DMT2023_HW1

March 30, 2023

0.1 Group composition:

————YOUR TEXT STARTS HERE————

Mignella, Laura, 1920520 \ Vestini, Maria Vittoria, 1795724

0.2 Homework 1

The homework consists of two parts:

1. Search-Engine Evaluation

and

2. Near-Duplicate-Detection

1 Part 1

In this part of the homework, you have to index collections of documents to build search-engines using the PyTerrier library.

Import **ALL** the Python packages that you need for Part 1.

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

!pip install python-terrier
import pyterrier as pt
if not pt.started():
  pt.init()
import pandas as pd
import matplotlib.pyplot as plt


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

1.1 Part 1.1

You have to build a search engine for the book *Le Morte D'Arthur* by Thomas Malory and **improve the search-engines performance** (the higher the better). The book is divided into two volumes. Each chapter is a document with two fields: title of the chapter and corpus of the chapter. You only want to index the corpus of each chapter.

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


# 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 (a PyTerrier friendly structure).
```

```
# Create the dataframe with columns titles and texts
data_frame_book = pd.DataFrame(data = {'docno': [str(x) for x in␣
 ↪range(len(titles))], 'titles':titles, 'texts': texts})


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

[5]:
```
#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#
```

[5]:

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

1.1.3 1.1.3

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

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

    remember = ""
    last_is_special_token = False

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

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

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



# 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#
```

[8]:
```
#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 Bagdemagus
King Lot of Orkney
King Uriens
King Pelleas
King Brandegore
King Arthur
King Mark of Cornwall
King Bors
King
King Mark
King Mordrains
King Pellam
King Solomon
King of England
King Pellinore
King Howel of Brittany
King Anguish of Ireland
King Pelles
Maimed King
King Leodegrance
King Lot
King Rience
King Evelake
King of the Land of Cameliard
King Ban
```

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",
                        "Lancelot": "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 topics for the queries.

Each character name (including alternative names) represents a topic.

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




# Dataframe where we will store the topics
topics = pd.DataFrame()
qid = []
query = []

# Iterate over the list of the sets containing the possible names of each␣
 ↪character
for id, different_names in enumerate(disambiguate_sets):
  # For each of the possible name
  for name in different_names:
    # We store its qid
    qid.append(str(id))
    # And the name
    query.append(name)

# Put the lists inside the dataframe
topics['qid'] = qid
topics['query'] = query




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

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


#  Show only the first 5 rows of the dataframe
topics.head(5)
```

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

[11]:

	qid	query
0	0	Sir Persant
1	0	Sir Persant of Inde
2	1	Sir Accolon
3	1	Sir Accolon of Gaul
4	1	Accolon

1.1.6 1.1.6

Prepare the relevance scores for the queries.

> A document is considered relevant (1) if its **corpus** contains the character's name or one of its alternative names, otherwise is not relevant (0).

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



# Define the dataframe to store the relevancy of the documents with respect to␣
 ↪the queries
qrels = pd.DataFrame(columns = ['qid','docno','label'])
relevant = {}

# Iterate over the tuples containing the id of the query and the query
for qid, name in zip(topics['qid'], topics['query']):
  # For each document we take its docno and corpus
  for docno, corpus in enumerate(data_frame_book.texts):
    # If we have yet to encounter the tuple (qid, docno)
    if (qid, str(docno)) not in relevant.keys():
      # We initialize the dictionary item, key = (qid, docno) and relevance = 0
      relevant[(qid, str(docno))] = 0
    # If the query is in the corpus of the document
    if name in corpus:
      # We set its relevance to 1
      relevant[(qid, str(docno))] = 1
      continue

# Insert all the qid, docno and relevance in the dataframe
for k, v in relevant.items():
  qrels = pd.concat([qrels, pd.DataFrame([[k[0], k[1], v]], columns = qrels.
 ↪columns)], ignore_index=True)



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

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

# Show the first and the last rows
print('First and last row:')
display(qrels.iloc[[0,-1]])
print('\nShape of the DataFrame:\t', qrels.shape)
```

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

```
First and last row:
```

	qid	docno	label
0	0	0	0
22131	43	502	1

```
Shape of the DataFrame:  (22132, 3)
```

1.1.7 1.1.7

Choose several preprocessing configurations (at least 2, no more than 4).

For each of them, construct an index on the `title` field.

For the last of them, report the number of indexed documents and terms.

[14]:
```
#YOUR CODE STARTS HERE#
# We will use the function 'create_index' seen during the lab1
def create_index(preprocessing1, preprocessing2, field, count):
  pd_indexer = pt.DFIndexer("./Inverted_Index" + str(count), overwrite=True,␣
 ↪stemmer=preprocessing1, stopwords=preprocessing2)
  indexref = pd_indexer.index(data_frame_book[field], data_frame_book["docno"])
  return indexref

# Preprocessing configurations that we will use
preprocessing1 = [None , None, "EnglishSnowballStemmer",␣
 ↪"EnglishSnowballStemmer"]
preprocessing2 = [None , "Stopwords" , None, "Stopwords"]
indexer = []
# We will save the indexer for each preprocessing configuration
for i, preprocess in enumerate(zip(preprocessing1, preprocessing2)):
  indexer.append(create_index(preprocess[0], preprocess[1], "titles", i))
# Extract the information from the indexer of the last preprocessing␣
 ↪configuration
statistics = pt.IndexFactory.of(indexer[-1]).getCollectionStatistics()
print('Number of Documents:', statistics.numberOfDocuments, '\nNumber of Terms:
 ↪', statistics.numberOfUniqueTerms)

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

```
Number of Documents: 503
Number of Terms: 807
```

1.1.8 1.1.8

Choose several weighting models (at least 2, no more than 5).

