DMT2023_HW3

May 19, 2023

0.1 Group composition:

————YOUR TEXT STARTS HERE————

Ben Kakitie, Joanna O., 1853548

Lauro, Francesco, 1706784

0.2 Homework 3

The homework consists of two parts:

1. Dimensionality Reduction

and

2. Supervised Learning

   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 `160`.

1 Part 1

In this part of the homework, you have to deal with Dimensionality Reduction.

[ ]:
```
#REMOVE_OUTPUT#
!pip install --upgrade --no-cache-dir gdown
from bs4 import BeautifulSoup
#YOUR CODE STARTS HERE#
from gensim import corpora          # gensim is used for Latent Semantic Analysis
from gensim.models import LsiModel
from gensim.models.coherencemodel import CoherenceModel

import nltk       # nltk is used to remove stopwords, perform stemming, etc.
nltk.download('stopwords')
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

import re       # Regex (re) is used to match patterns inside strings
import numpy as np      # numpy is used to handle arrays
import pandas as pd       # pandas is used to handle DataFrame
import matplotlib.pyplot as plt     # pyplot is used to create a figure and the axes in the figure
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

1.1 Part 1.1

The data you need to process comes from the book *Le Morte D'Arthur* by Thomas Malory.

You have to carry out Topic Modeling on book chapters.

The goal is to achieve a topic division within the following limits:

- The total computation may not exceed 10 minutes (starting from Part 1.1.5; Parts 1.1.1 to 1.1.4 are not considered for time calculation)

- The division into topics must be the “best one”

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]:
```python
#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]
#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.tail(8)





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

[5]:
```
    docno                                              title  \
495   496  how Sir Bedivere found him on the morrow dead …
496   497  of the opinion of some men of the death of Kin…
497   498  how when Sir Lancelot heard of the death of Ki…
498   499  how Sir Launcelot departed to seek the Queen G…
499   500  how Sir Launcelot came to the hermitage where …
500   501  how Sir Launcelot went with his seven fellows …
501   502  how Sir Launcelot began to sicken , and after …
502   503  how Sir Ector found Sir Launcelot his brother …

                                                  text
495  Then was Sir Bedivere glad , and thither he we…
496  yet some men say in many parts of England that…
```

497 And when he heard in his country that Sir Mord…
498 Then came Sir Bors de Ganis , and said : My lo…
499 But sithen I find you thus disposed , I ensure…
500 Then Sir Launcelot rose up or day , and told t…
501 Then Sir Launcelot never after ate but little …
502 And when Sir Ector heard such noise and light …

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 c in all_characters:
  if 'Sir' in c:
    print(c) #number of "Sir" characters = 67



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

```
Sir Segwarides
Sir Tristram de Liones
Sir Pedivere
Sir Galahad
Sir Agravaine
Sir Meliagrance
Sir Turquine
Sir Beaumains
Sir Sagramore le Desirous
Sir Nabon
Sir Colgrevance
Sir Amant
Sir Uriens
Sir Sadok
Sir Frol
Sir Mador
Sir Tristram
Sir Dagonet
Sir Gawaine
Sir Launcelot
Sir Lavaine
Sir Blamore
Sir Bedivere
Sir Pelleas
Sir Berluse
Sir Dinadan
Sir Brian
Sir Meliagaunce
Sir Persant of Inde
Sir Elias
Sir Belliance
Sir Tor
```

Sir Mordred
Sir Breunor
Sir Aglovale
Sir Urre
Sir Persant
Sir Bleoberis
Sir Carados
Sir Lionel
Sir Marhaus
Sir Breuse Saunce Pité
Sir Bliant
Sir Accolon
Sir Palomides
Sir Archade
Sir Lamorak de Galis
Sir Safere
Sir Pervivale
Sir Alisander
Sir Epinogris
Sir Suppinabiles
Sir Lancelot
Sir Uwaine
Sir Lamorak
Sir Gaheris
Sir Ector
Sir Bors
Sir Lanceor
Sir Anguish
Sir Malgrin
Sir Galahalt
Sir Kay
Sir Gareth
Sir Accolon of Gaul
Sir Percivale
Sir Galihodin

1.1.4 1.1.4

Preprocess the data

Consider only the titles

Each document must be a list of terms

Discard documents that have less than 10 (non-unique) words before the preprocessing (split by whitespace, ignore punctuation)

