DMT2023_HW1

March 31, 2023

————YOUR TEXT STARTS HERE————

Ben Kakitie, Joanna O., 1853548

Lauro, Francesco, 1706784

[ ]:
```
#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
from collections import defaultdict
import seaborn as sns,  matplotlib as mpl,  matplotlib.pyplot as plt
#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

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

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

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

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

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

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

docno1 = [str(l) for l in list(range(1, len(title1)+1))]
docno2 = [str(l) for l in list(range(len(title1)+1, len(title1) +␣
 ↪len(title2)+1))]
docno = [*docno1, *docno2]
#Transform the list into a pandas DataFrame (a PyTerrier friendly structure).

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





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

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

df.head(8)





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

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

```
6     7  how King Arthur was crowned , and how he made …
7     8  how King Arthur held in Wales , at a Pentecost…

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

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

    remember = ""
    last_is_special_token = False

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

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

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

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









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

[8]:
```
#YOUR CODE STARTS HERE#

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



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

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

[9]:
```python
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
```

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

qid = [str(q) for q in range(1, len(all_characters)+1)]
query = list(all_characters)

topics = pd.DataFrame(list(zip(qid, query)), columns=["qid", "query"])
# topics.info()
```

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

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

topics.head(5)





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

[11]:
```
  qid            query
0   1        Sir Amant
1   2  King of England
2   3    Sir Aglovale
3   4        Excalibur
4   5          Ireland
```

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

rel = []

for i in range(len(all_characters)):
  for char in all_characters:
    if char in text[i]:
      rel.append(1)
    else:
      rel.append(0)



qrels = pd.DataFrame(list(zip(qid, docno, rel)), columns=["qid", "docno",␣
 ↪"label"])
















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

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

print(qrels.take([0, -1]))

print(qrels.shape)




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

```
     qid docno  label
0      1     1      0
224  225   225      0
(225, 3)
```

```
[14]: #YOUR CODE STARTS HERE#
def create_index(preprocessing_configuration, field):
  pd_indexer = pt.DFIndexer("./Inverted_Index", overwrite=True)
  pd_indexer.setProperty("termpipelines", preprocessing_configuration)
  indexref = pd_indexer.index(df[field], df["docno"])
  return indexref

possible_preprocessing = ["Stopwords", #remove stopwords
                          "EnglishSnowballStemmer", #Probably the most famous␣
 ↪stemmer in the world
                          "Stopwords, EnglishSnowballStemmer", #Both previous␣
 ↪ones
                          "PorterStemmer"]  #remove commoner morphological and␣
 ↪inflexional endings from words in English

indeces = {}  #store results in a dictionary
for p in possible_preprocessing:
  indexref = create_index(p, "title")
  indeces[p] = indexref


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

