Information_Retrieval

July 20, 2021

1 Implementing an Information Retrieval Engine

in this project we are going to implement an information retrieval engine with this steps. * tokenization with normalization * creating inverted index and posting lists * binary searching * ranking search using tfidf weighting ### some ideas to increase speed: * champion list * clustering (K-Means) ### more options: * using labeled datasets to predict label of new datasets and adding feature of classification searching

[10]:

```
import pandas as pd
import numpy as np
import re
from re import search
import codecs
import sys
import string
import math
import heapq
import pickle
```

[2]:

```
stop_words = []
prefixes = []
postfixes = []
verb_roots = []
common_words = []
plural_singular = []
```

[3]:

```
########################################importing
def read_dataset(path, name):
    df = pd.read_excel(path + name)
    return df

def read_files():
    path = 'files/'

    #stop_words
    name = 'stop_words.txt'
    f = open(path + name, 'r', encoding='utf-8')
    Lines = f.read().splitlines()
```

```
# Strips the newline character
for line in Lines:
    stop_words.append(line)
f.close()

#prefixes
name = 'prefixes.txt'
f = open(path + name, 'r', encoding='utf-8')
Lines = f.read().splitlines()
# Strips the newline character
for line in Lines:
    prefixes.append(line)
f.close()

#postfixes
name = 'postfixes.txt'
f = open(path + name, 'r', encoding='utf-8')
Lines = f.read().splitlines()
# Strips the newline character
for line in Lines:
    postfixes.append(line)
f.close()

# verb_roots
name = 'verb.txt'
f = open(path + name, 'r', encoding='utf-8')
Lines = f.read().splitlines()
count = 0
# Strips the newline character
for line in Lines:
    line = line.split('-')
    verb_roots.append(line)
f.close()

#common_words
name = 'common_words.txt'
f = open(path + name, 'r', encoding='utf-8')
Lines = f.read().splitlines()
# Strips the newline character
for line in Lines:
    common_words.append(line)
f.close()

#plural_singular
name = 'plural_singular.txt'
f = open(path + name, 'r', encoding='utf-8')
Lines = f.read().splitlines()
```

```
    count = 0
    # Strips the newline character
    for line in Lines:
        line = line.split('-')
        plural_singular.append(line)
    f.close()

#######################################Normalization
def delete_punctuations(doc):
    punctuations = ' : !»«()[]*,{.}@!?&$#<> !~"\|-_+'
    edited_doc = doc.translate(str.maketrans('', '', punctuations))
    edited_doc = edited_doc.translate(str.maketrans('', '', string.punctuation))

    return edited_doc

def delete_stopWords(doc):
    edited_doc = doc
    for s in stop_words:
        my_regex = r"\b"+s+r"\b"
        edited_doc = re.sub(my_regex , "", edited_doc)
    return edited_doc

def delete_highFrequencyWords(inverted_indexes):
    for i in inverted_indexes:
        if len(i[1])/len(df) > 0.8:
            inverted_indexes.remove(i)
    return inverted_indexes

def delete_postfixes(doc):
    edited_doc = doc
    for p in postfixes:
        my_regex = p + r"\b"
        edited_doc = re.sub(my_regex , "", edited_doc)
    return edited_doc

def delete_prefixes(doc):
    edited_doc = doc
    for p in postfixes:
        my_regex = r"\b" + p
        edited_doc = re.sub(my_regex , "", edited_doc)
    return edited_doc

def replaceWithRoot(tokens):
    for i in range(0, len(tokens)):
        # print(verb_roots)
        for root in verb_roots:
            if search(root[0], tokens[i]) or search(root[1], tokens[i]):
                common = False
                for c in common_words:
                    if c == tokens[i]:
                        common = True
                        # print("common")
                        break
                if common == False:
                    my_regex = r"\b^"+root[1]+r"\.*"
                    x = re.findall(my_regex, tokens[i])

                    if len(x) == 0:
                        my_regex = r"\b...*"+root[1]+r"\b"
                        x1 = re.findall(my_regex, tokens[i])
                        if len(x1)==0:
                            # print("root")
                            tokens[i] = tokens[i].replace(tokens[i], root[2])

    return tokens

def replaceArabicWords(doc):
    doc = doc.replace(' ', ' '(
    doc = doc.replace(' ', ' '(
    doc = doc.replace(' ', ' '(
    doc = doc.replace(' ', ' '(
    doc = doc.replace(' ', ' '(
    doc = doc.replace(' ', ' '(
    doc = doc.replace(' ', ' '(
    doc = doc.replace(' ', ' '(
    doc = doc.replace(' ', ' '(
    return doc

def pluralToSingular(tokens):
    for i in range(0, len(tokens)):
        for ps in plural_singular:
            if search(ps[0], tokens[i]):
                tokens[i] = tokens[i].replace(tokens[i], ps[1])
    return tokens

########################################tokenization
def tokenize(df):
    content = df.content

    tokens = []
    for i in range(0, content.size):
        doc = content[i]
        # 68000 tokens
        doc = delete_punctuations(doc)
        # 50000 tokens
        doc = delete_stopWords(doc)

        doc = replaceArabicWords(doc)