After preprocessing, each document must be represented by at least 5 tokens

- Several preprocessing options are possible

[9]:
```
#YOUR CODE STARTS HERE#

# title = [*title1, *title2]

title_dict = {}

for t in title:
  title_dict[t] = re.sub(r'[^\w\s]', '', t).split(' ')


title_dict = {k: v for k, v in title_dict.items() if len(set(v)) >= 10}   #use␣
 ↪set() method to check for unique values in a list




def preprocess_data(doc_set, token_min_length=1):
    en_stop = set(stopwords.words('english')) # create English stop words list
    p_stemmer = PorterStemmer() # Create p_stemmer of class PorterStemmer

    processed_tokenized_texts = []

    for text in doc_set: # loop through document list
        lowercase_text = text.lower()
        pattern = re.compile(r'[^a-z]+')
        cleaned_text = pattern.sub(' ', lowercase_text).strip() #Clean text:␣
 ↪replace pattern with space
        tokenized_text = cleaned_text.split(" ") #Divide text in tokens
        stopped_tokens = [token for token in tokenized_text if not token in␣
 ↪en_stop] # remove stop words from tokens
        if token_min_length>1:
          meaningful_tokens = [token for token in stopped_tokens if len(token)␣
 ↪>= token_min_length] # remove very small words, length < 3
        else:
          meaningful_tokens = stopped_tokens
```

```
        stemmed_tokens = [p_stemmer.stem(token) for token in meaningful_tokens]␣
 ↪# stem tokens
        processed_tokenized_texts.append(stemmed_tokens) # add tokens to list
    return processed_tokenized_texts


clean_docs = preprocess_data(title_dict, token_min_length=10)



#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

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

for t in title_dict.keys():
  if "Bedivere" in t:
    print(t)




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

```
how Sir Bedivere found him on the morrow dead in an hermitage , and how he abode
there with the hermit
```

1.1.5 1.1.5

Build a dictionary of the terms in the documents.

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




dictionary = corpora.Dictionary(clean_docs)












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

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


print({k: dictionary[k] for k in list(dictionary)[:5]})




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

```
{0: 'counsel', 1: 'leodegr', 2: 'enchant', 3: 'prophesi', 4: 'canterburi'}
```

1.1.6 1.1.6

Perform a document-term encoding of the dataset.

- Several encodings are possible

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


doc_term_matrix = [dictionary.doc2bow(doc) for doc in clean_docs]














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

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

# pd.DataFrame(doc_term_matrix).to_numpy()

doc_term_matrix



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

[14]:
```
[[],
 [],
 [],
 [],
 [],
 [],
 [(0, 1)],
 [],
 [],
 [],
```

```
[],
[(1, 1)],
[],
[],
[],
[],
[],
[(2, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(3, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(1, 1)],
[(4, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(5, 1)],
[],
[],
[],
[],
```

```
[],
[(6, 1), (7, 1)],
[(5, 1), (8, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(9, 1)],
[],
[],
[],
[],
[],
[],
[],
[(10, 1)],
[],
[],
[(11, 1)],
[],
[(12, 1)],
[(8, 1), (13, 1)],
[(8, 1)],
[],
[],
[(14, 1)],
[],
[],
[],
[],
[],
[],
[(2, 1)],
[],
[],
[(6, 1)],
[(15, 1)],
[],
[(16, 1)],
[],
[],
[],
```

```
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(17, 1)],
[],
[],
[],
[],
[],
[],
[],
[(18, 1)],
[],
[],
[],
[],
[],
[(15, 1), (19, 1), (20, 1)],
[],
[],
[],
[],
[(21, 1), (22, 1)],
[],
[],
[(23, 1)],
[(24, 1)],
[],
[(25, 1)],
[],
[],
[],
[],
```

```
[(15, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(24, 1)],
[],
[(22, 1)],
[],
[],
[],
[(26, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(27, 1)],
[],
[(28, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[(29, 1), (30, 1)],
[],
[],
[(5, 1)],
[],
[],
[(31, 1)],
[],
[],
```

```
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(15, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(15, 1)],
[],
[],
[],
[],
[],
[],
[],
[(9, 1)],
[],
[],
[],
[],
[],
[],
[(32, 1)],
[],
[],
[],
[(33, 1)],
```

```
[],
[],
[],
[],
[],
[(34, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(15, 1)],
[],
[],
[],
[],
[(15, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[(15, 1)],
[(35, 1)],
[],
[],
[],
[],
[],
[],
[(36, 1)],
[],
[(26, 1)],
[],
[],
[(37, 1)],
[],
[],
[],
```

```
[(15, 1), (26, 1)],
[(10, 1), (26, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(9, 1)],
[],
[(5, 1)],
[],
[],
[],
[],
[],
[],
[(7, 1)],
[],
[(5, 1), (8, 1)],
[],
[],
[(2, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(10, 1)],
[],
[(38, 1)],
[],
[(0, 1)],
[],
```

```
[],
[(39, 1)],
[],
[(8, 1)],
[],
[],
[],
[],
[(40, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[],
[(41, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[(8, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
```

```
[],
[(0, 1), (17, 1)],
[],
[(42, 1)],
[(42, 1)],
[],
[(15, 1)],
[],
[(8, 1)],
[(8, 1)],
[(8, 1)],
[(43, 1)],
[],
[],
[],
[],
[],
[],
[],
[],
[(5, 1)],
[(5, 1)],
[],
[],
[(5, 1)],
[],
[],
[(44, 1)],
[],
[(45, 1)],
[],
[],
[(46, 1)],
[(46, 1)],
[],
[],
[(47, 1)],
[],
[],
[],
[],
[],
[(48, 1)],
[],
[],
[(48, 1)],
[],
```

