DMT2023_HW2

April 20, 2023

0.1 Group composition:

————YOUR TEXT STARTS HERE————

Mignella, Laura, 1920520

Vestini, Maria Vittoria, 1795724

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 scikit-network==0.28.3
import pandas as pd
import numpy as np
from sknetwork.ranking import PageRank
from sknetwork.utils import get_neighbors
from IPython.display import display



#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.

[50]:
```
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
```

[51]:
```
#YOUR CODE STARTS HERE#
#Extract all the chapters' titles and texts from the two volumes


# Extract the titles and texts from both the volumes using the parse_html␣
 ↪function
titles, texts = parse_html(vol1)
titles_1, texts_1 = parse_html(vol2)

# Append together the lists
titles = titles + titles_1
```

```
texts = texts + texts_1

#Transform the list into a pandas DataFrame.

# Create the dataframe with columns titles and texts
data_frame_book = pd.DataFrame(data = {'titles':titles, 'texts': texts})


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

[52]:
```
#YOUR CODE STARTS HERE#


# Show only the first 8 rows of the dataframe
data_frame_book.head(8)



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

[52]:
```
                                                titles  \
0  first , how Uther Pendragon sent for the duke …
1  how Uther Pendragon made war on the duke of Co…
2     of the birth of King Arthur and of his nurture
3                of the death of King Uther Pendragon
4  how Arthur was chosen king , and of wonders an…
5  how King Arthur pulled out the sword divers times
6  how King Arthur was crowned , and how he made …
7  how King Arthur held in Wales , at a Pentecost…

                                                texts
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.

[53]:
```
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"])
```

[54]:
```
#YOUR CODE STARTS HERE#
#Extract all characters' names


# Iterate over the titles column
for title in data_frame_book.titles:
    # Extract all characters' names using the function␣
 ↪"extract_character_names_from_string"
    # And save everything in the all_characters set using the "union" method of␣
 ↪the sets
    all_characters = all_characters.
 ↪union(extract_character_names_from_string(title))
```

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

[55]:
```
#YOUR CODE STARTS HERE#

# Iterate over the characters names
for name in all_characters:
    # Print the name if the string "King" is in it
    if 'King' in name:
        print(name)

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

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

[56]:
```
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.

[57]:
```
#YOUR CODE STARTS HERE#
# Here we will create a dictionary that we will use to map the characters to␣
 ↪the name that we will use for that character in the data frame
names = {}
single_names = all_characters.difference(set().union(*disambiguate_sets))
for s in disambiguate_sets + list(single_names): # For each character
  if type(s) is set: # If it has multiple possible names
    l = list(s)
    for name in l:
      names[name]=l[0] # We map every name in the first
  else: # If it has only one possible name
    names[s]=s # We map it on itself

couples = set()
# To extract the couples of charcters in a chapter, we take the corpus of each␣
 ↪chapter
for corpus in data_frame_book.texts:
  # We extract all the names in the chapter
  clean_corpus = ' '.join([x for x in corpus.split(' ') if len(x) > 0])
  chars = set([names[x] for x in␣
 ↪extract_character_names_from_string(clean_corpus).
 ↪intersection(all_characters)])
  # We take all the characters in the corpus
  for char1 in chars:
    for char2 in chars:
      # And create a tuple that contains all the possible couples of characters
      if char1 < char2:
        couples.add((char1, char2))
      elif char2 < char1:
        couples.add((char2, char1))
# Once we have all the couples we can create the dataframe
```

```
PR_df = pd.DataFrame(data = {'character_1': [x[0] for x in couples],␣
 ↪'character_2' : [x[1] for x in couples]})
#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

[58]:
```
#YOUR CODE STARTS HERE#
r = []
for i, row in PR_df.iterrows():
  # Check if the name that represent 'Sir Lamorak' is the row
  if names['Sir Lamorak'] == row['character_1'] or names['Sir Lamorak'] ==␣
 ↪row['character_2']:
    r.append(i)

PR_df.iloc[r]
#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[58]:

	character_1	character_2
53	Sir Lamorak	Sir Urre
57	Sir Lamorak	Sir Sagramore le Desirous
61	Corsabrin	Sir Lamorak
101	Sir Lamorak	Winchester
121	Gaheris	Sir Lamorak
…	…	…
3305	King Bagdemagus	Sir Lamorak
3306	Saracens	Sir Lamorak
3307	Sir Lamorak	Sir Percivale
3343	Sir Bors	Sir Lamorak
3359	Sir Lamorak	Surluse

[95 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.

[59]:
```
#YOUR CODE STARTS HERE#



