scikit-surprise structure.

[25]:
```
#YOUR CODE STARTS HERE#
file_path = 'rec_dataset.csv'
rec_dataset.to_csv(file_path, index = False, sep = '\t')
print("Loading Dataset...")
reader = Reader(line_format = 'user item rating', sep = '\t', rating_scale =␣
↪[1, 221], skip_lines=1)
data = Dataset.load_from_file(file_path, reader = reader)
print("Done.")

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

```
Loading Dataset…
Done.
```

2.1.4 2.1.4

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

[26]:
```
#YOUR CODE STARTS HERE#
kf = 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#
algorithms = [BaselineOnly(), NormalPredictor(), KNNBasic(), KNNWithMeans(),␣
↪KNNWithZScore(), KNNBaseline(),
              SVD(), SVDpp(), NMF(), SlopeOne(), 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#
bsl_options = {"method": "als", # Alternating Least Squares as baselines␣
 ↪estimation method
               "n_epochs": 25, # if a number of iterations is to be set, it␣
 ↪must be 25
               "reg_u": 12,
               "reg_i": 5
               }
bsl_algo = BaselineOnly(bsl_options = bsl_options)

NPredictot_algo = NormalPredictor() # no parameter to set

sim_options = {
    "name": "cosine",  # use cosine similarity as similarity measure
    "user_based": False,  # use item-item similarity
}
KNNBasic_algo = KNNBasic(sim_options = sim_options)
KNNWithMeans_algo = KNNWithMeans(sim_options = sim_options)
KNNWithZScore_algo = KNNWithZScore(sim_options = sim_options)
KNNBaseline_algo = KNNBaseline(sim_options = sim_options)
```

```
SVD_algo = SVD(n_epochs = 25)  # if a number of iterations is to be set, it␣
 ↪must be 25
SVDpp_algo = SVDpp(n_epochs = 25)  # if a number of iterations is to be set, it␣
 ↪must be 25
NMF_algo = NMF(n_epochs = 25)  # if a number of iterations is to be set, it␣
 ↪must be 25

SlopeOne_algo = SlopeOne() # no parameter to set
CoClustering_algo = CoClustering(n_epochs = 25) # if a number of iterations is␣
 ↪to be set, it must be 25



#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()
cores




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

[29]: 2

[30]:
```
#YOUR CODE STARTS HERE#
algorithms = [bsl_algo, NPredictot_algo, KNNBasic_algo, KNNWithMeans_algo,␣
 ↪KNNWithZScore_algo, KNNBaseline_algo,
              SVD_algo, SVDpp_algo, NMF_algo, SlopeOne_algo, CoClustering_algo]

benchmark = []
for algo in algorithms:
  algo_name = str(algo).split(' ')[0].split('.')[-1]
  print(algo_name)
  result = cross_validate(algo, data, measures=['RMSE'], cv=kf, verbose=True,␣
 ↪n_jobs = 2)

  for key in result:
    result[key] = np.mean(result[key]) # compute the mean of the folds

  result['Algorithm'] = algo_name
  benchmark.append(result)
  print(" ")


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

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    7.6925  20.6797 8.3552  12.2425 5.9721
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.01    0.00    0.00

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    15.8818 22.2651 12.7690 16.9720 3.9527
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.01    0.00    0.00

KNNBasic
Evaluating RMSE of algorithm KNNBasic on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    9.8281  20.4867 9.3652  13.2267 5.1371
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.01    0.01    0.01    0.00
```

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    9.0225  20.0378 9.2885  12.7829 5.1311
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00

KNNWithZScore
Evaluating RMSE of algorithm KNNWithZScore on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    8.2492  19.9170 7.8457  12.0040 5.5978
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00

KNNBaseline
Evaluating RMSE of algorithm KNNBaseline on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    9.7412  20.2007 9.1699  13.0372 5.0707
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.01    0.01    0.00    0.00    0.00

SVD
Evaluating RMSE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    5.5974  20.4205 6.8199  10.9460 6.7181
Fit time          0.01    0.01    0.00    0.01    0.00
Test time         0.00    0.00    0.00    0.00    0.00