        doc = delete_postfixes(doc)

        doc = delete_prefixes(doc)


        tokenized_doc = doc.split()
        tokenized_doc = replaceWithRoot(tokenized_doc)
        tokenized_doc = pluralToSingular(tokenized_doc)

        for token in tokenized_doc:
            if len(token) > 0:
                temp = []
                temp.append(token)
                temp.append(df.id[i])
                tokens.append(temp)
    # tokens.sort()
    return tokens

#######################################create inverted index
def create_inverted_indexes(tokens):
    tokens.sort()
    inverted_indexes = []
    doc_temp = []
    token_temp = ""
    for token in tokens:
        if token[0] == token_temp:
            doc_temp.append(token[1])
        else:
            if len(token[0])>0:
                temp = []
                temp.append(token_temp)

                (unique, counts) = np.unique(doc_temp, return_counts=True)
                # frequencies = np.array((unique, counts), dtype='i4,f4').T.
 ↪view(np.recarray)
                frequencies = []
                for i in range(len(unique)):
                    temp1 = []
                    temp1.append(unique[i])
                    temp1.append(counts[i])
                    frequencies.append(temp1)
                temp.append(frequencies)
                inverted_indexes.append(temp)
                doc_temp = []
                doc_temp.append(token[1])

        token_temp = token[0]
        # if len(doc_temp) > 0 and token[1] != doc_temp[-1]:

    inverted_indexes = delete_highFrequencyWords(inverted_indexes)
    return inverted_indexes

#######################################calculate weights
def calculate_tfidf(df, inverted_indexes):
    doc_length = np.zeros(len(df)+1)
    len_df = len(df)
    for i in range(len(inverted_indexes)):
        if len(inverted_indexes[i][1]) > 0:
            # print(weighted_inverted_index[i])
            # print(len(inverted_indexes[i][1]))
            idf = math.log(len_df/len(inverted_indexes[i][1]), 10)
            inverted_indexes[i].append(idf)
            # print("idf = ", idf)
            for j in range(len(inverted_indexes[i][1])):
                tf = 1 + math.log(inverted_indexes[i][1][j][1], 10)
                # print("tf = ", tf)
                w = tf*idf
                # print("w : ",w)
                inverted_indexes[i][1][j][1] = w
                # if inverted_indexes[i][1][j][0] == 6545:
                #     print("6545: ", inverted_indexes[i][1][j])
                #     print("6545: ", inverted_indexes[i][0])
                # if inverted_indexes[i][1][j][0] == 6550:
                #     print("6550: ", inverted_indexes[i][1][j])
                #     print("6550: ", inverted_indexes[i][0])


                doc_length[inverted_indexes[i][1][j][0]] =␣
 ↪doc_length[inverted_indexes[i][1][j][0]] + np.power(w, 2)

    doc_length =  np.sqrt(doc_length)
    return inverted_indexes, doc_length

#######################################create champion list
def create_championList(inverted_indexes, r):
    limit = r
    championList = []
    for i in range(len(inverted_indexes)):
        temp = []
        temp.append(inverted_indexes[i][0])
        temp_postingList = list(inverted_indexes[i][1])
        temp_postingList.sort(key=lambda x: x[1], reverse=True)

        temp1 = []
        r = limit
        if r > len(temp_postingList):
            r = len(temp_postingList)
        for j in range(r):
            # print(temp_postingList[j])
            temp1.append(temp_postingList[j])
        temp.append(temp1)
        if len(inverted_indexes[i]) == 3:
            temp.append(inverted_indexes[i][2])
        championList.append(temp)
    return championList

#######################################Searching
def search_token(inverted_indexes, token_name):
    # for token in inverted_indexes:
    #     if token[0] == token_name:
    #         # print("token*****: ", token[0])
    #         return token[1], token[2]
    if token_name in inverted_indexes:
        return inverted_indexes[token_name][0], inverted_indexes[token_name][1]
    # print("this token not exist in our database")
    return -1, -1

def print_res(df, res):
    p_res = pd.DataFrame()

    p_res = pd.merge(df, res, on=['id'], how='inner')
    if 'topic' in df:
        our_res = p_res[['id', 'rank', 'topic', 'url']]
    else:
        our_res = p_res[['id', 'rank', 'url']]
    our_res = our_res.sort_values(["rank"], ascending=False)

    return(our_res)


def binary_search(df, inverted_indexes, query):
    res = []
    docs_id = pd.DataFrame()


    query = delete_punctuations(query)
    query = delete_stopWords(query)
    query = delete_postfixes(query)
    query = delete_prefixes(query)
    query = replaceArabicWords(query)

    tokenized_query = query.split()
    tokenized_query = replaceWithRoot(tokenized_query)
    tokenized_query = pluralToSingular(tokenized_query)

    if len(tokenized_query) == 0:
        print("It looks like there aren't many great matches for your search")
        return


    for i in range (0 ,len(tokenized_query)):
        docs, idf = search_token(inverted_indexes, tokenized_query[i])