# Select all the names in the dataframe, get rid of the duplicates and sort
all_character_names =  sorted([x for x in set(list(PR_df['character_1']) +␣
 ↪list(PR_df['character_2']))])

# Print the names
print(' Sorted list: \n', all_character_names,
      '\n\n Lenght of the list:', len(all_character_names))









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

```
 Sorted list:
 ['Abbot', 'Alice', 'Alisander', 'Almaine', 'Almesbury', 'Andred', 'Anglides',
'Archbishop of Canterbury', 'Avoutres', 'Balan', 'Balin', 'Beale Isoud', 'Beale
Pilgrim', 'Benwick', 'Boudwin', 'Bragwaine', 'Camelot', 'Carbonek', 'Carlion',
'Castle of Maidens', 'Castle of Pendragon', 'Chapel Perilous', 'Christmas',
'Constantine', 'Corsabrin', 'Court', 'Dame Brisen', 'Damosel of the Lake',
'Dinadan', 'Dover', 'Elaine', 'Epinogris', 'Excalibur', 'Fair Maid of Astolat',
'Feast of Pentecost', 'Forest Perilous', 'France', 'Gaheris', 'Garlon', 'God',
'Gouvernail', 'Griflet', 'Helin le Blank', 'Humber', 'Igraine', 'Isle',
'Joseph', 'Joyous Gard', 'Kehydius', 'King', 'King Anguish of Ireland', 'King
Bagdemagus', 'King Ban', 'King Bors', 'King Brandegore', 'King Evelake', 'King
Howel of Brittany', 'King Lot of Orkney', 'King Mark', 'King Mordrains', 'King
Pellam', 'King Pelles', 'King Pellinore', 'King Rience', 'King Solomon', 'King
Uriens', 'Knight of the Red Launds', 'La Cote Male Taile', 'Lady Ettard', 'Lady
Lionesse', 'Lady of the Lake', 'Lambegus', 'Leodegrance', 'Logris', 'Lonazep',
'Lucius', 'Maimed King', 'Maledisant', 'May-day', 'Melias', 'Merlin', 'Nero',
'Our Lord', 'Palamides', 'Pope', 'Queen Guenever', 'Queen Isoud', 'Queen Morgan
le Fay', 'Queen of Orkney', 'Questing Beast', 'Red Knight', 'Romans', 'Rome',
'Round Table', 'Sangreal', 'Saracens', 'Siege Perilous', 'Sir Accolon of Gaul',
'Sir Aglovale', 'Sir Agravaine', 'Sir Alisander', 'Sir Amant', '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
Ector', 'Sir Elias', 'Sir Frol', 'Sir Galahad', 'Sir Galahalt', 'Sir Galihodin',
'Sir Gareth', 'Sir Gawaine', 'Sir Kay', 'Sir Lamorak', 'Sir Lancelot', 'Sir
Lanceor', 'Sir Lavaine', 'Sir Lionel', 'Sir Mador', 'Sir Marhaus', 'Sir
Meliagaunce', 'Sir Meliagrance', 'Sir Mordred', 'Sir Nabon', 'Sir Palomides',
'Sir Pedivere', 'Sir Pelleas', 'Sir Percivale', 'Sir Persant', 'Sir Sadok', 'Sir
Safere', 'Sir Sagramore le Desirous', 'Sir Segwarides', 'Sir Suppinabiles', 'Sir
Tor', 'Sir Tristram', 'Sir Turquine', 'Sir Urre', 'Sir Uwaine', 'Surluse',
'Tintagil', 'Ulfius', 'Uther Pendragon', 'Wales', 'Winchester', 'York']

 Lenght of the list: 159
```

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).