SVDpp
Evaluating RMSE of algorithm SVDpp on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    6.2230  18.3028 7.5964  10.7074 5.3999
Fit time          0.02    0.02    0.01    0.01    0.00
Test time         0.01    0.01    0.00    0.01    0.00

NMF
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    7.8043  18.1617 10.3305 12.0988 4.4094
Fit time          0.01    0.01    0.00    0.01    0.00
Test time         0.00    0.00    0.00    0.00    0.00
```

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    7.8125  19.3233 11.5555 12.8971 4.7941
Fit time          0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00

CoClustering
Evaluating RMSE of algorithm CoClustering on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    7.6448  22.3942 13.7018 14.5802 6.0534
Fit time          0.02    0.02    0.01    0.01    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#
ranking = pd.DataFrame(benchmark)
ranking.sort_values('test_rmse', ascending = False)









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

[31]:

	test_rmse	fit_time	test_time	Algorithm
1	16.971962	0.000502	0.002812	NormalPredictor
10	14.580241	0.014315	0.001431	CoClustering
2	13.226660	0.000245	0.008039	KNNBasic
5	13.037234	0.000605	0.004833	KNNBaseline
9	12.897076	0.000684	0.002578	SlopeOne
3	12.782938	0.001471	0.003522	KNNWithMeans
0	12.242474	0.001007	0.003336	BaselineOnly
8	12.098817	0.006896	0.001353	NMF
4	12.003980	0.003490	0.003889	KNNWithZScore

6	10.945954	0.005468	0.001522	SVD
7	10.707385	0.013948	0.008441	SVDpp

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.

[33]:
```
#YOUR CODE STARTS HERE#
kf = KFold(n_splits=5, random_state=42)

param_distributions = {
# parameters for method
'n_factors' : range(5,35),
'n_epochs': range(30,60)
}

rs = RandomizedSearchCV(SVDpp, param_distributions=param_distributions,␣
 ↪n_iter=5, measures=['rmse'], cv=kf, n_jobs=2,
                        joblib_verbose=1000)
rs.fit(data)

print()
# best RMSE score
print("BEST_SCORE: " + str(rs.best_score['rmse']))

# combination of parameters that gave the best RMSE score
print()
print("BEST_PARAMETERS: ")
print(rs.best_params['rmse'])
print()

print()
```

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

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:    0.2s
[Parallel(n_jobs=2)]: Batch computation too fast (0.1652s.) Setting
batch_size=2.
[Parallel(n_jobs=2)]: Done   2 tasks      | elapsed:    0.2s
[Parallel(n_jobs=2)]: Done   3 tasks      | elapsed:    0.4s
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:    0.4s
[Parallel(n_jobs=2)]: Done   6 tasks      | elapsed:    0.7s
[Parallel(n_jobs=2)]: Done   8 tasks      | elapsed:    0.7s
[Parallel(n_jobs=2)]: Done  10 tasks      | elapsed:    0.9s
[Parallel(n_jobs=2)]: Done  12 tasks      | elapsed:    1.1s
[Parallel(n_jobs=2)]: Done  14 tasks      | elapsed:    1.3s
[Parallel(n_jobs=2)]: Done  16 tasks      | elapsed:    1.4s
[Parallel(n_jobs=2)]: Done  18 tasks      | elapsed:    1.4s
[Parallel(n_jobs=2)]: Done  20 tasks      | elapsed:    1.5s
[Parallel(n_jobs=2)]: Done  22 out of  25 | elapsed:    1.9s remaining:    0.3s
[Parallel(n_jobs=2)]: Done  25 out of  25 | elapsed:    2.0s remaining:    0.0s
[Parallel(n_jobs=2)]: Done  25 out of  25 | elapsed:    2.0s finished

BEST_SCORE: 9.991197511584094

BEST_PARAMETERS:
{'n_factors': 28, 'n_epochs': 58}
```

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———-

Knowing that the items are divided in two groups, I_A and I_B and users can like (rating 1) all items in group I_A and dislike those in group I_B (or viceversa), if a new user u is added and we record his preference of an itam a of group I_A (rating 1) thus we also know that he can just rate 0 the item b of group I_B.

If the user adds that they do not like an item j of group I_B, the rating above will not change.