        if docs != -1:
            temp_docs_id = pd.DataFrame(docs)
            docs_id = pd.DataFrame(docs_id.append(temp_docs_id))

    if len(docs_id) < 1:
        print("It looks like there aren't many great matches for your search")
        return -1

    docs_id = docs_id[0].value_counts().reset_index()
    docs_id.columns = ['id', 'rank']
    df_res = print_res(df, docs_id)
    return df_res


def ranked_search(df, inverted_indexes, query, doc_length, k, using_heap):
    res = []
    docs_id = pd.DataFrame()
    try:
        using_class = False
        pattern = "cat\:(.*?)\ "
        topic = re.search(pattern, query).group(1)
        if len(topic) > 0:
            print(topic)
            using_class = True
    except AttributeError:
        topic = re.search(pattern, query)

    query = delete_punctuations(query)
    query = delete_stopWords(query)
    query = delete_postfixes(query)
    query = delete_prefixes(query)
    query = replaceArabicWords(query)

    tokenized_query = query.split()
    tokenized_query = replaceWithRoot(tokenized_query)
    tokenized_query = pluralToSingular(tokenized_query)

    if len(tokenized_query) == 0:
        print("It looks like there aren't many great matches for your search")
        return -1

    (unique, counts) = np.unique(tokenized_query, return_counts=True)

    tokenized_query = np.asarray((unique, counts)).T
    # print(tokenized_query)

    doc_score_temp = []
    for i in range (0 ,len(tokenized_query)):
        tf_query = 1 + math.log(int(tokenized_query[i][1]), 10)
        # print(tokenized_query[i][0])
        docs, idf = search_token(inverted_indexes, tokenized_query[i][0])
        # print(docs)


        if docs != -1:
            # print(docs)
            # print(idf)
            w_termInQuery = tf_query*idf
            # print("weight term in q: ", tf_query, idf, w_termInQuery)

            for i in range(len(docs)):
                temp = []
                if using_class == True:
                    if dict_topics[docs[i][0]] == topic:
                        temp.append(docs[i][0])
                        # print("docs[i][1]: ",docs[i][1])
                        temp.append(w_termInQuery*docs[i][1])
                        doc_score_temp.append([docs[i][0],␣
↪w_termInQuery*docs[i][1]])
                else:
                    temp.append(docs[i][0])
                    # print("docs[i][1]: ",docs[i][1])
                    temp.append(w_termInQuery*docs[i][1])
                    doc_score_temp.append([docs[i][0],␣
↪w_termInQuery*docs[i][1]])

    if len(doc_score_temp) < 1:
        print("It looks like there aren't many great matches for your search")
        return -1
    doc_score_temp.sort()
    doc_score = []
    doc_score.append(doc_score_temp[0])
    for i in range(1, len(doc_score_temp)):
        if doc_score[len(doc_score)-1][0] == doc_score_temp[i][0]:
            doc_score[len(doc_score)-1][1] = doc_score[len(doc_score)-1][1] +␣
 ↪doc_score_temp[i][1]
        else:
            doc_score.append(doc_score_temp[i])

    # for i in range(len(doc_score)):
    #     doc_score[i][1] = doc_score[i][1] / doc_length[doc_score[i][0]]

    res = []
    if using_heap == False:
        doc_score.sort(key=lambda x: x[1], reverse=True)
        if len(doc_score) > k:
            for i in range (k):
                res.append(doc_score[i])
        else:
            res = doc_score
    if using_heap == True:
        hp = []
        for e in doc_score:
            hp.append((e[1], e[0]))

        nlargest = heapq.nlargest(k, hp)
        k_high_ranked = []
        for i in nlargest:
            k_high_ranked.append([i[1], i[0]])

        res = k_high_ranked

    docs_id = pd.DataFrame(res, columns=['id', 'rank'])
    res_df = print_res(df, docs_id)
    return res_df

#######################################phase3 Clustering
def k_means(k, iteration):
    #select k centroids randomly
    centroids = []
    randoms = np.random.randint(1, len(inverted_index_perDoc) , size=(k))
    for i in range(len(randoms)):
        centroids.append(inverted_index_perDoc[randoms[i]-1][1])
    # print(centroids)
    max_index_row = 0
    cluster_docs = []
    for it in range(iteration):
        print(it)
        doc_membership = np.zeros((len(df), k))
        for c in range(len(centroids)):
            print("******c= ",c)
            for term in centroids[c]:
            # for key,value in centroids[c].items():
                # print(dict_inverted_indexes)
                docs, idf = search_token(dict_inverted_indexes, term[0])
                if docs != -1:
                    for doc in docs:
                        doc_membership[doc[0]-1][c] =␣
↪doc_membership[doc[0]-1][c] + term[1]*doc[1]
        #update centroids

        temp_index_max = max_index_row
        max_index_row = np.argmax(doc_membership, axis=1)
        if np.array_equal(temp_index_max, max_index_row):
            print("centroids not changed")
            cluster_name = np.argmax(doc_membership, axis=1)
            for c in range(len(centroids)):
                cluster_docs.append(np.where(cluster_name == c))

            return centroids, doc_membership, cluster_docs