For each of them, for each of the indices created in last step, build a retrieval model.

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


# We will use the function 'create_retrieval_model' seen during the lab1
def create_retrieval_model(indexref, scoring_function):
  return pt.BatchRetrieve(indexref, wmodel = scoring_function)

# Our chosen weighting models
weighting_models = ['CoordinateMatch', 'Tf', 'TF_IDF', 'BM25']

retrieval_models = []

# Saving all our models
for indexref in indexer:
  for wmodel in weighting_models:
    retrieval_models.append(create_retrieval_model(indexref, wmodel))


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

1.1.9 1.1.9

Choose several evaluation metrics (at least 3, no more than 6) and put them in a list.

Add the following metrics to the list: Recall at 5, Normalized Discounted Cumulative Gain at 20, Mean Average Precision.

Obviously, the metrics you choose cannot be **completely identical** to these 3 we specified.

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

# Our evaluation metrics
evaluation_metrics = ['recall_5', 'ndcg_cut_20', 'map', 'P_5', 'P_10',
                      'ndcg_cut_5', 'recall_10', 'num_q']
```

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

1.1.10 1.1.10

For each index built in step 1.1.7, run an experiment to obtain the values associated to each evaluation metrics specified in 1.1.8 for each of the weighting models chosen in 1.1.9.

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



names = []
preprocessing = ["", "Stopwords", "EnglishSnowballStemmer", "Stopwords,␣
 ↪EnglishSnowballStemmer"]

# Set the names for the indexes of the result dataframe
for preprocess in preprocessing:
  for model in weighting_models:
    names.append(model + '; ' + preprocess)

# Run the experiment for all our models
experiment_result = pt.Experiment(
    retrieval_models,
    topics,
    qrels,
    eval_metrics=evaluation_metrics,
    names=names)









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

1.1.11 1.1.11

For the last index constructed (i.e. corresponding to the last preprocessing chosen), print out the PyTerrier table with the weighting models chosen by you on the rows and the evaluation metrics chosen by you + those specified by us on the columns.

Highlight the best results in the result table.

[51]:
```
#YOUR CODE STARTS HERE#


# Extract the pd.dataframe and the maximum results on the last preprocessing␣
 ↪configuration
n = len(experiment_result)
experiment_result.iloc[n-len(weighting_models):n]


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

[51]:

	name	recall_5	ndcg_cut_20	map	P_5	P_10	ndcg_cut_5	recall_10	num_q
12	CoordinateMatch; Stopwords, EnglishSnowballSte…	0.083581	0.390297	0.210468	0.472727	0.377273	0.511467	0.112634	44.0
13	Tf; Stopwords, EnglishSnowballStemmer	0.057451	0.308081	0.162349	0.377273	0.336364	0.386131	0.094701	44.0
14	TF_IDF; Stopwords, EnglishSnowballStemmer	0.193017	0.639180	0.375833	0.713636	0.593182	0.772016	0.249467	44.0
15	BM25; Stopwords, EnglishSnowballStemmer	0.195953	0.644357	0.376948	0.722727	0.602273	0.777980	0.253654	44.0

1.1.12 1.1.12

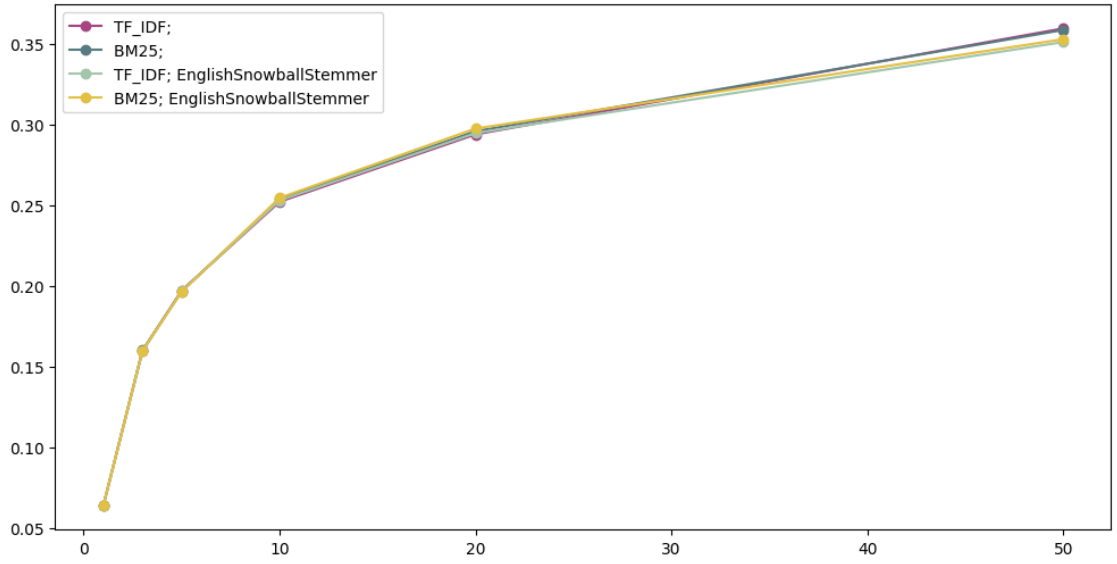
Select the Top-4 configurations (preprocessing, weighting model) according to the Mean Average Precision (MAP), taking into account the results obtained in section 1.1.10.