[60]:
```
#YOUR CODE STARTS HERE#

# Initialize the matrix with all zeros
adj_matrix = np.zeros((len(all_character_names),len(all_character_names)))

# And for each row of the matrix
for _, row in PR_df.iterrows():
  # Select the index of the characters
  i = all_character_names.index(row['character_1'])
  j = all_character_names.index(row['character_2'])
  # And set to one the coefficients of the matrix that represent the edge␣
 ↪between the two characters
  adj_matrix[i,j] = 1
  # We want the graph to be undirected, so the matrix has to be symmetric
  adj_matrix[j,i] = 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.

[61]:
```
#YOUR CODE STARTS HERE#

# Define the damping factor
damping_factor = 0.85
alpha = 1 - damping_factor


# Create the pagerank method with the wanted parameters
pagerank = PageRank(damping_factor=damping_factor, solver="piteration",
                    n_iter=1000, tol=10**-6)

# And apply the method to our graph
pagerank_vector = pagerank.fit_transform(adj_matrix)
```

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

[62]:
```
#YOUR CODE STARTS HERE#
# Select the top 15 characters by sorting the vector of results
k = 15
top_15 = sorted(list(zip(all_character_names, pagerank_vector)),
       key=lambda a: a[1], reverse = True)[0:k]

# show the top 15
top_15
#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[62]:
```
[('King', 0.022371021408497945),
 ('Sir Lancelot', 0.02117811094623829),
 ('God', 0.020621689323674385),
 ('Sir Tristram', 0.01745979495661758),
 ('Sir Gawaine', 0.01660446077157346),
 ('Queen Guenever', 0.016038743318161894),
 ('King Mark', 0.015221158601822373),
 ('Round Table', 0.014680265077684877),
 ('King Anguish of Ireland', 0.013396436757061966),
 ('Sir Lamorak', 0.013241375931287621),
 ('Sir Bors', 0.013009876115596454),
 ('Sir Kay', 0.012922154783244356),
 ('Sir Mordred', 0.01287067178130469),
 ('Sir Palomides', 0.012463669501149152),
 ('Sir Galahad', 0.012372857279072046)]
```

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`.

[63]:
```
from scipy.cluster.hierarchy import weighted
#YOUR CODE STARTS HERE#
damping_factor = 0.75 # Define the damping factor to then create the method␣
 ↪with the wanted parameters
pagerank = PageRank(damping_factor=damping_factor, solver="piteration",␣
 ↪n_iter=1000, tol=10**-6)
# Look for all the queens and count how many of them there are
queens = set()
count = 0
for name in all_characters:
  if names[name] in set(all_character_names).difference(queens) and 'Queen' in␣
 ↪name.split()[0]:
    count+=1
    queens.add(names[name])
# Set the weights of the nodes for the transport probability: if queen ->␣
 ↪weight = 1/#queens else 0
weights = {}
for i, name in enumerate(all_character_names):
  weights[i] = 0
  if name in queens: weights[i] = 1./count
# Apply the pagerank
pagerank_vector = pagerank.fit_transform(adj_matrix, seeds=weights)
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[64]:
```
#YOUR CODE STARTS HERE#
# Select the top 16 characters by sorting the vector of results
k = 16
top_16 = sorted(list(zip(all_character_names, pagerank_vector)),
       key=lambda a: a[1], reverse = True)[0:k]

# show the top 16
top_16
#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

[64]:
```
[('Queen Guenever', 0.06386317383402261),
 ('Queen Morgan le Fay', 0.058735726143593535),
 ('Queen Isoud', 0.05447450753414415),
 ('Queen of Orkney', 0.053713961791015105),
 ('Igraine', 0.053585004978319146),
 ('King', 0.017005292846005753),
```

```
('Sir Lancelot', 0.016357887069491015),
('God', 0.015998821319430104),
('Sir Gawaine', 0.014082359439138928),
('King Mark', 0.013609321793203042),
('Sir Tristram', 0.013142106554640303),
('King Anguish of Ireland', 0.012298532711595402),
('Round Table', 0.011603895748750308),
('Sir Kay', 0.011316660024553303),
('Gaheris', 0.010865382927156551),
('Sir Lamorak', 0.010792566515075411)]
```

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`.