        centroids = []
        for i in range(k):
            # print("******k= ",i)
            doc_id_perCluster = np.where(max_index_row == i)
            # print(doc_id_perCluster)
            temp = []
            for doc_id in doc_id_perCluster[0]:
                # print(doc_id)
                # print("hey")
                for term in inverted_index_perDoc[doc_id][1]:
                    temp.append(term)
            centroids.append(temp)
        for i in range(k):
            # data_items = centroids[i].items()
            # data_list = list(data_items)
            df_centroid = pd.DataFrame(centroids[i], columns=['term','weight'])
            df_centroid = df_centroid.groupby('term', as_index=False)['weight'].
↪mean()
            centroids[i] = df_centroid.to_numpy()
            # centroids[i] = dict(centroids[i])
        print(doc_membership)
        cluster_name = np.argmax(doc_membership, axis=1)
        for c in range(len(centroids)):
            cluster_docs.append(np.where(cluster_name == c))

    return centroids, doc_membership, cluster_docs

def ranked_search_withClustering(df, inverted_indexes, query, doc_length, k,␣
 ↪using_heap, b2):
    res = []
    docs_id = pd.DataFrame()

    query = delete_punctuations(query)
    query = delete_stopWords(query)
    query = delete_postfixes(query)
    query = delete_prefixes(query)
    query = replaceArabicWords(query)

    tokenized_query = query.split()
    tokenized_query = replaceWithRoot(tokenized_query)
    tokenized_query = pluralToSingular(tokenized_query)

    if len(tokenized_query) == 0:
        print("It looks like there aren't many great matches for your search")
        return -1

    (unique, counts) = np.unique(tokenized_query, return_counts=True)

    tokenized_query = np.asarray((unique, counts)).T
    # print(tokenized_query)
    doc_score_temp = []
    docs_id = np.array
    for i in range (0 ,len(tokenized_query)):
        tf_query = 1 + math.log(int(tokenized_query[i][1]), 10)
        # print(tokenized_query[i][0])
        docs, idf = search_token(inverted_indexes, tokenized_query[i][0])
        similarityToCentroids = np.zeros(len(centroids))
        w_termInQuery = tf_query*idf
        for c in range(len(centroids)):
            if tokenized_query[i][0] in centroids[c]:
                similarityToCentroids[c] = similarityToCentroids[c] +␣
 ↪w_termInQuery*centroids[c][tokenized_query[i][0]]
    for b in range(b2):
        # print(similarityToCentroids)
        mostSimilar = np.argmax(similarityToCentroids)
        # print(mostSimilar)
        similarityToCentroids[mostSimilar] = 0

        if b == 0:
            docs_id = cluster_docs[mostSimilar][0]
        else:
            docs_id_temp = cluster_docs[mostSimilar][0]
            docs_id = np.append(docs_id, docs_id_temp)

    # print("*****")
    # print(docs_id)
    # print(len(docs_id))

    # print(similarityToCentroids)
    #search in docs
    doc_score_temp = []
    doc_score = []
    for i in range (0 ,len(tokenized_query)):
        # print(tokenized_query[i][0])
        tf_query = 1 + math.log(int(tokenized_query[i][1]), 10)
        # print(tokenized_query[i][0])
        docs, idf = search_token(inverted_indexes, tokenized_query[i][0])

        w_termInQuery = tf_query*idf
        # print(len(docs_id))
        for doc_id in docs_id:
            if tokenized_query[i][0] in inverted_index_perDoc[doc_id][1]:
                # print(doc_id)
                # print(w_termInQuery)
                # print(inverted_index_perDoc[doc_id][1][tokenized_query[i][0]])
                doc_score_temp.append([doc_id + 1,␣
↪inverted_index_perDoc[doc_id][1][tokenized_query[i][0]] * w_termInQuery])
    if len(doc_score_temp) < 1:
        print("It looks like there aren't many great matches for your search")
        return -1

    doc_score_temp.sort()
    # print(doc_score_temp)

    doc_score.append(doc_score_temp[0])
    # print(len(doc_score_temp))
    for i in range(1, len(doc_score_temp)):
        if doc_score[len(doc_score)-1][0] == doc_score_temp[i][0]:
            # print(doc_score_temp[i][0])
            # print(doc_score[len(doc_score)-1][1])
            doc_score[len(doc_score)-1][1] = doc_score[len(doc_score)-1][1] +␣
↪doc_score_temp[i][1]
            # print(doc_score[len(doc_score)-1][1])
        else:
            doc_score.append(doc_score_temp[i])


    # for i in range(len(doc_score)):
    #     doc_score[i][1] = doc_score[i][1] / doc_length[doc_score[i][0]]

    res = []
    if using_heap == False:
        doc_score.sort(key=lambda x: x[1], reverse=True)
        if len(doc_score) > k:
            for i in range (k):
                res.append(doc_score[i])
        else:
            res = doc_score
    if using_heap == True:
        hp = []
        for e in doc_score:
            hp.append((e[1], e[0]))

        nlargest = heapq.nlargest(k, hp)
        k_high_ranked = []
        for i in nlargest:
            k_high_ranked.append([i[1], i[0]])

        res = k_high_ranked


    docs_id = pd.DataFrame(res, columns=['id', 'rank'])
    df_res = print_res(df, docs_id)
    return df_res
```