For these 4 configurations, provide the following plot (re-run the evaluations just for this configurations, to get the required evaluation metrics):

- Recall@k plot
  - the x axis represents the considered values for k: you must consider k {1, 3, 5, 10, 20, 50}
  - the y axis represents the average Recall@k over all provided queries
  - each curve represents one of the 4 search engine configurations

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


# Find which are the Top-4 configurations according to the MAP
top4 = experiment_result['map'].nlargest(n=4).index

# Take the Top-4 configurations
new_models = [x for i, x in enumerate(retrieval_models) if i in top4]

new_eval = ['recall_1', 'recall_3', 'recall_5', 'recall_10', 'recall_20',␣
 ↪'recall_50']

# Run the experiment for the requested evaluation metrics
new_result = pt.Experiment(
    new_models,
    topics,
    qrels,
    eval_metrics=new_eval,
    names=[x for i, x in enumerate(names) if i in top4]
)

# Plot the results
colors = ['#A74482', '#587B7F', '#A2C5AC', '#E2C044', '#878E99', '#BA274A',␣
 ↪'#4C956C', '#7B6D8D', '#044389']
plt.figure(figsize=(12, 6))
new_eval = [1,3,5,10,20,50]

for i in range(4):
  plt.plot(new_eval, new_result.iloc[i][1:], marker='o', color = colors[i])

plt.legend([x for i, x in enumerate(names) if i in top4])
plt.show()
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 50#
```



1.1.13 1.1.13

According only to the Recall@k plot, which is the best search engine configuration? Explain your answer in **at most 3 sentences**.

————YOUR TEXT STARTS HERE————

From the graph seems like the best choice for a configuration would be one between BM25 (with the Stemmer) and TF-IDF (without preprocessing).

In fact, the first has recall a bit higher for k = 20 and the second can be easily recognized as the best for k = 50.

In the end we decided to select the TF_IDF since the BM25's recall seems to get a lot lower for k=50, while for the rest of the values the recalls are almost the same.

1.1.14 1.1.14

For the configuration you selected in Part 1.1.13, provide an **example of the functioning** of your search engine.

The query should be `King Mark of Cornwall`.

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

query = 'King Mark of Cornwall'

# The TF_IDF without preprocessing
example_functioning = retrieval_models[2].search(query)

# Add the titles to the dataframe
example_functioning['titles'] = list(data_frame_book.iloc[[int(x) for x in␣
 ↪example_functioning.docno]].titles)
example_functioning









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

[20]:

	qid	docid	docno	rank	score	query	titles
0	1	231	231	0	7.657937	King Mark of Cornwall	how King Mark was sorry for the good renown of…
1	1	215	215	1	7.626290	King Mark of Cornwall	how King Mark , by the advice of his council ,…
2	1	259	259	2	7.066060	King Mark of Cornwall	how King Arthur made King Mark to be accorded …
3	1	33	33	3	6.911590	King Mark of Cornwall	how a dwarf reproved Balin for the death of La…
4	1	263	263	4	5.588507	King Mark of Cornwall	how King Arthur , the Queen , and Launcelot re…
..	..	…	…	…	…	…	
328	1	360	360	328	0.692548	King Mark of Cornwall	
329	1	137	137	329	0.680346	King Mark of Cornwall	
330	1	417	417	330	0.680346	King Mark of Cornwall	
331	1	135	135	331	0.635553	King Mark of Cornwall	
332	1	133	133	332	0.625261	King Mark of Cornwall	

```
..                                                    …
328  how the Queen desired to see Galahad ; and how…
329  how Beaumains came to the lady , and when he c…
330  how Sir Percivale 's sister bled a dish full o…
331  how after long fighting Beaumains overcame the…
332  how the damosel and Beaumains came to the sieg…

[333 rows x 7 columns]
```

1.2 Part 1.2

1.2.1 1.2.1

Scenario: The company ExcaliburDMT needs a search engine for the book *Le Morte D'Arthur* by Thomas Malory. The book is divided into two volumes.

> They want to consider each chapter as a document with two fields: title of the chapter and corpus of the chapter. For now, they only want to index the title of each chapter.
>
> They only want their users to be able to query character names and match them exactly (also, order is important).
>
> They would like to show 10 results on the screen in a random order, in a sword-like shape.
>
> The company wants to evaluate the performance of the search engine: each chapter's title containing a character's name is considered relevant for a query containing that character's name.

Character names are extracted from the collection of documents. See Part 1.1.3.

Order is important: if the query is "King Arthur", "Arthur, King of Britain" should have less scoring than "King Arthur of Camelot".

What is the configuration (as defined in part 1.2) that would best meet the needs of the Excalibur-DMT company? **Use at most 3 sentences (1 per section).**

————YOUR TEXT STARTS HERE————

Preprocessing: No preprocessing

Weighting model: BM25

Evaluation metric: precision_10

Provide an explanation of your choice in **at most 3 sentences**.

————YOUR TEXT STARTS HERE————

Preprocessing: Titles tent to be short, we choose do use no preprocessing since we don't want to lose too many terms.

Weighting model: We choose this model because, from our researches, we found that PyTerrier implementation of this method should take into account the terms' order.

Evaluation metric: Since we previously saw that it gives back good results.

2 Part 2

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


import csv
import random
import numpy as np
import itertools as it
import pandas as pd
import matplotlib.pyplot as plt
import time


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

[22]:
```
set__characters_of_interest = set(
    [' ', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd',␣
 ↪'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
     'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'])
def cleaner(text, set__characters_of_interest):
    new_text = ""
    #
    previous_copied_character = "a"
    for c_character in text:
        #
        c_character = c_character.lower()
        #
        if c_character not in set__characters_of_interest:
            c_character = " "
        #
        if c_character == " " and c_character == previous_copied_character:
            continue
        #
        new_text += c_character
        #
        previous_copied_character = c_character
        #
    #
    new_text = new_text.strip()
    #
    return new_text
```

[23]:
```
def get_shingle_id(shingle):
    global max_shingle_id
```