```
<ipython-input-14-a43f5f8a7dc6>:5: DeprecationWarning: Setting of termpipelines
property directly is deprecated
  indexref = pd_indexer.index(df[field], df["docno"])
```

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

def create_retrieval_model(indexref, scoring_function):
  return pt.BatchRetrieve(indexref, wmodel = scoring_function)

possible_wmodels = ["CoordinateMatch", #Term presence
                    "Tf", #Term frequency
                    "TF_IDF",
                    "BM25"]

wmodels = defaultdict(dict)  #store results in a dictionary
for k, v in indeces.items():
  for wm in possible_wmodels:
    terms_presence_preproc = create_retrieval_model(indexref, wm)  #Boolean␣
 ↪matching: 1=if a term is inside a query
    wmodels[k][wm] = terms_presence_preproc




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

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

eval_metrics=["P_1",
              "P_3",
              "num_q",
              "recall_5",
              "ndcg_cut_20",
              "map"]











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

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

res_exp = {} #store results from experiment in a dictionary
```

```
for k in wmodels.keys():
  index = wmodels[k] #exctract the index
  res_exp[k] = pt.Experiment(
                  [index[wm] for wm in index],
                  topics,
                  qrels,
                  names = possible_wmodels,
                  eval_metrics = eval_metrics,
                  highlight="bold"
              )
  print(f'preprocessing: {k}')
  display(res_exp[k])











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

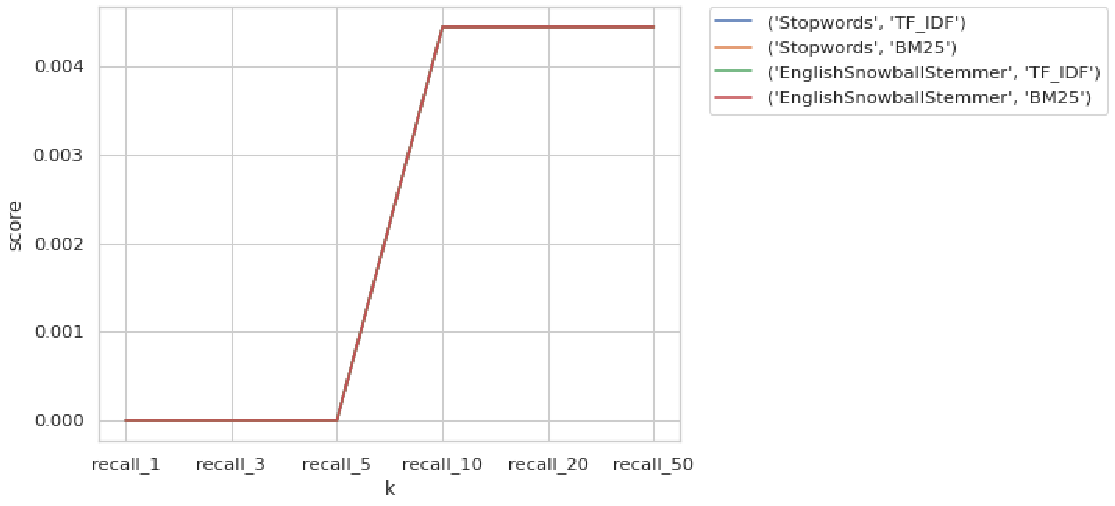
```
preprocessing: Stopwords

<pandas.io.formats.style.Styler at 0x7fc4b1efebb0>

preprocessing: EnglishSnowballStemmer

<pandas.io.formats.style.Styler at 0x7fc4b19c0cd0>

preprocessing: Stopwords, EnglishSnowballStemmer

<pandas.io.formats.style.Styler at 0x7fc4b1efeb20>

preprocessing: PorterStemmer

<pandas.io.formats.style.Styler at 0x7fc4b241b100>
```

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

last_index = res_exp[list(res_exp.keys())[-1]]

hli = last_index.highlight_max(color = 'lightgreen', axis = 0)

display(hli)

```

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

```
<pandas.io.formats.style.Styler at 0x7fc4b241b100>
```

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

result_top4 = pd.concat([res_exp[k].data[['name','map']] for k in res_exp.
 ↪keys()],
                  keys = res_exp.keys(),
                  names = ['index'] )

top4 = result_top4.sort_values('map',
                         ascending = False)[:4].reset_index().rename(columns=␣
 ↪{'index':'preprocessing',
                                                                              ␣
 ↪'name':'weighting model'})

sns.set(rc={"figure.dpi":75})
sns.set_context('notebook')
sns.set_style("whitegrid")

lst = list(zip(*map(top4[['preprocessing','weighting model']].get,
                    top4[['preprocessing','weighting model']])))

new_eval = {}
for el in lst:
  val = []
  for l in el:
    val.append(create_retrieval_model(indexref, l))
  new_eval[el] = [v for v in val]
# print(new_eval.values()) #dict_values([['CoordinateMatch', 'Tf'], ['Tf',␣
 ↪'Tf'], ['TF_IDF', 'Tf'], ['BM25', 'Tf']])