1.1 Reading datasets and files

[4]:

```
df = read_dataset("datasets/", "IR_Spring2021_ph12_7k.xlsx")
read_files()
```

1.2 Tokenization

[5]:

```
tokens = tokenize(df)
```

1.3 create inverted index per term + champion list

[6]:

```
inverted_indexes = create_inverted_indexes(tokens)
inverted_indexes.pop(0)
inverted_indexes, doc_length = calculate_tfidf(df, inverted_indexes)
#normalize inverted_index
```

```
for i in range(len(inverted_indexes)):
    for j in range(len(inverted_indexes[i][1])):
        inverted_indexes[i][1][j][1] = inverted_indexes[i][1][j][1] /␣
 ↪doc_length[inverted_indexes[i][1][j][0]]

championList = create_championList(inverted_indexes, 4)
# print(inverted_indexes[1])

##create dictionaries
dict_inverted_indexes = []
for i in range(len(inverted_indexes)):
    temp = []
    temp.append(inverted_indexes[i][0])
    temp.append([inverted_indexes[i][1], inverted_indexes[i][2]])
    dict_inverted_indexes.append(temp)
# inverted_indexes = dict(inverted_indexes)
# print(inverted_indexes['  '([
dict_championList = []
for i in range(len(championList)):
    temp = []
    temp.append(championList[i][0])
    temp.append([championList[i][1], championList[i][2]])
    dict_championList.append(temp)

dict_inverted_indexes = dict(dict_inverted_indexes)
dict_championList = dict(dict_championList)
```

1.4 Create inverted index per doc

[ ]:

```
#create inverted_index_perDoc
doc_term = []
for i in range(len(inverted_indexes)):
    for j in range(len(inverted_indexes[i][1])):
        doc_term.append([inverted_indexes[i][1][j][0], inverted_indexes[i][0],␣
 ↪inverted_indexes[i][1][j][1]])

doc_term.sort()
# print(doc_term)
inverted_index_perDoc = []
temp = 0
temp = doc_term[0][0]
temp1 = []
for i in range(0, len(doc_term)):
    # print(inverted_index_perDoc[len(inverted_index_perDoc)][0])
    if temp == doc_term[i][0]:
        temp1.append([doc_term[i][1], doc_term[i][2]])
    else:
        inverted_index_perDoc.append([temp, temp1])
        temp = []
        temp1 = []

        temp = doc_term[i][0]
        temp1.append([doc_term[i][1], doc_term[i][2]])
inverted_index_perDoc.append([temp, temp1])
```