```
    global map__shingle__shingle_id
    #
    shingle_id = map__shingle__shingle_id.get(shingle, -1)
    #
    if shingle_id >= 0:
        return shingle_id
    #
    max_shingle_id += 1
    shingle_id = max_shingle_id
    map__shingle__shingle_id[shingle] = max_shingle_id
    #
    return shingle_id
```

[24]:
```
def shingler(text, width=2):
    #
    set__shingle_id = set()
    #
    tokenized_text = text.split(" ")
    #
    max_index_plus_1 = 1 if len(tokenized_text) <= width else␣
 ↪len(tokenized_text) - width +1
    for index in range(max_index_plus_1):
        #
        c_shingle = tuple(tokenized_text[index:index + width])
        #
        shingle_id = get_shingle_id(c_shingle)
        #
        # if shingle_id in set__shingle_id:
        #    print(shingle_id, c_shingle)
        #
        res = set__shingle_id.add(shingle_id)
        #
    return set__shingle_id
```

[25]:
```
def create_sets_of_shingle_ids(input_file_name, output_file_name,
                               input_file_delimiter='\t',␣
 ↪input_file_quotechar='"',
                               set__characters_of_interest=[" "],␣
 ↪shingle_width=3,
                               doc_id_column_idx=0, field_column_idx=1):
    #
    output_file = open(output_file_name, 'w', encoding="utf-8")
    output_file_csv_writer = csv.writer(output_file, delimiter='\t',␣
 ↪quotechar='"', quoting=csv.QUOTE_NONE)
    header = ['set_id', 'set_of_integers']
    output_file_csv_writer.writerow(header)
    #
```

```
    input_file = open(input_file_name, 'r', encoding="utf-8")
    input_file_csv_reader = csv.reader(input_file,␣
 ↪delimiter=input_file_delimiter, quotechar=input_file_quotechar)
    next(input_file_csv_reader)
    for record in input_file_csv_reader:
        #
        doc_id = int(record[doc_id_column_idx])
        document = record[field_column_idx]
        #
        cleaned_document = cleaner(document, set__characters_of_interest)
        #
        set__shingle_id = shingler(cleaned_document, width=shingle_width)
        #
        output_file_csv_writer.writerow([doc_id, set__shingle_id])
        #
        #
        if doc_id % 1000 == 0:
            print("Last processed doc_id:", doc_id)
        #
    input_file.close()
    output_file.close()
    print("Last processed doc_id:", doc_id)
    print()
    print("max_shingle_id=", max_shingle_id)
    print()
    print()
    return max_shingle_id
```

[26]:
```
def is_prime(number):
    #
    if number == 2:
        return True
    if (number % 2) == 0:
        return False
    for j in range(3, int(number ** 0.5 + 1), 2):
        if (number % j) == 0:
            return False
    #
    return True
```

[27]:
```
def create_hash_functions(number_of_hash_functions,␣
 ↪upper_bound_on_number_of_distinct_elements, seed=42):
    random.seed(seed)
    #
    map__hash_function_id__a_b_p = {}
    #
    set_of_all_hash_functions = set()
```

```
    while len(set_of_all_hash_functions) < number_of_hash_functions:
        a = random.randint(1, upper_bound_on_number_of_distinct_elements - 1)
        b = random.randint(0, upper_bound_on_number_of_distinct_elements - 1)
        p = random.randint(upper_bound_on_number_of_distinct_elements, 10 *␣
↪upper_bound_on_number_of_distinct_elements)
        while is_prime(p) == False:
            p = random.randint(upper_bound_on_number_of_distinct_elements,
                               10 * upper_bound_on_number_of_distinct_elements)
        #
        c_hash_function = (a, b, p)
        set_of_all_hash_functions.add(c_hash_function)
    #
    for c_hash_function_id, c_hash_function in␣
↪enumerate(set_of_all_hash_functions):
        map__hash_function_id__a_b_p[c_hash_function_id] = c_hash_function
    #
    return map__hash_function_id__a_b_p
```

[28]:
```
def create_c_set_MinWiseHashing_sketch(c_set,
                                       map_as_list__index__a_b_p,
                                       total_number_of_hash_functions,␣
↪use_numpy_version = True):
    if use_numpy_version:
      app = np.array(map_as_list__index__a_b_p)
      c_set_MinWiseHashing_sketch = list(np.min((app[:,:1]*np.
↪array(list(c_set))[None,:]+app[:,1:2])%app[:,2:],axis=1))
    else:
      plus_inf = float("+inf")
      c_set_MinWiseHashing_sketch = [plus_inf] * total_number_of_hash_functions
      for c_element_id in c_set:
          for index, (a, b, p) in enumerate(map_as_list__index__a_b_p):
              c_hash_value = (a * c_element_id + b) % p
              if c_hash_value < c_set_MinWiseHashing_sketch[index]:
                  c_set_MinWiseHashing_sketch[index] = c_hash_value
              #
          #
      #
    return c_set_MinWiseHashing_sketch
```

[29]:
```
def create_MinWiseHashing_sketches(input_file_name,␣
↪upper_bound_on_number_of_distinct_elements,
                                   ␣
↪number_of_hash_functions_that_is_also_the_sketch_lenght_and_also_the_number_of_simulated_pe
                                   output_file_name, use_numpy_version=True):
    #
    map__hash_function_id__a_b_p = create_hash_functions(
```