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

res_exp_recall_k_plot = pt.Experiment(
                        [wmodels[i][model] for (i,model) in lst],
                        topics,
                        qrels,
                        names = [str(i) for i in new_eval.keys()], # == lst
                        eval_metrics= new_metrics,)

res_exp_recall_k_plot= res_exp_recall_k_plot.set_index('name').stack().
 ↪reset_index()
res_exp_recall_k_plot.columns = ['configuration','k','score']


sns.lineplot(data= res_exp_recall_k_plot,
             x = 'k', y = 'score', hue = 'configuration')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)
plt.show()




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



————YOUR TEXT STARTS HERE————

The best search engine configuration for the Recall@k is the one that uses “EnglishSnowballStammer” as preprocessing and “BM25” as weighting model. Nevertheless, since it relies on the document collection to which the search engine is applied and the evaluation query used for a topic research, this does not necessarily imply that these are the best configurations in general.

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

query = 'King Mark of Cornwall'
BM25_rm = create_retrieval_model(indexref, scoring_function = "BM25")

search_results = BM25_rm.search(query)
display(search_results)

def display_formatted_query_result(results_dict, df):
  for field,results in results_dict.items():
    previous_max_col_width = pd.options.display.max_colwidth #save current␣
 ↪max_col_width
    pd.options.display.max_colwidth = 1000 #change max_colwidth to display more␣
 ↪text
    merged_df = pd.merge(results, df, left_on='docno', right_on='docno') #merge␣
 ↪the two df
    merged_df = merged_df.loc[:, ["rank", "score", *field]] #subset to columns␣
 ↪we are interested in
    display(merged_df) #display result
    pd.options.display.max_colwidth = previous_max_col_width #reset previous␣
 ↪max_col_width

display_formatted_query_result({("title",):search_results},df)
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

```
    qid  docid docno  rank      score                  query
0     1    259   260     0  11.396126  King Mark of Cornwall
1     1    215   216     1  10.069068  King Mark of Cornwall
2     1    231   232     2   9.954499  King Mark of Cornwall
3     1     33    34     3   9.328458  King Mark of Cornwall
4     1    252   253     4   7.401941  King Mark of Cornwall
..   ..    …     …     …        …                      …
328   1     24    25   328  -0.397608  King Mark of Cornwall
329   1    390   391   329  -0.397608  King Mark of Cornwall
330   1    295   296   330  -0.402207  King Mark of Cornwall
331   1    282   283   331  -0.426897  King Mark of Cornwall
332   1    410   411   332  -0.432203  King Mark of Cornwall

[333 rows x 6 columns]

     rank      score  \
0       0  11.396126
1       1  10.069068
2       2   9.954499
3       3   9.328458
4       4   7.401941
..    …        …
```

```
328   328  -0.397608
329   329  -0.397608
330   330  -0.402207
331   331  -0.426897
332   332  -0.432203

                                                                                    ␣
 ↪                                               title
0                         how King Arthur made King Mark to be accorded with Sir␣
 ↪Tristram , and how they departed toward Cornwall
1                      how King Mark , by the advice of his council , banished␣
 ↪Sir Tristram out of Cornwall the term of ten years
2    how King Mark was sorry for the good renown of Sir Tristram some of King␣
 ↪Arthur 's knights jousted with knights of Cornwall
3     how a dwarf reproved Balin for the death of Lanceor , and how King Mark of␣
 ↪Cornwall found them , and made a tomb over them
4          how King Mark had slain Sir Amant wrongfully to-fore King Arthur ,␣
 ↪and Sir Launcelot fetched King Mark to King Arthur
..                                                                                  ␣
 ↪                                                 …
328                                               how Arthur by the mean of Merlin␣
 ↪gat Excalibur his sword of the Lady of the Lake
329                                      how Sir Gawaine was nigh weary of the␣
 ↪quest of the Sangreal , and of his marvellous dream
330                                      how they approached the Castle Lonazep ,␣
 ↪and of other devices of the death of Sir Lamorak
331                                                                            of␣
 ↪the fourth day , and of many great feats of arms
332                                                                                 ␣
 ↪of the marvels of the sword and of the scabbard

[333 rows x 3 columns]
```

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: CoordinateMatch

Evaluation metric: BM25

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

————YOUR TEXT STARTS HERE————

If "EnglishSnowballStemmer" is used, it would undermind the order of the words (requirement B), or if "PorterStemmer" is used, it would remove commoner morphological and inflexional endings from words in English ("The King of England" == "King of England").