1.5 K-mean clustering

clustering with 5 cluster and 15 iterate

[9]:

```
centroids, doc_membership, cluster_docs = k_means(5, 15)
# convert to dictionary
for i in range(len(inverted_index_perDoc)):
    inverted_index_perDoc[i][1] = dict(inverted_index_perDoc[i][1])

for i in range(len(centroids)):
    centroids[i] = dict(centroids[i])
```

```
0
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.01635833 0.02583013 0.02802454 0.01622359 0.01485935]
 [0.01159337 0.02286786 0.03645791 0.00437282 0.01992944]
 [0.04654571 0.02357023 0.04232842 0.04915135 0.04236345]
 …
 [0.01669898 0.02484786 0.01179503 0.01239389 0.0133338 ]
 [0.0333959  0.0074656  0.00885971 0.04210428 0.01265786]
 [0.01553063 0.00414809 0.00892014 0.049225   0.00836284]]
1
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.88035023 0.69198294 1.03850731 0.6418104  0.78024407]
 [0.35471667 0.37662604 0.77069784 0.36872201 0.38244441]
 [0.85173088 0.82224719 0.85716869 0.79972436 0.70059218]
 …
 [0.99424323 1.24051999 0.86913889 0.74935537 0.74527044]
 [0.56814908 0.52126303 0.51272382 0.81038211 0.4904896 ]
 [0.62440987 0.51963553 0.48594854 0.71332696 0.51860228]]
2
******c=  0
```

```
******c=  1
******c=  2
******c=  3
******c=  4
[[0.84113325 0.70788066 1.04340338 0.59689236 0.80965285]
 [0.36211233 0.37133181 0.77006517 0.34196482 0.40417511]
 [0.80400301 0.83844831 0.92106841 0.61530901 0.72656924]
 …
 [0.97193718 1.25735747 0.86035644 0.73246534 0.75302122]
 [0.53246148 0.53363319 0.5165028  0.82836371 0.50670292]
 [0.61116631 0.52862286 0.51122655 0.72907053 0.54173564]]
3
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83619791 0.70404543 1.04316513 0.59722589 0.81826595]
 [0.36021471 0.37099132 0.75669801 0.34382918 0.40039877]
 [0.79949743 0.83838301 0.91511888 0.61593661 0.73580023]
 …
 [0.95769318 1.25753766 0.86795976 0.73309708 0.77244315]
 [0.52665321 0.53343521 0.51431524 0.82873412 0.54196879]
 [0.60635215 0.52416524 0.50281577 0.7291016  0.5748234 ]]
4
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.8341222  0.70316993 1.04242152 0.59725407 0.82136413]
 [0.35928717 0.37076605 0.75045798 0.34402155 0.38920532]
 [0.7984286  0.83693448 0.91212598 0.61598368 0.72926381]
 …
 [0.95906803 1.25681547 0.8668664  0.73237354 0.78913129]
 [0.52508513 0.53285207 0.52004646 0.82866179 0.54544739]
 [0.60278588 0.52396042 0.51108741 0.7290383  0.60675043]]
5
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83875412 0.70196989 1.03753988 0.59725407 0.82116002]
 [0.35936783 0.37072368 0.74679513 0.34402155 0.38894928]
 [0.79910575 0.8364026  0.90928555 0.61598368 0.73080441]
 …
 [0.96162085 1.25668785 0.85660098 0.73237354 0.78846362]
 [0.52552807 0.53281817 0.51626058 0.82866179 0.5456783 ]
 [0.60626789 0.52392723 0.50934039 0.7290383  0.60721567]]
6
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83790513 0.70239846 1.03721116 0.59723606 0.82273217]
 [0.35955527 0.37051181 0.74658994 0.34396527 0.38652987]
 [0.79900405 0.83641639 0.90889513 0.61571852 0.73258933]
 …
 [0.9613691  1.25673103 0.85648941 0.73222957 0.78830774]
 [0.52527035 0.53305737 0.51623631 0.82864243 0.54500894]
 [0.60613837 0.52457405 0.50921913 0.72895034 0.60672468]]
7