[65]:
```
#YOUR CODE STARTS HERE#

# Define the damping factor to then create the method with the wanted parameters
damping_factor = 0.2
pagerank = PageRank(damping_factor=damping_factor, solver="piteration",␣
 ↪n_iter=1000, tol=10**-6)

# Look for all the knights
knights = set()
count = 0
for name in all_characters:
  if names[name] in set(all_character_names).difference(knights) and 'Sir' in␣
 ↪name.split()[0]:
    count+=1
    knights.add(names[name])

# Apply the Pagerank for each of the knights
pagerank_vectors = []
for kinght in knights:
  # The teleportation weights
  weights = {}
  # That will be 1 just for the knight we are considering in that moment
  weights[all_character_names.index(kinght)] = 1
  # Save the result of all the personalized pageranks
  pagerank_vectors.append((kinght, pagerank.fit_transform(adj_matrix,␣
 ↪seeds=weights)))




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

[66]:
```
#YOUR CODE STARTS HERE#


k = 2
# For each of the personalized pageranks
for pv in pagerank_vectors:
  # Print the name of the kinght and the result of the pagerank
  print(pv[0], '\n', sorted(list(zip(all_character_names, pv[1])),
       key=lambda a: a[1], reverse = True)[0:k], '\n')
```

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

```
Sir Belliance
 [('Sir Belliance', 0.8008607940818672), ('King', 0.0330754744405072)]

Sir Lamorak
 [('Sir Lamorak', 0.801048283103955), ('King', 0.0027812577330679094)]

Sir Accolon of Gaul
 [('Sir Accolon of Gaul', 0.8005601140602714), ('King', 0.010135950815895325)]

Dinadan
 [('Dinadan', 0.800791042534658), ('King', 0.0028308916455675385)]

Sir Suppinabiles
 [('Sir Suppinabiles', 0.8005798280093401), ('King', 0.023645519134523174)]

Sir Alisander
 [('Sir Alisander', 0.8008169241628564), ('King', 0.007629021304784188)]

Sir Lancelot
 [('Sir Lancelot', 0.8015592779391585), ('King', 0.002494118022031526)]

Sir Mador
 [('Sir Mador', 0.8005726808363692), ('King', 0.003524391934856867)]

Sir Brian
 [('Sir Brian', 0.8006512556422246), ('King', 0.008060950970066039)]

Sir Frol
 [('Sir Frol', 0.8013516277200265), ('King', 0.017539886853954068)]

Sir Bors
 [('Sir Bors', 0.8009524170535292), ('God', 0.0026793662482931654)]

Gaheris
 [('Gaheris', 0.8007980443243505), ('King', 0.002634913628815058)]
```

```
Sir Palomides
 [('Sir Palomides', 0.8009444633111121), ('Sir Lancelot',
0.0027518585785190485)]

Sir Tristram
 [('Sir Tristram', 0.8013431106298161), ('Sir Lancelot', 0.0026193443173855063)]

Sir Lanceor
 [('Sir Lanceor', 0.8004745863057798), ('King', 0.011322972008969572)]

Sir Amant
 [('Sir Amant', 0.8004166125288041), ('King', 0.015144128843161226)]

Sir Nabon
 [('Sir Nabon', 0.8010565611556747), ('King', 0.01266131081139842)]

Sir Elias
 [('Sir Elias', 0.800390495009646), ('King', 0.027256007006888353)]

Sir Galahalt
 [('Sir Galahalt', 0.8005797759532575), ('King', 0.007399278790086142)]

Sir Galahad
 [('Sir Galahad', 0.8009994095729441), ('God', 0.002785418432454099)]

Sir Pelleas
 [('Sir Pelleas', 0.8006289155013961), ('King', 0.00321412160749255)]

Sir Breunor
 [('Sir Breunor', 0.8004022523545612), ('Sir Lancelot', 0.02725863683055151)]

Sir Agravaine
 [('Sir Agravaine', 0.8007064047261943), ('King', 0.00303576624361697)]

Sir Carados
 [('Sir Carados', 0.8005788217695715), ('King', 0.0034754366784941573)]

Sir Sagramore le Desirous
 [('Sir Sagramore le Desirous', 0.8006203540434872), ('King',
0.003053006366023786)]

Sir Urre
 [('Sir Urre', 0.8005806307721638), ('King', 0.003532798886655961)]

Sir Lavaine
 [('Sir Lavaine', 0.8006328983736872), ('King', 0.004225939829037803)]
```

```
Sir Kay
 [('Sir Kay', 0.8008766411653517), ('King', 0.0025922952686297197)]

Sir Mordred
 [('Sir Mordred', 0.8009472502199051), ('King', 0.0027040345520742788)]

Sir Gareth
 [('Sir Gareth', 0.8007322717873805), ('King', 0.0028785143522568855)]