```
    ␣
↪number_of_hash_functions_that_is_also_the_sketch_lenght_and_also_the_number_of_simulated_pe
      upper_bound_on_number_of_distinct_elements)
  #
  map__set_id__MinWiseHashing_sketch = {}
  #
  total_number_of_hash_functions = len(map__hash_function_id__a_b_p)
  # sorted_list_all_hash_function_id = sorted(map__hash_function_id__a_b_p.
↪keys())
  map_as_list__index__a_b_p = tuple([(a, b, p) for a, b, p in␣
↪map__hash_function_id__a_b_p.values()])
  #
  input_file = open(input_file_name, 'r', encoding="utf-8")
  input_file_csv_reader = csv.reader(input_file, delimiter='\t',␣
↪quotechar='"', quoting=csv.QUOTE_NONE)
  header = next(input_file_csv_reader)
  num_record_so_far = 0
  for record in input_file_csv_reader:
    num_record_so_far += 1
    if num_record_so_far % 100 == 0:
        print(num_record_so_far)
    c_set_id = int(record[0])
    c_set = eval(record[1])

    c_set_MinWiseHashing_sketch =␣
↪create_c_set_MinWiseHashing_sketch(c_set,map_as_list__index__a_b_p,
                                                                    ␣
↪total_number_of_hash_functions,
                                                                    ␣
↪use_numpy_version)

    #print(len(c_set_MinWiseHashing_sketch))
    map__set_id__MinWiseHashing_sketch[c_set_id] = c_set_MinWiseHashing_sketch
  input_file.close()
  #
  output_file = open(output_file_name, 'w', encoding="utf-8")
  output_file_csv_writer = csv.writer(output_file, delimiter='\t',␣
↪quotechar='"', quoting=csv.QUOTE_NONE)
  header = ['set_id', 'MinWiseHashing_sketch']
  output_file_csv_writer.writerow(header)
  sorted_list_all_set_id = sorted(map__set_id__MinWiseHashing_sketch.keys())
  for c_set_id in sorted_list_all_set_id:
      output_file_csv_writer.writerow([c_set_id,␣
↪str(map__set_id__MinWiseHashing_sketch[c_set_id])])
  output_file.close()
  #
```

```
    return
```

[30]:
```
def load_map__set_id__MinWiseHashing_sketch_from_file(input_file_name):
    map__set_id__MinWiseHashing_sketch = {}
    #
    input_file = open(input_file_name, 'r', encoding="utf-8")
    input_file_csv_reader = csv.reader(input_file, delimiter='\t',␣
 ↪quotechar='"', quoting=csv.QUOTE_NONE)
    header = next(input_file_csv_reader)
    for record in input_file_csv_reader:
        c_set_id = int(record[0])
        c_MinhiseHashing_sketch = tuple(eval(record[1]))
        #
        map__set_id__MinWiseHashing_sketch[c_set_id] = c_MinhiseHashing_sketch
        #
    input_file.close()
    #
    return map__set_id__MinWiseHashing_sketch
```

[31]:
```
def get_set_of_CANDIDATES_to_be_near_duplicates(r, b,␣
 ↪map__set_id__MinWiseHashing_sketch):
    #
    set_of_CANDIDATES_to_be_near_duplicates = set()
    #
    for c_band_progressive_id in range(b):
        #
        print("c_band_progressive_id", c_band_progressive_id)
        #
        c_band_starting_index = c_band_progressive_id * r
        c_band_ending_index = (c_band_progressive_id + 1) * r
        #
        map__band__set_set_id = {}
        #
        for c_set_id in map__set_id__MinWiseHashing_sketch:
            #
            if r * b != len(map__set_id__MinWiseHashing_sketch[c_set_id]):
                n = len(map__set_id__MinWiseHashing_sketch[c_set_id])
                message = "ERROR!!! n != r*b " + str(n) + "!=" + str(r * b) + ";
 ↪ " + str(n) + "!=" + str(r) + "*" + str(
                    b)
                raise ValueError(message)
            #
            c_band_for_c_set = tuple(
                ␣
 ↪map__set_id__MinWiseHashing_sketch[c_set_id][c_band_starting_index:
 ↪c_band_ending_index])
            #
```