"CoordinateMatch" gives importance to the presence of a term in the query.

"BM25" improves average precision in respect to "TD-IDF"

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

import pandas as pd
from tqdm import tqdm
import csv, time, random
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import matplotlib.pyplot as plt
import itertools as it


#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
```

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

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

df = pd.read_csv("/content/150K_lyrics_from_MetroLyrics.csv")


print(df.columns.values.tolist())

print(df.iloc[[-3,-2,-1]].song)











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

```
['ID', 'song', 'year', 'artist', 'genre', 'lyrics']
149997    oh-what-a-fool-i-have-been
149998                     lonely-boy
149999                sidewalk-sinner
Name: song, dtype: object
```

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

#The files created in this homework from now on will be put in the /content/
↪drive/MyDrive folder.
max_shingle_id = 0
map__shingle__shingle_id = {}
```

```
output_file = open("/content/drive/MyDrive/hw1_set_id_set_of_integers.tsv",␣
  ↪'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)
#print(shingles_set)

for index, row in tqdm(df.iterrows()):
  output_file_csv_writer.writerow([index, shingler(row.lyrics, 4)])

output_file.close()

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

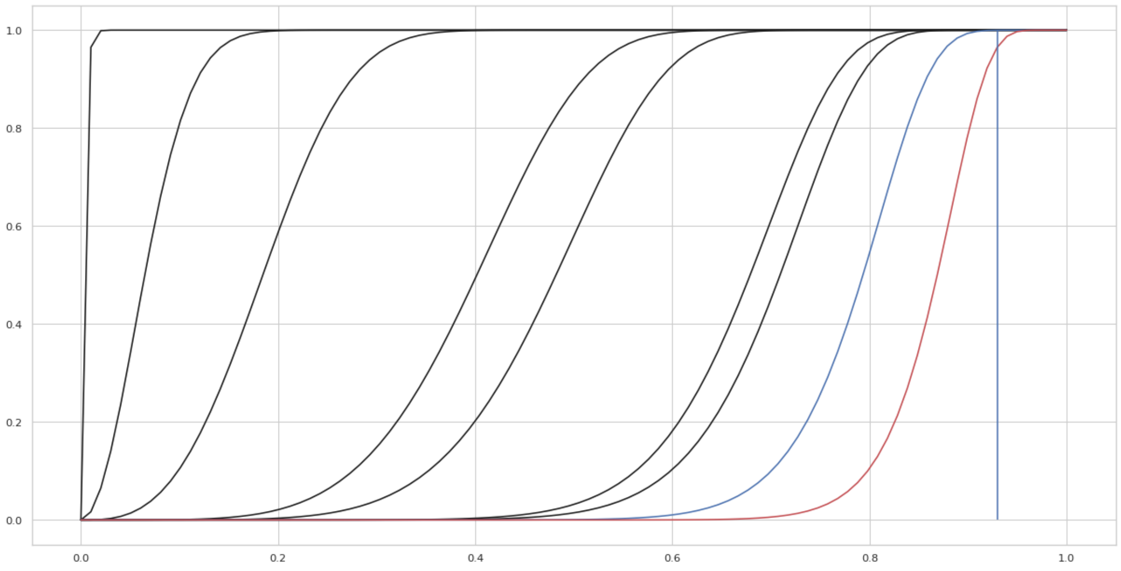
————YOUR TEXT STARTS HERE————

We chose 330 hash functions because it's very close to 332, the highest possible number given the constraints - and all else equal, the higher the number of hash functions is, the more accurate the result is likely to be - while having many more dividers than 332, therefore allowing for many more possible configurations.

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

create_MinWiseHashing_sketches("/content/drive/MyDrive/
 ↪hw1_set_id_set_of_integers.tsv", 150000,
                                   330, "/content/drive/MyDrive/
 ↪hw1_set_id_MinWiseHashing_sketch.tsv")













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

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

# All the possible (r,b) combinations that satisfy the constraints highlighted␣
 ↪at the beginning of part 2 are:
# (1,330), (2,165), (3,110), (5,66), (6,55), (10,33), (11,30), (15,22), (22,15)

def probability_to_be_selected(j,r,b):
  return 1-((1-j**r)**b)

x = np.linspace(0,1,100)
y1 = probability_to_be_selected(x,1,330)
y2 = probability_to_be_selected(x,2,165)
y3 = probability_to_be_selected(x,3,110)
y4 = probability_to_be_selected(x,5,66)
y5 = probability_to_be_selected(x,6,55)
y6 = probability_to_be_selected(x,10,33)
y7 = probability_to_be_selected(x,11,30)
y8 = probability_to_be_selected(x,15,22)    #this function will be plotted in␣