Epinogris
 [('Epinogris', 0.8005585407182026), ('King', 0.0035621283570752 62)]

Sir Bliant
 [('Sir Bliant', 0.8004536637757023), ('God', 0.023478595969928802)]

Sir Segwarides
 [('Sir Segwarides', 0.8006338844789459), ('King', 0.003940445632432037)]

Sir Archade
 [('Sir Archade', 0.8002890943944528), ('Sir Lancelot', 0.05379922660028132)]

Sir Colgrevance
 [('Sir Colgrevance', 0.8005170755129534), ('King', 0.003956670510573554)]

Sir Meliagaunce
 [('Sir Meliagaunce', 0.8004295183620493), ('King', 0.013942527394610237)]

Sir Aglovale
 [('Sir Aglovale', 0.8006345390414795), ('King', 0.0032571977957174 63)]

Sir Uwaine
 [('Sir Uwaine', 0.8007556648892579), ('King', 0.0029046894120706656)]

Sir Beaumains
 [('Sir Beaumains', 0.8005798952720696), ('King', 0.0036375428966975 78)]

Sir Dagonet
 [('Sir Dagonet', 0.8006211790036106), ('King', 0.007173483349863 78)]

Sir Persant
 [('Sir Persant', 0.8006236696329644), ('King', 0.0038610458194103 73)]

Sir Safere
 [('Sir Safere', 0.8006779861220782), ('King', 0.0031141414802272 95)]

Sir Tor
 [('Sir Tor', 0.8006181792489376), ('King', 0.003242662916978498)]
```

```
Sir Ector
 [('Sir Ector', 0.800917338531429), ('God', 0.0029474417303742336)]

Sir Bleoberis
 [('Sir Bleoberis', 0.8007484762611261), ('King', 0.0029280716620912624)]

Sir Turquine
 [('Sir Turquine', 0.8005288885573142), ('King', 0.004282808422207913)]

Sir Percivale
 [('Sir Percivale', 0.8009409495429156), ('God', 0.0028777049833220926)]

Sir Sadok
 [('Sir Sadok', 0.8007081975573034), ('King', 0.003068687943392694)]

Sir Galihodin
 [('Sir Galihodin', 0.800657097483308), ('King', 0.003053673541940632)]

Sir Breuse Saunce Pité
 [('Sir Breuse Saunce Pité', 0.8008260596946248), ('Sir Lancelot',
0.004661106982046734)]

Sir Lionel
 [('Sir Lionel', 0.8008279618569063), ('King', 0.0030052982032704 72)]

Sir Blamore
 [('Sir Blamore', 0.8005702568192157), ('King', 0.0062325803168876166)]

Sir Pedivere
 [('Sir Pedivere', 0.8006414797662187), ('King', 0.023686707238523206)]

Sir Bedivere
 [('Sir Bedivere', 0.8006741757503604), ('King', 0.0038534444736762715)]

Sir Gawaine
 [('Sir Gawaine', 0.8011520798156457), ('King', 0.0025080942918508506)]

Sir Meliagrance
 [('Sir Meliagrance', 0.8004620077747835), ('King', 0.0100318366301106)]

Sir Marhaus
 [('Sir Marhaus', 0.80076919670252), ('King', 0.004015827205528585)]

Sir Berluse
 [('Sir Berluse', 0.8003614581037668), ('King', 0.01508214687079243)]
```

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)

[67]:
```
#YOUR CODE STARTS HERE#

'''
Given that we already computed and stored the results of the personalized
pagerank for all the knights, now to find the Topic-Specific pagerank in an␣
 ↪"online"
context, we just need to take the results of the personalized pagerank of all␣
 ↪the
nodes of the needed topic, that was computed "offline", and compute an average␣
 ↪pagerank.
'''

knights_topic_vector = sum([x for _, x in pagerank_vectors])/
 ↪len(pagerank_vectors)









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

[68]:
```
#YOUR CODE STARTS HERE#
# Select the top 16 characters by sorting the vector of results
k = 8
top_8 = sorted(list(zip(all_character_names, knights_topic_vector)),
       key=lambda a: a[1], reverse = True)[0:k]
# Show the top 8
top_8

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

[68]:
```
[('Sir Lancelot', 0.02100054076499114),
 ('Sir Tristram', 0.01903913868031739),
 ('Sir Lamorak', 0.018416755693588673),
 ('Sir Gawaine', 0.018169670362941297),
 ('Sir Palomides', 0.017814579630685844),
```