```
[],
[(49, 1)],
[],
[],
[],
[(50, 1)],
[(50, 1)],
[],
[(50, 1), (51, 1)],
[(50, 1)],
[(50, 1)],
[],
[(50, 1)],
[],
[],
[],
[],
[],
[],
[],
[(42, 1)],
[],
[],
[],
[],
[(48, 1)],
[],
[(17, 1)],
[],
[],
[],
[(52, 1)],
[(17, 1)],
[],
[],
[],
[],
[(34, 1)],
[],
[],
[],
[],
[(53, 1)],
[],
[],
[],
[],
```

```
[],
[(4, 1), (54, 1)],
[(55, 1)],
[],
[(56, 1)]]
```

1.1.7 1.1.7

Perform Latent Semantic Analysis for at least 5 different numbers of topics.

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


possible_numbers_of_topics = [2,3,4,5,6,8,10,12,15,20,25,30]  #12 different␣
 ↪numbers of topics
lsa_model_results = []

for number_of_topics in possible_numbers_of_topics:
  # print("LSA for #topics:",number_of_topics)
  lsa_model = LsiModel(doc_term_matrix, num_topics=number_of_topics, id2word =␣
 ↪dictionary) # train model
  lsa_model_results.append(lsa_model)











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

1.1.8 1.1.8

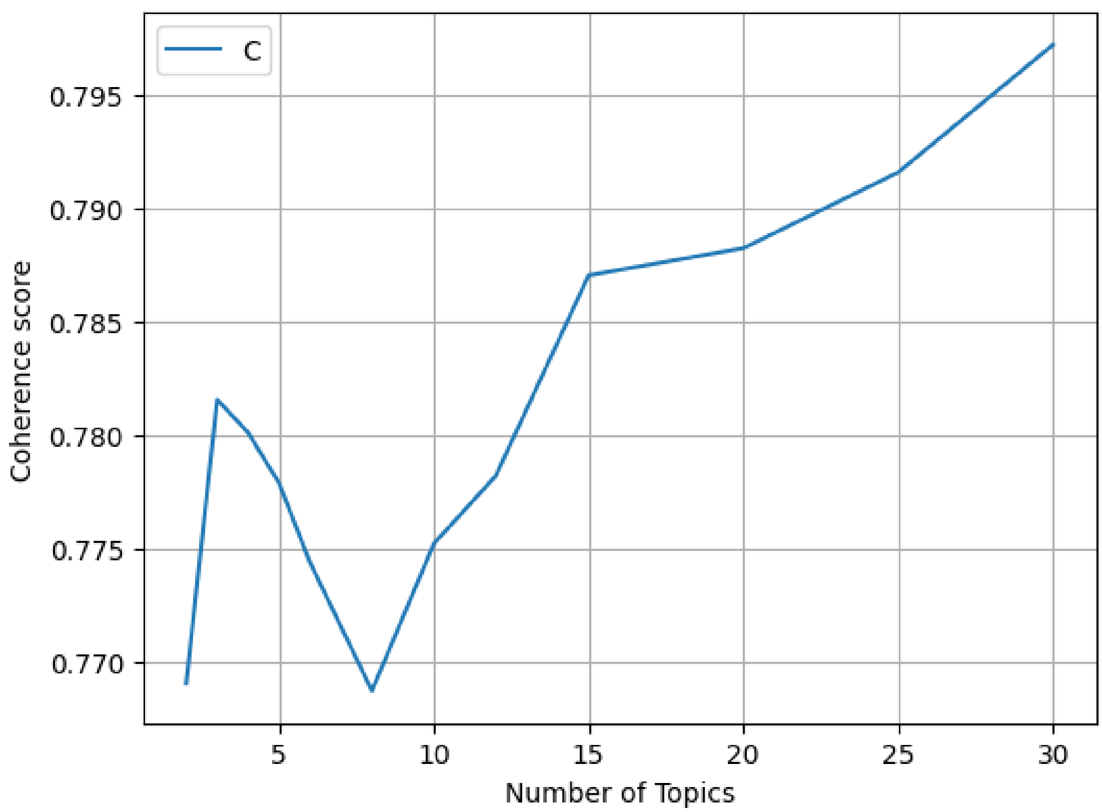
For each of the calculations above, calculate a measure of the "goodness" of the division into topics.

[16]:
```
#YOUR CODE STARTS HERE#

goodness = []      # coherence_values


for l in lsa_model_results:

  coherence_model = CoherenceModel(model=l, texts=clean_docs,␣
 ↪dictionary=dictionary, coherence='c_v')
  goodness.append(coherence_model.get_coherence())




for good in goodness:
  print(good)
```

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

```
0.7690239606374122
0.7815483452846879
0.7800924459218908
0.7778600852472717
0.7743602901478924
0.7687018921700913
0.7751931210152702
0.7781863828673852
0.7870327211939101
0.7882301835003277
0.7915759905393277
0.7972046636679728
```

[17]:
```
#YOUR CODE STARTS HERE#



plt.plot(possible_numbers_of_topics, goodness)
plt.xlabel("Number of Topics")
plt.ylabel("Coherence score")
plt.legend(("Coherence values"), loc='best')
plt.grid()
plt.show()









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



————-YOUR TEXT STARTS HERE————-

The “best” number of topics to model the dataset is 15, as the graph shows a logarithmic growth after this number of topics (graph “elbow”). From this number, the coherence score is proportional to the number of topics and it grows slower as the topics increase.