 ↪blue
y9 = probability_to_be_selected(x,22,15)    #this function will be plotted in␣
 ↪red


plt.rcParams["figure.figsize"] = (20,10)
```

```
fig = plt.figure()
plt.
 ↪plot(x,y1,"k",x,y2,"k",x,y3,"k",x,y4,"k",x,y5,"k",x,y6,"k",x,y7,"k",x,y8,"b",x,y9,"r")
plt.vlines(0.93,0,1)
plt.show()



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



[40]:
```
#YOUR CODE STARTS HERE#
r = 15
b = 22
#YOUR CODE ENDS HERE#
#THIS IS LINE 5#
```

————YOUR TEXT STARTS HERE————

Our choice is the (r = 15, b = 22) combination because we can see from the plot that all combinations, except for (r = 22, b = 15), give an amount of False-Negatives - which constitute the worse type of error - close to 0, as shown by the fact that the probability of two documents with a jaccard similarity >= 0.93 to become near-duplicate candidates is extremely close to 1. Moreover, by looking at the chart it's evident that, among all the combinations different from (r = 22, b = 15), the combination (r = 15, b = 22) is the one that is likely to give the smallest amount of False-Positives and to select the smallest amount of candidates, allowing for the smallest computational time.

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

start = time.time()

#global output_file_name
output_file_name = "/content/drive/MyDrive/
 ↪hw1_NearDuplicates_set_a_id_set_b_id_approximate_jaccard.tsv"

mine_couples_of_Near_Duplicates("/content/drive/MyDrive/
 ↪hw1_set_id_MinWiseHashing_sketch.tsv", r, b, 0.93)

end = time.time()
print("Execution time:", end - start, "seconds")
















#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

Number of pairs of sets to be near-duplicate CANDIDATES= 17926


Number of REFINED pairs of sets to be near-duplicate CANDIDATES= 15920

Execution time: 67.7056314945221 seconds
```

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

df = pd.read_csv("/content/drive/MyDrive/
 ↪hw1_NearDuplicates_set_a_id_set_b_id_approximate_jaccard.tsv", sep='\t')

#print(df.head())

print(len(df))
```

```




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

```
15920
```

————YOUR TEXT STARTS HERE————

A trivial choice of the LSH configuration would be one that considers all document pairs as candidates (except for at most 100 pairs). Such configuration would have a corresponding S-curve whose rapid increase part is positioned to the extreme left of the plot. Given additional information about the true number of near-duplicate documents in the collection, we can move the curve to the right and select less near-duplicate candidates, as long as:

1. we check every candidate later and verify that the number of true near-duplicate documents given in output by the pipeline is greater or equal than the total number of near-duplicate documents in the collection subtracted by 100
2. we respect the 2 minutes time constraint.