```
('Sir Mordred', 0.016602206730111153),
('Dinadan', 0.016474830051371615),
('Gaheris', 0.016352375295125687)]
```

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———-

We know for the theory that the PageRank of a graph corresponds to the stationary distribution π that we obtain with: $\pi = \pi P$, where P is the matrix that define the probabilities to go from one node to another at any given moment.

$$P = (1-\alpha)P_{RW} + \frac{\alpha}{n}1_{n\cdot n} = \begin{bmatrix} \frac{\alpha}{n} & \frac{1-\alpha}{n-1}\cdot\frac{\alpha}{n} & \dots & \frac{1-\alpha}{n-1}\cdot\frac{\alpha}{n} \\ 1-\alpha+\frac{\alpha}{n} & \frac{\alpha}{n} & \dots & \frac{\alpha}{n} \\ \dots & & & \\ 1-\alpha+\frac{\alpha}{n} & \frac{\alpha}{n} & \dots & \frac{\alpha}{n} \end{bmatrix}$$

Given this matrix and the equation of the PageRank, we can find the formula for the PageRank of A by solving the first equation that we obtain by multipling the row vector $\pi = [\pi_A, \pi_1, ..., \pi_{n-1}]$ and the matrix P, and using the fact that $\sum_{i=1}^{n-1}\pi_i = 1 - \pi_A$ the equation is going to be:

$$\pi_A = \frac{\alpha}{n}\pi_A + (1-\pi_A)(1-\alpha+\frac{\alpha}{n})$$

By solving the equation we find that $\pi_A = \frac{n-\alpha n+\alpha}{2n-\alpha n}$ that depends on both the number of nodes and the dumping factor $1-\alpha$.

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 Dataset, Reader
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
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.

[70]:
```
#YOUR CODE STARTS HERE#

# Find which are the names that are used as representatives for the rulers and␣
 ↪knights
ruler = set([names[x] for x in all_characters if 'Queen' in x or 'King' in x])
knight = set([names[x] for x in all_characters if 'Sir' in x or 'Knight' in x])

couples = []
# For all the chapters' texts
for corpus in data_frame_book.texts:
  clean_corpus = ' '.join([x for x in corpus.split(' ') if len(x) > 0])
  # Find the names of all the characters in the corpus
  chars = set([names[x] for x in␣
 ↪extract_character_names_from_string(clean_corpus).
 ↪intersection(all_characters)])
  # And which of them are rulers or knight
  ruler_corpus = chars.intersection(ruler)
  knight_corpus = chars.intersection(knight)
  # Put all the possible couples of ruler-knight in tuples
  for char1 in ruler_corpus:
    for char2 in knight_corpus:
        couples.append((char1, char2))

# Create the DataFrame as of all the couples (the duplicates are still there)
RS_df = pd.DataFrame(data = {'Ruler': [x[0] for x in couples], 'Knight' : [x[1]␣
 ↪for x in couples]})

# To find the rating of a (ruler, knight) couple we have to count how many␣
 ↪times
# They appear together, so we groupby (ruler, knight).
RS_df = RS_df.groupby(['Ruler', 'Knight']).size().reset_index(name='Rating')


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

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.

[71]:
```
#YOUR CODE STARTS HERE#
# Find the min and max for all the columns (fields)
for column in RS_df.columns:
  print('Min and max of the field', column, ':')
  print('-', RS_df[column].min(), '\n-', RS_df[column].max())
# Find the rows of the knight 'Sir Accolon'
r = []
for i, row in RS_df.iterrows():
  if names['Sir Accolon'] == row['Knight']:
    r.append(i)
# And print those rows
RS_df.iloc[r]

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

```
Min and max of the field Ruler :
- Igraine
- Queen of Orkney
Min and max of the field Knight :
- Dinadan
- Sir Uwaine
Min and max of the field Rating :
- 1
- 201
```

[71]:

	Ruler	Knight	Rating
12	King	Sir Accolon of Gaul	10
220	King Lot of Orkney	Sir Accolon of Gaul	1
352	King Uriens	Sir Accolon of Gaul	5
397	Queen Guenever	Sir Accolon of Gaul	1
461	Queen Morgan le Fay	Sir Accolon of Gaul	9

2.1.3 2.1.3

Load the dataset into the appropriate scikit-surprise structure.

[72]:
```
#YOUR CODE STARTS HERE#

# Create the surprise structure
reader = Reader(rating_scale=(1,201))
data = Dataset.load_from_df(RS_df, 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.