1.1.9 1.1.9

Print the 10 most important words for the 5 most important topics.

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

number_of_topics = 5


for topic_i, words_and_importance in lsa_model.
↪print_topics(num_topics=number_of_topics, num_words=10):
  print("TOPIC:",topic_i) #gives us the topic's id and the actual word

  for app in words_and_importance.split(" + "):
    value, token = app.split("*")
    value = float(value)
    token = str(token.replace('"',""))
    print("\t",value,token)

  print()



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

```
TOPIC: 0
         -0.793 marvel
         -0.604 adventur
         -0.082 exposit
         0.0 tournament
         -0.0 meliagr
         0.0 complain
         -0.0 enchant
         -0.0 tourney
         0.0 encount
         0.0 worship

TOPIC: 1
         -0.974 tournament
         -0.156 fellowship
         -0.116 worship
         -0.116 encount
         -0.021 tourney
         -0.0 marvel
         -0.0 adventur
         0.0 meliagr
         0.0 counsel
```

```
        0.0 colgrev

TOPIC: 2
        -0.796 adventur
        0.595 marvel
        0.108 exposit
        0.0 meliagr
        0.0 forgiv
        0.0 colgrev
        0.0 fellowship
        -0.0 complain
        -0.0 discharg
        0.0 bagdemagu

TOPIC: 3
        -0.982 meliagr
        -0.189 forgiv
        -0.0 adventur
        0.0 marvel
        0.0 exposit
        0.0 commun
        0.0 colgrev
        0.0 bagdemagu
        -0.0 tournament
        0.0 displeas

TOPIC: 4
        -0.851 commun
        -0.526 counsel
        0.0 fellowship
        0.0 tourney
        -0.0 tournament
        -0.0 worship
        -0.0 encount
        0.0 colgrev
        0.0 bagdemagu
        0.0 displeas
```

———-YOUR TEXT STARTS HERE———-

Throught the 'print_topics' function, we selected the 5 most important topics by their order of significance, which allowed us to print out the topics id, the float value and the token (i.e., the preprocessed word)

———-YOUR TEXT STARTS HERE———-

The top 5 topics obtained are the the most frequent topics in the documents corpus, as a document is about a particular topic. The following tokens are then ordered by their probability value based on their significance (from the highest to lowest value).

1.2 Part 1.2

1.2.1 1.2.1

Suppose you have a dataset with N samples and M features.

You only have B units of memory available on your storage medium.

Assume further that each feature occupies a constant number b of memory units and that this cannot be changed (e.g. you cannot change the precision of floats).

Assuming that the entire dataset cannot fit on your storage medium, how would you accommodate all N samples while retaining as much information about your data as possible?

Use at most 3 sentences.

———-YOUR TEXT STARTS HERE———-

To accommodate all N samples while retaining as much information about the data as possible we could use the “random sampling” technique.

After randomly selecting a subset of the N samples that can fit into the available memory B, we can perform an analysis on this subset of data, which will give a general idea of the trends and patterns in the data.

Another approach is to use dimensionality reduction methods, like Principal Component Analysis (PCA), to reduce the number of features in the dataset, which can help to reduce the memory requirements of the dataset while retaining as much information as possible.

2 Part 2

In this part, your goal is to obtain the best classification on a dataset according to a metric specified in each section.

[ ]:
```
#REMOVE_OUTPUT#
#YOUR CODE STARTS HERE#
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
# from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import time, tqdm, sklearn
import matplotlib.pyplot as plt
#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

2.1 Part 2.1

In this part, you will perform a tf-idf encoding of the data, and then train a classifier, optimising its hyper-parameters.

In the various steps, we will slowly prepare a pipeline to perform a hyper-parameter optimisation; try to prepare the required objects with this target in mind.

The goal is to maximise the accuracy on the test set.

2.1.1 2.1.1

Prepare the dataset for Supervised Learning.

> It should be a Pandas DataFrame with two fields: `Text`, `Label`.
>
> The `Text` column must contain the text of a chapter
>
> The `Label` column must contain a value of 0 or 1

- The `Label` is 0 if the chapter is in Book 1
- The `Label` is 1 if the chapter is in Book 2

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



# title1, text1 = parse_html(vol1)
# title2, text2 = parse_html(vol2)
# text = [*text1, *text2]
```

```python

label1 = [0]*len(text1)
label2 = [1]*len(text2)
label = [*label1, *label2]
#print(label)

df_sl = pd.DataFrame(list(zip(text, label)), columns =['Text', 'Label'])















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

[21]:

```python
#YOUR CODE STARTS HERE#


print(df_sl.head(2))
print(df_sl.tail(2))








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

```
                                                Text  Label
0  It befell in the days of Uther Pendragon , whe…      0
1  Then Ulfius was glad , and rode on more than a…      0
                                                  Text  Label
501  Then Sir Launcelot never after ate but little …      1
502  And when Sir Ector heard such noise and light …      1
```

2.1.2 2.1.2

Divide the dataset into training (68%), validation (17%) and test set (15%).