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.837503   0.70239846 1.03720681 0.59723606 0.82351726]
 [0.35952281 0.37051181 0.7466002  0.34396527 0.3864208 ]
 [0.79896925 0.83641639 0.90891646 0.61571852 0.73166773]
 …
 [0.96093533 1.25673103 0.85646989 0.73222957 0.78881185]
 [0.52473845 0.53305737 0.51622169 0.82864243 0.54465672]
 [0.6060831  0.52457405 0.50917871 0.72895034 0.6070605 ]]
8
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83734002 0.70235158 1.03761294 0.59723606 0.82584906]
 [0.35920086 0.37045694 0.74691594 0.34396527 0.38548818]
 [0.7986601  0.8361688  0.90971004 0.61571852 0.72599553]
 …
 [0.96034472 1.25638027 0.85675939 0.73222957 0.78866338]
 [0.52446592 0.53303711 0.51622568 0.82864243 0.54485661]
 [0.60523466 0.52449851 0.50941208 0.72895034 0.60815968]]
9
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83812426 0.70025883 1.0367578  0.59723606 0.82365693]
```

```
 [0.36038415 0.37014187 0.74684886 0.34396527 0.38169926]
 [0.80023001 0.83219373 0.90959127 0.61571852 0.72247973]
 …
 [0.96072133 1.25502863 0.85647823 0.73222957 0.78674747]
 [0.52510378 0.53073637 0.51624061 0.82864243 0.54409341]
 [0.60543127 0.52336626 0.509373   0.72895034 0.60659893]]
10
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83795507 0.70025883 1.0367578  0.59723606 0.82394985]
 [0.36029712 0.37014187 0.74684886 0.34396527 0.38201954]
 [0.8000438  0.83219373 0.90959127 0.61571852 0.72295745]
 …
 [0.96056791 1.25502863 0.85647823 0.73222957 0.78714116]
 [0.52499787 0.53073637 0.51624061 0.82864243 0.5450161 ]
 [0.60529663 0.52336626 0.509373   0.72895034 0.60710539]]
11
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83782894 0.70025883 1.03668322 0.59723606 0.82452064]
 [0.35987105 0.37014187 0.74670073 0.34396527 0.38266862]
 [0.79925469 0.83219373 0.90952894 0.61571852 0.72604255]
 …
 [0.9603122  1.25502863 0.8564541  0.73222957 0.78819825]
 [0.52454107 0.53073637 0.51623946 0.82864243 0.5469786 ]
 [0.60502849 0.52336626 0.50937192 0.72895034 0.60763627]]
12
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83719289 0.69932311 1.03668322 0.59723606 0.82535048]
 [0.35850731 0.36962176 0.74670073 0.34396527 0.38457691]
 [0.79844156 0.83098886 0.90952894 0.61571852 0.72774917]
 …
 [0.96576528 1.25490316 0.8564541  0.73222957 0.81045362]
 [0.52313815 0.53038378 0.51623946 0.82864243 0.55037687]
 [0.60502488 0.52310169 0.50937192 0.72895034 0.606833  ]]
13
******c=  0
******c=  1
```

```
******c=  2
******c=  3
******c=  4
[[0.83664449 0.6992957  1.03637782 0.59750278 0.81746341]
 [0.35479847 0.36956068 0.74638005 0.34404704 0.38718833]
 [0.79574011 0.83091403 0.90865681 0.61573794 0.76134762]
 …
 [0.96426081 1.25460841 0.85609673 0.73221642 0.81366268]
 [0.52185876 0.53029954 0.51616906 0.8285756  0.54956583]
 [0.60353286 0.52306486 0.50932801 0.72870639 0.63263472]]
14
******c=  0
******c=  1
******c=  2
******c=  3
******c=  4
[[0.83516382 0.6990089  1.03646053 0.58334994 0.8333959 ]
 [0.353869   0.36955436 0.74641657 0.34405352 0.38554493]
 [0.79564791 0.83041071 0.90860593 0.61463313 0.75882708]
 …
 [0.95996653 1.25403307 0.8560517  0.72058642 0.82114127]
 [0.52178909 0.53011333 0.51610388 0.82860823 0.54011899]
 [0.60183174 0.52303009 0.50934498 0.72860253 0.62148114]]
```