[73]:
```
#YOUR CODE STARTS HERE#

# Create the k-fold object
kf = KFold(n_splits=3, random_state=24)




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

2.1.5 2.1.5

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

[74]:
```
#YOUR CODE STARTS HERE#

# Define a list with the 11 algorithms avaliable in the surprise library
algorithms  = [NormalPredictor,
               BaselineOnly,
               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
```

[75]:
```
#YOUR CODE STARTS HERE#

# Given the algorithms defined above, we will need 3 different parameters␣
 ↪configuarations
predictor_options = { # For the Baseline
  'method': "als",
  'name' : 'cosine',
  'user_based': False,
  'n_epochs' : 25
}

predictor_options_KNN = { # KNN
  'method': "als",
  'name' : 'cosine',
  'user_based': False
}

predictor_options_SVD = { # SVD and cluster
    'random_state' : 24,
    'n_epochs' : 25
}

# List of the configuaration options
pred_options = [predictor_options, predictor_options_KNN, predictor_options_SVD]
```

```
# List that we will use to map each algorithm to the right configuration (as␣
 ↪index of the pred_option list)
conf = [None, 0, 1, 1, 1, 3, 2, 2, 2, None, 2]


#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.

[76]:
```
#YOUR CODE STARTS HERE#

import multiprocessing

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


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

[76]: 2

[77]:
```
#YOUR CODE STARTS HERE#
results = []
# Select the right configuration for each method
for i, algo in enumerate(algorithms):
  if conf[i]==None: # No configuration needed
    results.append(cross_validate(algo(), data, measures=['MSE'], cv=kf,␣
 ↪verbose=False, n_jobs = cores))
  elif conf[i]==0: # Baseline configuaration
    current_algo = algo(pred_options[0], verbose=False)
    results.append(cross_validate(current_algo, data, measures=['MSE'], cv=kf,␣
 ↪verbose=False, n_jobs = cores))
  elif conf[i]==1: # KNN configuration
    current_algo = algo(sim_options=pred_options[1], verbose=False)
    results.append(cross_validate(current_algo, data, measures=['MSE'], cv=kf,␣
 ↪verbose=False, n_jobs = cores))
  elif conf[i]==2: # SVD and cluster configuration
    current_algo = algo(**pred_options[2], verbose=False)
    results.append(cross_validate(current_algo, data, measures=['MSE'], cv=kf,␣
 ↪verbose=False, n_jobs = cores))
  else: # KNNbaseline, it needs both the Baseline and the KNN configuaration
    current_algo = algo(sim_options=pred_options[1],␣
 ↪bsl_options=pred_options[0], verbose=False)
    results.append(cross_validate(current_algo, data, measures=['MSE'], cv=kf,␣
 ↪verbose=False, n_jobs = cores))
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

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.

[78]:
```
#YOUR CODE STARTS HERE#

rank = []
# Rewrite all the names of the algorithms to use them as label
algo_names  = ['NormalPredictor',
               'BaselineOnly',
               'KNNBasic',
               'KNNWithMeans',
               'KNNWithZScore',
               'KNNBaseline',
               'SVD',
               'SVDpp',
               'NMF',
               'SlopeOne',
               'CoClustering']

# For all the results find the mean of the test_mse
for algo, res in zip(algo_names, results):
  # And append it in a list together with the name of the agorithm
  rank.append((str(algo),np.mean(res['test_mse'])))

# Print the sorted results in descending orders (worst to best)
sorted(rank, key=lambda x: x[1], reverse = True)




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

[78]:
```
[('SVDpp', 12863.260195075147),
 ('NormalPredictor', 334.2337301276057),
 ('SlopeOne', 207.36073410996994),
 ('CoClustering', 195.51217837360778),
 ('KNNWithZScore', 190.4164669325418),
 ('KNNBasic', 183.32967952510748),
 ('BaselineOnly', 180.3352881218807),
 ('KNNBaseline', 177.64617589432893),
 ('KNNWithMeans', 175.37526862700872),
 ('SVD', 153.6774547627191),
```