[22]:
```
#YOUR CODE STARTS HERE#



x = df_sl['Text']
y = df_sl['Label']

train_val_x, test_x, train_val_y, test_y = train_test_split(x, y, test_size=0.
↪15, shuffle=True, random_state = 160)

train_x, val_x, train_y, val_y = train_test_split(train_val_x, train_val_y,␣
↪test_size=0.20, shuffle=True, random_state = 160)










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

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

print("The percentage of samples with negative labels (0) in the training set␣
↪is "+str(((len(train_y)-sum(train_y))*100)/len(train_y))+' %')
print("The percentage of samples with negative labels (0) in the validation set␣
↪is "+str(((len(val_y)-sum(val_y))*100)/len(val_y))+' %')
print("The percentage of samples with negative labels (0) in the test set is␣
↪"+str(((len(test_y)-sum(test_y))*100)/len(test_y))+' %')



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

```
The percentage of samples with negative labels (0) in the training set is
48.97360703812317 %
The percentage of samples with negative labels (0) in the validation set is
47.674418604651166 %
The percentage of samples with negative labels (0) in the test set is
39.473684210526315 %
```

2.1.3 2.1.3

Create an object that performs a tf-idf transformation on the data. The transformation must **NOT** lowercase character names.

Create a dictionary containing configurations for the tf-idf vectorizer. Each hyper-parameter should have exactly **3 values**.

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

vectorizer = TfidfVectorizer(lowercase= False, norm= None)

# TfidfVectorizer_parameters = {'vect__strip_accents': ['ascii', 'unicode',␣
 ↪None], 'vect__analyzer': ['word', 'char', 'char_wb'],
#                         'vect__ngram_range': [(1, 1), (2, 2), (3, 3)],␣
 ↪'vect__max_df': [0.9, 0.95, 1.0], 'vect__min_df': [1,2,3],
#                         'vect__norm': ['l1', 'l2', None]}

TfidfVectorizer_parameters = {'vect__analyzer': ['word', 'char', 'char_wb'],
                          'vect__ngram_range': [(1, 1), (2, 2), (3, 3)]}















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

2.1.4 2.1.4

Choose a maximum of 2 classification algorithms (from those seen during the course) and prepare objects containing them.

For each of the selected classification algorithms, prepare a hyper-parameter configuration.

Each configuration must vary **at least 4 different hyper-parameters**.

If a parameter is itself composed of several parameters (if it is a dictionary, for example), each of these must vary at least 4 different hyper-parameters.

[25]:
```
#YOUR CODE STARTS HERE#



clf1 = KNeighborsClassifier()
clf2 = SVC(random_state= 160)

# KNeighborsClassifier_parameters = {'clf__n_neighbors': [5, 10, 15, 20],␣
↪'clf__weights': ['uniform', 'distance'],
#                                       'clf__algorithm': ['auto', 'ball_tree',␣
↪'kd_tree', 'brute'],
#                                       'clf__metric': ['cityblock', 'cosine',␣
↪'euclidean', 'haversine', 'l1', 'l2', 'manhattan', 'nan_euclidean']}

KNeighborsClassifier_parameters = {'clf__n_neighbors': [5, 10, 15, 20],␣
↪'clf__weights': ['uniform', 'distance'],
                                    'clf__algorithm': ['auto', 'brute'],
                                    'clf__metric': ['cityblock', 'cosine',␣
↪'euclidean']}

# SVC_parameters = {'clf__C': [0.25, 0.5, 1., 2., 4.], 'clf__kernel':␣
↪['linear', 'poly', 'rbf', 'sigmoid', 'precomputed'],
#                   'clf__gamma': ['scale', 'auto'], 'clf__shrinking': [True,␣
↪False]}

SVC_parameters = {'clf__C': [0.5, 1., 2.], 'clf__kernel': ['linear', 'poly',␣
↪'rbf', 'sigmoid'],
                  'clf__gamma': ['scale', 'auto'], 'clf__shrinking': [True,␣
↪False]}











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

2.1.5 2.1.5

For each of the classification algorithms selected in step 2.1.4, perform a 5-fold Cross-Validation on the validation set, combining the configurations of the vectorizer defined in step 2.1.3 and those of the classifier being used defined in step 2.1.4.

Perform the best hyper-parameter optimisation you can afford in **LESS than 15 minutes**.

If you are using two classifications algorithms, the maximum total optimisation time is **INSTEAD** 30 minutes.