```
            if c_band_for_c_set not in map__band__set_set_id:
                map__band__set_set_id[c_band_for_c_set] = set()
            map__band__set_set_id[c_band_for_c_set].add(c_set_id)
            #

        for c_set_set_id in map__band__set_set_id.values():
            #
            if len(c_set_set_id) > 1:
                #
                for set_id_a, set_id_A in it.combinations(c_set_set_id, 2):
                    if set_id_a < set_id_A:
                        set_of_CANDIDATES_to_be_near_duplicates.add((set_id_a,␣
 ↪set_id_A))
                    else:
                        set_of_CANDIDATES_to_be_near_duplicates.add((set_id_A,␣
 ↪set_id_a))
        #
    #
    return set_of_CANDIDATES_to_be_near_duplicates
```

[32]:
```
def compute_approximate_jaccard(set_a_MinWiseHashing_sketch,␣
 ↪set_b_MinWiseHashing_sketch):
    appx_jaccard = 0.
    #
    for index in range(len(set_a_MinWiseHashing_sketch)):
        #
        if set_a_MinWiseHashing_sketch[index] ==␣
 ↪set_b_MinWiseHashing_sketch[index]:
            appx_jaccard += 1
        #
    appx_jaccard /= len(set_a_MinWiseHashing_sketch)
    #
    return appx_jaccard
```

[33]:
```
def␣
 ↪compute_approximate_jaccard_to_REDUCE_the_number_of_CANDIDATES_to_be_near_duplicates(
        set_of_CANDIDATES_to_be_near_duplicates,
        map__set_id__MinWiseHashing_sketch, jaccard_threshold):
    map__set_a_id__set_A_id__appx_jaccard = {}
    #
    for set_a_id, set_A_id in set_of_CANDIDATES_to_be_near_duplicates:
        #
        set_a_MinWiseHashing_sketch =␣
 ↪map__set_id__MinWiseHashing_sketch[set_a_id]
        set_A_MinWiseHashing_sketch =␣
 ↪map__set_id__MinWiseHashing_sketch[set_A_id]
```

```
        #
        appx_jaccard = compute_approximate_jaccard(set_a_MinWiseHashing_sketch,␣
↪set_A_MinWiseHashing_sketch)
        #
        if appx_jaccard >= jaccard_threshold:
            map__set_a_id__set_A_id__appx_jaccard[(set_a_id, set_A_id)] =␣
↪appx_jaccard
        #
    #
    return map__set_a_id__set_A_id__appx_jaccard
```

[34]:
```
def mine_couples_of_Near_Duplicates(input_file_name, r, b, jaccard_threshold):
    #
    print("Starting the loading of the MinWiseHashing sketches from the input␣
↪file.")
    map__set_id__MinWiseHashing_sketch =␣
↪load_map__set_id__MinWiseHashing_sketch_from_file(input_file_name)
    print()
    print("Number of sets=", len(map__set_id__MinWiseHashing_sketch))
    print()
    #
    print("Starting the mining of the CANDIDATES couples to be near duplicates.
↪")
    set_of_CANDIDATES_to_be_near_duplicates =␣
↪get_set_of_CANDIDATES_to_be_near_duplicates(r, b,
                                                                              ␣
↪           map__set_id__MinWiseHashing_sketch)
    #
    print()
    print("Number of pairs of sets to be near-duplicate CANDIDATES=",␣
↪len(set_of_CANDIDATES_to_be_near_duplicates))
    print()
    #
    map__set_a_id__set_A_id__appx_jaccard =␣
↪compute_approximate_jaccard_to_REDUCE_the_number_of_CANDIDATES_to_be_near_duplicates(
        set_of_CANDIDATES_to_be_near_duplicates,␣
↪map__set_id__MinWiseHashing_sketch, jaccard_threshold)
    print()
    print("Number of REFINED pairs of sets to be near-duplicate CANDIDATES=",
          len(map__set_a_id__set_A_id__appx_jaccard))
    print()
    #
    output_file = open(output_file_name, 'w', encoding="utf-8")
    output_file_csv_writer = csv.writer(output_file, delimiter='\t',␣
↪quotechar='"', quoting=csv.QUOTE_NONE)
    header = ['set_a_id', 'set_b_id', 'approximate_jaccard']
```

```
    output_file_csv_writer.writerow(header)
    sorted_list_all_set_id = sorted(map__set_id__MinWiseHashing_sketch.keys())
    for set_a_id__set_A_id in map__set_a_id__set_A_id__appx_jaccard:
        appx_jaccard = map__set_a_id__set_A_id__appx_jaccard[set_a_id__set_A_id]
        output_file_csv_writer.writerow([set_a_id__set_A_id[0],␣
↪set_a_id__set_A_id[1], appx_jaccard])
    output_file.close()
    return
```

2.1 Part 2.1

2.1.1 2.1.1

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

[ ]:
```
#REMOVE_OUTPUT#
!gdown 16LQDmla82XFK1B0lr8H9ycm01pxjURXN
```

2.1.2 2.1.2

Inspect the dataset: print the list of fields names. Print the value of the `song` field for the last 3 documents.

[36]:
```
#YOUR CODE STARTS HERE#

# Load the file
MetroLyrics = pd.read_csv('150K_lyrics_from_MetroLyrics.csv')

# Print the wanted informations
print('Name of the columns: ', list(MetroLyrics.columns))

print('\nLast 3 entries in the "song" column: \n')
MetroLyrics.iloc[-4:-1].song









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

```
Name of the columns:  ['ID', 'song', 'year', 'artist', 'genre', 'lyrics']

Last 3 entries in the "song" column:
```