```
('NMF', 98.54402940287844)]
```

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.

[79]:
```
#YOUR CODE STARTS HERE#

# Define the k-fold for k = 5
kf = KFold(n_splits=5, random_state=24)
# Define all the possible parameters configurations
grid_of_parameters = { # 3*2*2*2 = 24 possible configurations
    'random_state' : [24],
    'n_epochs': [5, 15, 30],
    'biased': [True, False],
    'reg_pu': [0.06, 0.1],
    'reg_qi': [0.06, 0.1]
}
# Create the method with the possible configurations
gs = GridSearchCV(NMF,
                  param_grid=grid_of_parameters,
                  measures=['rmse'],
                  cv=kf, n_jobs=cores
)
# And apply it to the data
gs.fit(data)
# Print the best score and parameters
print("Best Root Mean Squared Error: " + str(gs.best_score['rmse']), "\nBest␣
 ↪Parameters: " +str(gs.best_params['rmse']))

# Compute the MAE for both the default configuration and the new best␣
 ↪parameters
default = cross_validate(NMF(**pred_options[2]), data, measures=['mae'], cv=kf,␣
 ↪verbose=False)
```

```
new = cross_validate(NMF(**gs.best_params['rmse']), data, measures=['mae'],␣
 ↪cv=kf, verbose=False)
print("\nDefault configuration's Mean Absolute Error:", np.
 ↪mean(default['test_mae']),
      "\nNew configuration's Mean Absolute Error:", np.mean(new['test_mae']))
#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

```
Best Root Mean Squared Error: 7.363804060223044
Best Parameters: {'random_state': 24, 'n_epochs': 5, 'biased': False, 'reg_pu':
0.1, 'reg_qi': 0.1}

Default configuration's Mean Absolute Error: 3.3142751073096193
New configuration's Mean Absolute Error: 3.1748789751624775
```

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———-

When dealing with user-based collaborative filtering we can estimate the rating of a user u on an item i using the following:

\

$$R_{u,i} = \sum_{y \in N(u)} \frac{sim(y,u)R_{y,i}}{\sum_{y \in N(u)} sim(y,u)} = \begin{cases} \frac{min\{|U_a|,\ N\} \cdot sim(u_A,u)}{\sum_{y \in N(u)} sim(y,u)} \ if\ i \in I_A \\ \frac{max\{N - |U_a|,\ 0\} \cdot sim(u_B,u)}{\sum_{y \in N(u)} sim(y,u)} \ if\ i \in I_B \end{cases}$$

\ where $sim(u,y)$ is the cosine similarity, between user u and y, $N(u)$ is the set of the N users most similar to u, $u_A \in U_A$, $u_B \in U_B$ and the system is given by the fact that $R_{u_A,i_A} = 1,\ R_{u_A,i_B} = 0$ (opposite for u_B) and that the first rating of the user u makes it more similar to u_A than u_B.

\

$$If\ |U_A| \geq N : R_{u,i} = \begin{cases} \frac{N \cdot sim(u_A,u)}{\sum_{y \in N(u)} sim(y,u)} = 1\ if\ i \in I_A \\ \frac{0 \cdot sim(u_B,u)}{\sum_{y \in N(u)} sim(y,u)} = 0\ if\ i \in I_B \end{cases}$$

\ This because all the users in $N(u)$ are users of type A, this means that all the cosine similarities between u and $y \in N(u)$ is the same. For this reason we will have: $R_{u,i} = R_{y,i},\ y \in U_A$.

\

$$If\ |U_A| < N : R_{u,i} = \begin{cases} \frac{|U_A| \cdot sim(u_A,u)}{\sum_{y \in N(u)} sim(y,u)} \ if\ i \in I_A \\ \frac{(N - |U_A|) \cdot sim(u_B,u)}{\sum_{y \in N(u)} sim(y,u)} \ if\ i \in I_B \end{cases}$$

\ The users in $N(u)$ are both users of type A and of type B, so the estimated rating is going to be affected also by the reviews of the users B. After the first rating of the user u, $R_{u,i}$ is going to be a weighted average of $R_{y,i}$ so it will depend on both $|U_A|$ and the similarities, later, after the second

rating, we will compute again $R_{u,i}$ and this will be closer to the rating of U_A since $sim(u_A, u)$ should be grater that before.