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

pipeline1 = Pipeline([
        ('vect', vectorizer),
        ('clf', clf1),
        ])
pipeline2 = Pipeline([
        ('vect', vectorizer),
        ('clf', clf2),
        ])

parameters_1 = TfidfVectorizer_parameters | KNeighborsClassifier_parameters
parameters_2 = TfidfVectorizer_parameters | SVC_parameters
parameters_dummy = {'vect__strip_accents': ['ascii', 'unicode', None],␣
 ↪'vect__analyzer': ['word', 'char', 'char_wb']}
parameters_dummy_2 = {'vect__strip_accents': ['ascii', 'unicode', None],␣
 ↪'vect__analyzer': ['word', 'char', 'char_wb'],
                      'clf__metric': ['cityblock', 'cosine', 'euclidean'],␣
 ↪'clf__algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']}
# grid_search1 = GridSearchCV(pipeline1, parameters_1, scoring = metrics.
 ↪make_scorer(metrics.matthews_corrcoef),
#␣
 ↪
 ↪= 5, n_jobs = -1, verbose = 10)
# grid_search2 = GridSearchCV(pipeline2, parameters_2, scoring = metrics.
 ↪make_scorer(metrics.matthews_corrcoef),
#␣
 ↪
 ↪ cv = 5, n_jobs = -1, verbose = 10)

grid_search1 = GridSearchCV(pipeline1, parameters_1, scoring = metrics.
 ↪make_scorer(metrics.accuracy_score), cv = 5, n_jobs = -1,
                            verbose = 10)
grid_search2 = GridSearchCV(pipeline2, parameters_2, scoring = metrics.
 ↪make_scorer(metrics.accuracy_score), cv = 5, n_jobs = -1,
                            verbose = 10)

start = time.time()

# grid_search1.fit(val_x[0:10], val_y[0:10])
# stop1 = time.time()
```

```
# grid_search2.fit(val_x[0:10], val_y[0:10])

grid_search1.fit(val_x, val_y)
# stop1 = time.time()
grid_search2.fit(val_x, val_y)

end = time.time()

#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
Fitting 5 folds for each of 432 candidates, totalling 2160 fits
Fitting 5 folds for each of 432 candidates, totalling 2160 fits
```

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



print(end-start)




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

```
886.6141264438629
```

2.1.6 2.1.6

For each of the optimisations run in step 2.1.5:

Select the 5 best configurations and print them.

[28]:
```
#YOUR CODE STARTS HERE#
results_mean_test_score_1, results_mean_test_score_2, results_params_1,␣
↪results_params_2 = [], [], [], []
results_test_score_sd_1, results_test_score_sd_2 = [], []
for i in range(len(grid_search1.cv_results_['params'])):
  results_mean_test_score_1.append(grid_search1.
↪cv_results_['mean_test_score'][i])
  results_test_score_sd_1.append(grid_search1.cv_results_['std_test_score'][i])
  results_params_1.append(grid_search1.cv_results_['params'][i])
df_1 = pd.DataFrame(list(zip(results_params_1, results_mean_test_score_1,␣
↪results_test_score_sd_1)),
                    columns =['configuration', 'mean_test_score',␣
↪'std_test_score'])
display("Top 5 configurations for KNeighborsClassifier:", df_1.
↪sort_values("mean_test_score", ascending=False, ignore_index=True).head(5))
for i in range(len(grid_search2.cv_results_['params'])):
  results_mean_test_score_2.append(grid_search2.
↪cv_results_['mean_test_score'][i])
  results_test_score_sd_2.append(grid_search2.cv_results_['std_test_score'][i])
  results_params_2.append(grid_search2.cv_results_['params'][i])
df_2 = pd.DataFrame(list(zip(results_params_2, results_mean_test_score_2,␣
↪results_test_score_sd_2)),
                    columns =['configuration', 'mean_test_score',␣
↪'std_test_score'])
print('\n')
display("Top 5 configurations for SVC:", df_2.sort_values("mean_test_score",␣
↪ascending=False, ignore_index=True).head(5))
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

```
'Top 5 configurations for KNeighborsClassifier:'

                                       configuration  mean_test_score  \
0  {'clf__algorithm': 'brute', 'clf__metric': 'co…         0.837255
1  {'clf__algorithm': 'auto', 'clf__metric': 'cos…         0.837255
2  {'clf__algorithm': 'auto', 'clf__metric': 'cos…         0.813725
3  {'clf__algorithm': 'brute', 'clf__metric': 'co…         0.813725
4  {'clf__algorithm': 'brute', 'clf__metric': 'co…         0.802614

   std_test_score
0        0.022866
1        0.022866
2        0.044713
```

```
3        0.044713
4        0.045358
```

```
'Top 5 configurations for SVC:'
```

```
                                       configuration  mean_test_score  \
0  {'clf__C': 1.0, 'clf__gamma': 'scale', 'clf__k…         0.859477
1  {'clf__C': 2.0, 'clf__gamma': 'auto', 'clf__ke…         0.859477
2  {'clf__C': 1.0, 'clf__gamma': 'scale', 'clf__k…         0.859477
3  {'clf__C': 2.0, 'clf__gamma': 'scale', 'clf__k…         0.859477
4  {'clf__C': 2.0, 'clf__gamma': 'scale', 'clf__k…         0.859477

   std_test_score
0        0.088658
1        0.088658
2        0.088658
3        0.088658
4        0.088658
```