[36]:
```
149996    do-you-wanna-touch-me-oh-yeah
149997       oh-what-a-fool-i-have-been
149998                        lonely-boy
Name: song, dtype: object
```

2.1.3 2.1.3

Turn the lyrics field of each document into a sets of shingles of length 4 and save the result to a file named `hw1_set_id_set_of_integers.tsv`

[ ]:
```
#REMOVE_OUTPUT#
#YOUR CODE STARTS HERE#
# Initialize the shingle_id and the dictionary
max_shingle_id = -1
map__shingle__shingle_id = {}
# Define our parameters
input_file_name = '150K_lyrics_from_MetroLyrics.csv'
input_file_delimiter = ','
input_file_quotechar = '"'
output_file_name = "hw1_set_id_set_of_integers.tsv"
shingle_width = 4
doc_id_column_idx = 0
field_column_idx = 5
# Use the predefine function 'create_sets_of_shingle_ids'
max_shingle_id = create_sets_of_shingle_ids(input_file_name, output_file_name,␣
 ↪input_file_delimiter,
                                             input_file_quotechar,␣
 ↪set__characters_of_interest, shingle_width,
                                             doc_id_column_idx, field_column_idx)

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

2.1.4 2.1.4

Load the file containing the sets of shingles and apply MinWiseHashing, saving the result into a file named `hw1_set_id_MinWiseHashing_sketch.tsv`. Choose the number of hash functions (n) in relation to the constraints highlighted at the beginning of part 2. Provide an explanation for your choice in **exactly one sentence**.

————YOUR TEXT STARTS HERE————

We choose n=210 because it's a multiple of different prime numbers (this gives us more choices for r and b) and it's big enough to mantain lot of the information.

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

input_file_name = "hw1_set_id_set_of_integers.tsv"
output_file_name = 'hw1_set_id_MinWiseHashing_sketch.tsv'

# Number of hash functions
n = 210

# Call the function 'create_MinWiseHashing_sketches' that will compute the␣
 ↪MinWiseHashing
create_MinWiseHashing_sketches(input_file_name, max_shingle_id, n,
                               output_file_name)









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

2.1.5 2.1.5

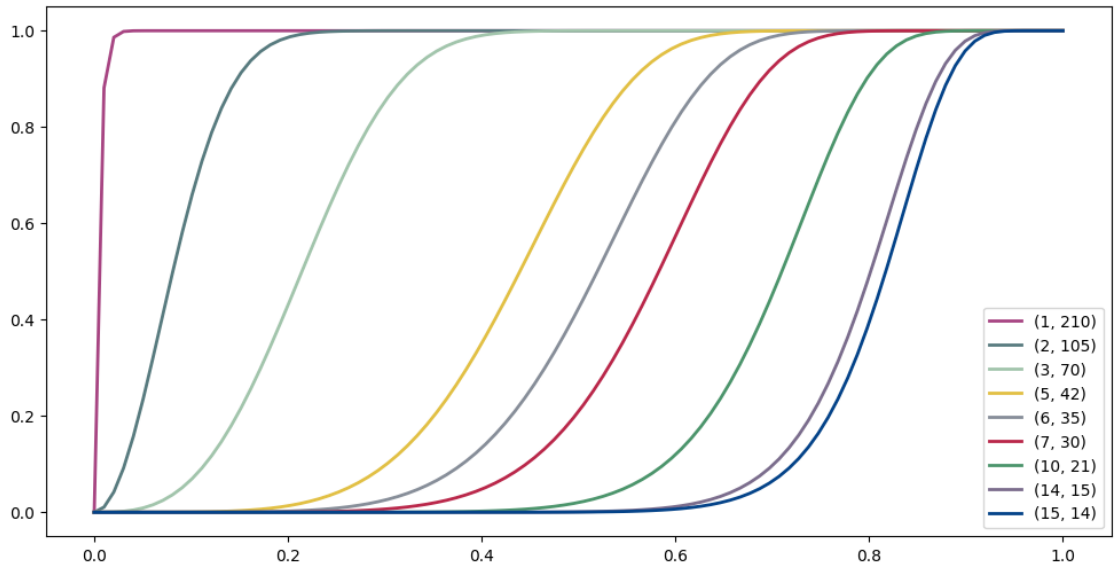
To perform Locality Sensivity Hashing, you have to choose the number of rows (r) and the number bands (b). List all the possible choices of r and b that satisfy the constraints highlighted at the beginning of part 2, according to the number of hash functions you chose.

For all of these configurations, plot all the associated S-curves. The S-curve is defined as the probability (y-axis) that a pair of documents with Jaccard similarity j (x-axis) is selected as a near-duplicate candidate given r and b. Plot all S-curves in the same plot.

[39]:
```
#YOUR CODE STARTS HERE#

r_b = []

# Function that compute the S-curve
def s_function(r, b, j):
  return(1-(1-(j**r))**b)

# Creating all the possible tuple (r,b) such that the constraint is satisfied
for r in range(1,n):
  if n%r==0:
    b = int(n/r)
    if (1-(0.93**r))**b<0.04:
      r_b.append((r,b))

# Show the tuples
```

```
print(r_b, '\n')

# Plot the S-curves
plt.figure(figsize=(12,6))

for i, rb in enumerate(r_b):
  x=np.linspace(0, 1, 100)
  plt.plot(x, s_function(rb[0], rb[1], x), color = colors[i], linewidth = 2)
plt.legend(r_b)
plt.show()


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

```
[(1, 210), (2, 105), (3, 70), (5, 42), (6, 35), (7, 30), (10, 21), (14, 15),
(15, 14)]
```



2.1.6 2.1.6

Among all the configurations you plotted in the previous step, choose the one that gives the smallest amount of False-Positives and False-Negatives near-duplicates candidates, satisfying the provided constraints. You **must** take into account, that after the LSH procedure, the approximate Jaccard similarity between near-duplicate candidates is computed and used to reduce their number.