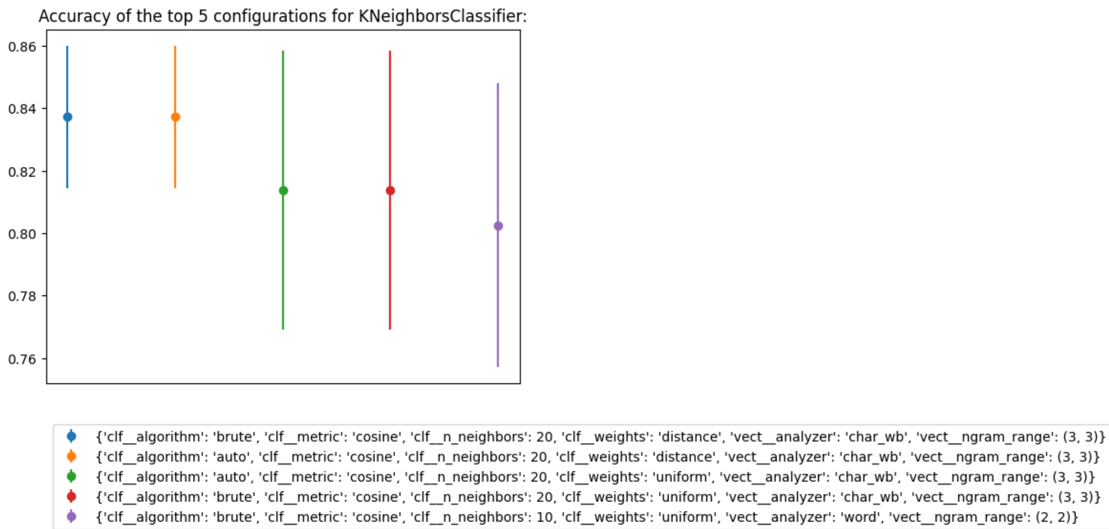
2.1.7 2.1.6

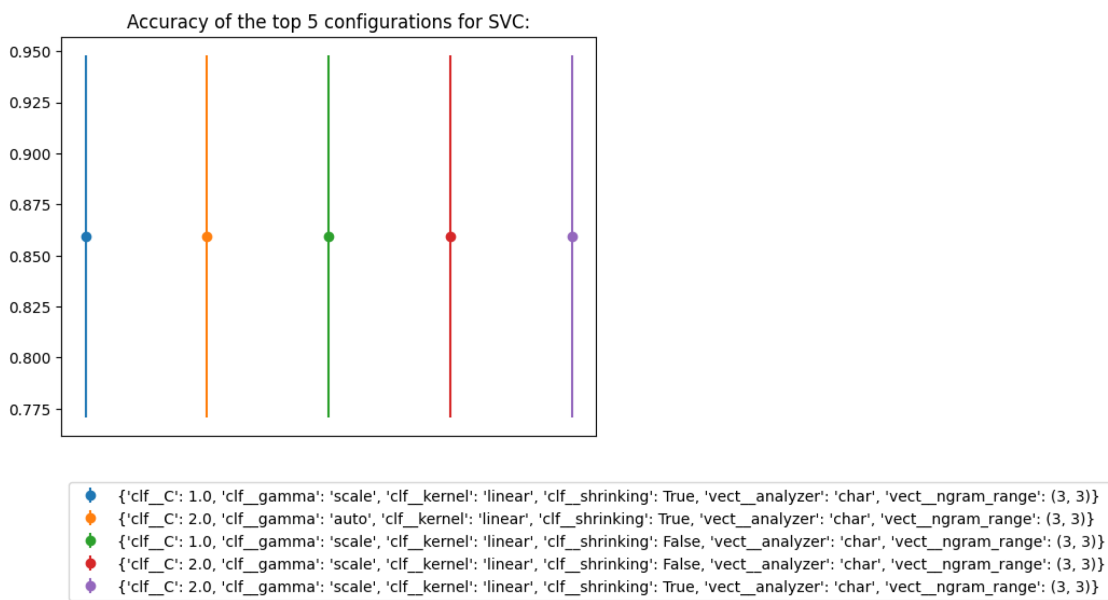
For each of the optimisations run in step 2.1.5:

> Produce a plot with mean and standard deviation of the accuracy calculated on the test set (**of each fold**) for the 5 configuration selected in step 2.1.6.