Provide an explanation for your choice in **at most 3 sentences**.

[40]:
```
#YOUR CODE STARTS HERE#
r = r_b[3][0]
b = r_b[3][1]
#YOUR CODE ENDS HERE#
#THIS IS LINE 5#
```

————YOUR TEXT STARTS HERE————

Since we **must** take into account that the approximate Jaccard will be computed, we will get rid of the False Positives, we just need to choose the configuration that minimize the other parameters (time and False Negatives).

First for the False Negatives, we saw (using the s_function) that the values [(1, 210), (2, 105), (3, 70), (5, 42)] are the one that minimize the probability of False Negatives, P(FN|J=0.93)=0.

Now we just need to minimize the time, to stay inside the 2 minutes constraint, so out of the 4 configutions mentioned above, we will take the rightmost.

2.1.7 2.1.7

Load the file containing the MinWiseHashing sketches and perform Locality Sensivity Hashing, using the parameters you chose in last step, considering also the computation of approximate Jaccard to reduce the number of candidates. Save the Near-Duplicates candidates obtained to a file named `hw1_NearDuplicates_set_a_id_set_b_id_approximate_jaccard.tsv`.

Print the execution time.

[41]:
```
#YOUR CODE STARTS HERE#

# Define our Jaccard threshold
jaccard_threshold = 0.93

input_file_name = "hw1_set_id_MinWiseHashing_sketch.tsv"
output_file_name = "hw1_NearDuplicates_set_a_id_set_b_id_approximate_jaccard.
 ↪tsv"

start_time = time.time()

# Call the function that will apply the LSH
mine_couples_of_Near_Duplicates(input_file_name, r, b, jaccard_threshold)

print('Execution time:', str(round(time.time()-start_time, 3))+'s')









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

```
Starting the loading of the MinWiseHashing sketches from the input file.

Number of sets= 150000

Starting the mining of the CANDIDATES couples to be near duplicates.
c_band_progressive_id 0
c_band_progressive_id 1
c_band_progressive_id 2
```

```
c_band_progressive_id 3
c_band_progressive_id 4
c_band_progressive_id 5
c_band_progressive_id 6
c_band_progressive_id 7
c_band_progressive_id 8
c_band_progressive_id 9
c_band_progressive_id 10
c_band_progressive_id 11
c_band_progressive_id 12
c_band_progressive_id 13
c_band_progressive_id 14
c_band_progressive_id 15
c_band_progressive_id 16
c_band_progressive_id 17
c_band_progressive_id 18
c_band_progressive_id 19
c_band_progressive_id 20
c_band_progressive_id 21
c_band_progressive_id 22
c_band_progressive_id 23
c_band_progressive_id 24
c_band_progressive_id 25
c_band_progressive_id 26
c_band_progressive_id 27
c_band_progressive_id 28
c_band_progressive_id 29
c_band_progressive_id 30
c_band_progressive_id 31
c_band_progressive_id 32
c_band_progressive_id 33
c_band_progressive_id 34
c_band_progressive_id 35
c_band_progressive_id 36
c_band_progressive_id 37
c_band_progressive_id 38
c_band_progressive_id 39
c_band_progressive_id 40
c_band_progressive_id 41

Number of pairs of sets to be near-duplicate CANDIDATES= 36436


Number of REFINED pairs of sets to be near-duplicate CANDIDATES= 17102

Execution time: 88.675s
```

2.1.8 2.1.8

Load the file containing the number of near-duplicates candidates. Print the number of near-duplicates candidates you found.

[42]:
```
#YOUR CODE STARTS HERE#

# Loading the file
output_file = open(output_file_name, 'r', encoding="utf-8")
output_file_csv_reader = csv.reader(output_file, delimiter='\t', quotechar='"',␣
 ↪quoting=csv.QUOTE_NONE)

# Count the number of candidates that is equal to the number of elements in␣
 ↪'output_file_csv_reader'
header = next(output_file_csv_reader)
cont=0

for _ in output_file_csv_reader:
    cont+=1

output_file.close()

# Print the number of candidates
print('Number of near-duplicates candidates:', cont)









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

```
Number of near-duplicates candidates: 17102
```

2.2 Part 2.2

You will be given a scenario and you will have to provide the best solution.

2.2.1 2.2.1

Let us consider the same scenario as in Part 2.1, with the only addition of not wanting more than 100 False Negatives. How would the choice of the LSH configuration change? Would you need any more information to satisfy the new constraint?

————YOUR TEXT STARTS HERE————

Given our previous choices (n = 210, r = 5, b = 42): We already said that we have probablity of getting a False Negative is 0 for J=0.93 and, since this probability decreases while J grows, we have P(FN|J>=0.93)=0. So we don't need to apply changes to our configuration.

In general we don't have enough informations: From the slides, we know that the False Negative rate can be computed as the area over the S-Curve and to the right of the threshold J. So, to be able to find the actual number of False Negatives, we would need some more informations, like the real number of Near Duplicates.