[29]:
```
#YOUR CODE STARTS HERE#
df_1_plot = df_1.sort_values("mean_test_score", ascending=False,␣
 ↪ignore_index=True).head(5)
df_2_plot = df_2.sort_values("mean_test_score", ascending=False,␣
 ↪ignore_index=True).head(5)
for i in range(5):
  plt.errorbar(df_1_plot.index[i], df_1_plot['mean_test_score'][i],␣
 ↪yerr=df_1_plot['std_test_score'][i], linestyle='None', fmt='o',
               label=df_1_plot['configuration'][i])
plt.legend(loc='upper left', bbox_to_anchor=(0,-0.1))
plt.title("Accuracy of the top 5 configurations for KNeighborsClassifier:")
plt.gca().xaxis.set_visible(False)
plt.show()
for i in range(5):
  plt.errorbar(df_2_plot.index[i], df_2_plot['mean_test_score'][i],␣
 ↪yerr=df_2_plot['std_test_score'][i], linestyle='None', fmt='o',
               label=df_2_plot['configuration'][i])

plt.legend(loc='upper left', bbox_to_anchor=(0,-0.1))
plt.title("Accuracy of the top 5 configurations for SVC:")
plt.gca().xaxis.set_visible(False)
```

```
plt.show()
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```





————-YOUR TEXT STARTS HERE————-

For the KNeighborsClassifier we chose the configuration corresponding to the first error bar from left (in addition to random_state= 160 for KNeighborsClassifier() and (lowercase= False, norm= None) for TfidfVectorizer()) because it's one of the two best configurations in terms of accuracy.

For the SVC we chose the configuration corresponding to the first error bar from left (in addition to lowercase= False and norm= None for TfidfVectorizer()) because it's one of the five best configurations in terms of accuracy.

2.1.8 2.1.8

For each of the optimisations, obtain a classifier using the parameters you selected in step 2.1.6.

[30]:
```
#YOUR CODE STARTS HERE#
best_params_1, best_params_2 = df_1_plot['configuration'][0],␣
↪df_2_plot['configuration'][0]
best_params_1_vect, best_params_1_clf, best_params_2_vect, best_params_2_clf =␣
↪{}, {}, {}, {}

for i in best_params_1.keys():
  if (i[0:4]=='vect'):
    best_params_1_vect[i[6:]] = best_params_1[i]
  else:
    best_params_1_clf[i[5:]] = best_params_1[i]

for i in best_params_2.keys():
  if (i[0:4]=='vect'):
    best_params_2_vect[i[6:]] = best_params_2[i]
  else:
    best_params_2_clf[i[5:]] = best_params_2[i]

vectorizer1 = TfidfVectorizer(lowercase= False, norm= None,␣
↪**best_params_1_vect)
vectorizer2 = TfidfVectorizer(lowercase= False, norm= None,␣
↪**best_params_2_vect)
clf1, clf2 = KNeighborsClassifier(**best_params_1_clf), SVC(random_state = 160,␣
↪**best_params_2_clf)

pipeline1 = Pipeline([
        ('vect', vectorizer1),
        ('clf', clf1),])
pipeline2 = Pipeline([
        ('vect', vectorizer2),
        ('clf', clf2),])
pipeline1.fit(train_val_x,train_val_y)
pipeline2.fit(train_val_x,train_val_y)
#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

[30]:
```
Pipeline(steps=[('vect',
                 TfidfVectorizer(analyzer='char', lowercase=False,
                                 ngram_range=(3, 3), norm=None)),
                ('clf', SVC(kernel='linear', random_state=160))])
```

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

pred_test_y_1 = pipeline1.predict(test_x)
```

```python
pred_test_y_2 = pipeline2.predict(test_x)


confusion_matrix_1 = metrics.confusion_matrix(test_y, pred_test_y_1)
confusion_matrix_2 = metrics.confusion_matrix(test_y, pred_test_y_2)
print("KNeighborsClassifier: True-Classes X Predicted-Classes \n", pd.
 ↪DataFrame(confusion_matrix_1), sep='')
print("SVC: True-Classes X Predicted-Classes \n", pd.
 ↪DataFrame(confusion_matrix_2), sep='')



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

```
KNeighborsClassifier: True-Classes X Predicted-Classes
    0   1
0  23   7
1   3  43
SVC: True-Classes X Predicted-Classes
    0   1
0  28   2
1   3  43
```

2.2 Part 2.2

2.2.1 2.2.1

You have a training set containing N documents. There are M_1 unique terms within the dataset.

The test dataset will have M_2 unique terms within it. However, we know that only a small amount of these will be in common with the training dataset.

What precautions could we use to preprocess the data?

What could we change at test time and which of the classification algorithms seen in class would best suit the change?

Use at most 4 sentences.

———-YOUR TEXT STARTS HERE———-

When preprocessing the data, we could apply lowercase conversion, stopwords removal stemming and/or lemmatization in order to reduce the amount of unique terms in M_1 and therefore increase the proportion of common unique terms between M_1 and M_2. At test time we can apply the same preprocessing techniques to the test set M_2, in order to further increase the proportion of common unique terms between M_1 and M_2. Another thing we could do at test time to deal with terms in M_2 not seen at training time is to simply ignore them and remove them from the test dataset. We can also use word embeddings, and in that case neural networks are best suited since they can use the similarity between embedding of words in M_2 but not in M_1 and embedding of words in M_1 but not in M_2.