[34]:

```
query = "           "

# binary search
print("binary search")
print(binary_search(df, dict_inverted_indexes, query))
# rank search using all posting lists
print("rank search using all posting lists")
print(ranked_search(df, dict_inverted_indexes, query, doc_length, k = 10,␣
 ↪using_heap = True))
# rank search using champion list
print("rank search using champion list")
print(ranked_search(df, dict_championList, query, doc_length, k = 10,␣
 ↪using_heap = True))
# rank search using clustering to pure some unrelated docs
print("rank search using clustering")
print(ranked_search_withClustering(df, dict_inverted_indexes, query,␣
 ↪doc_length, k = 10, using_heap = True, b2 = 2))

query = "cat:health           "
print("rank search with classification.\n categories: 'sport', 'politics',␣
 ↪'economy', 'health', 'culture'")
print(ranked_search(df, dict_inverted_indexes, query, doc_length, k = 10,␣
 ↪using_heap = True))
```

```
binary search
        id  rank                                                url
793   6385     3  https://www.isna.ir/news/99111208510/ …    -
351   5535     3  https://www.isna.ir/news/99020906382/    --…
307   5453     3  https://www.isna.ir/news/99011407171/  …  -
606   6020     3  https://www.isna.ir/news/99062720787/   … -
567   5946     3  https://www.isna.ir/news/99061107994/  -…  -
…     …    …                                              …
357   5548     1  https://www.isna.ir/news/99021410284/   … -
358   5549     1  https://www.isna.ir/news/99021410064/  …  -
359   5550     1  https://www.isna.ir/news/99021410020/    - …
360   5554     1  https://www.isna.ir/news/99021510817/  …   -
1045  7000     1  https://www.isna.ir/news/98092015327/  …  -

1046] rows x 3 columns]
rank search using all posting lists
     id      rank                                               url
2  5453  0.699405  https://www.isna.ir/news/99011407171/  …   -
3  5461  0.699405  https://www.isna.ir/news/99011407171/  …   -
8  6886  0.368826  https://www.isna.ir/news/98071813768/  …   -
9  6889  0.368826  https://www.isna.ir/news/98071813768/  …   -
5  5946  0.353189  https://www.isna.ir/news/99061107994/  -…  -
6  6013  0.351518  https://www.isna.ir/news/99062720787/   … -
7  6020  0.351518  https://www.isna.ir/news/99062720787/   … -
1  4366  0.344058  https://www.isna.ir/news/98012912873/  …  -
0  1606  0.334954  https://www.isna.ir/news/98111309170/   -… -
4  5535  0.331943  https://www.isna.ir/news/99020906382/    --…
rank search using champion list
     id      rank                                               url
4  5453  0.452051  https://www.isna.ir/news/99011407171/  …   -
5  5461  0.452051  https://www.isna.ir/news/99011407171/  …   -
1  1606  0.334954  https://www.isna.ir/news/98111309170/   -… -
2  1921  0.292579  https://www.isna.ir/news/99050100921/  …   -
6  5904  0.275321  https://www.isna.ir/news/99052719974/   -… -
7  6606  0.256339  https://www.isna.ir/news/98012510503/  …   -
3  2913  0.254779  https://www.isna.ir/news/98071511860/   … -
0   756  0.248147  https://www.isna.ir/news/99091814045/    -…
8  6626  0.120213  https://www.isna.ir/news/98022110734/  …   -
9  6629  0.120213  https://www.isna.ir/news/98022110734/  …   -
rank search using clustering
     id      rank                                               url
8  6886  0.368826  https://www.isna.ir/news/98071813768/  …   -
9  6889  0.368826  https://www.isna.ir/news/98071813768/  …   -
5  1606  0.334954  https://www.isna.ir/news/98111309170/   -… -
7  1921  0.292579  https://www.isna.ir/news/99050100921/  …   -
6  1698  0.229151  https://www.isna.ir/news/98121713481/ …    -
4   972  0.225456  https://www.isna.ir/news/99120402801/  …   -
0   597  0.208519  https://www.isna.ir/news/99072820710/   …  -
```

```
3   819  0.203495  https://www.isna.ir/news/99101612434/ …   -
1   600  0.197149  https://www.isna.ir/news/99072921619/    --…
2   607  0.196446  https://www.isna.ir/news/99080100388/ …   -
rank search with classification.
 categories: 'sport', 'politics', 'economy', 'health', 'culture'
health
     id      rank                                              url
0  5453  0.699405  https://www.isna.ir/news/99011407171/  …  -
1  5461  0.699405  https://www.isna.ir/news/99011407171/  …  -
8  6886  0.368826  https://www.isna.ir/news/98071813768/  …  -
9  6889  0.368826  https://www.isna.ir/news/98071813768/  …  -
4  5946  0.353189  https://www.isna.ir/news/99061107994/  -…  -
5  6013  0.351518  https://www.isna.ir/news/99062720787/   … -
6  6020  0.351518  https://www.isna.ir/news/99062720787/   … -
2  5535  0.331943  https://www.isna.ir/news/99020906382/    --…
3  5738  0.316212  https://www.isna.ir/news/99041410234/   … -
7  6681  0.311661  https://www.isna.ir/news/98032712929/    … -
```

1.6 Labeling datasets Using KNN

:[5]

```
df_11k = read_dataset("datasets/", "IR00_3_11k News.xlsx")
df_17k = read_dataset("datasets/", "IR00_3_17k News.xlsx")
df_20k = read_dataset("datasets/", "IR00_3_20k News.xlsx")
df_50k = pd.concat([df_11k, df_17k, df_20k])
```

[11]:

```
df_50k['id']  = range(1, 1 + len(df_50k))
# duplicates_loaded = len(df_50k[df_50k.duplicated(subset=['url', 'content',␣
 ↪'topic'])])
# print(duplicates_loaded)
```

[13]:

```
df_50k = df_50k.reset_index(drop=True)
```

[16]:

```
tokens_50k = tokenize(df_50k)
```

[17]:

```
len(tokens_50k)
```

[17]:

```
16898010
```

[18]:

```
inverted_indexes_50k = create_inverted_indexes(tokens_50k)
```

[19]:

```
inverted_indexes_50k.pop(0)
inverted_indexes_50k, doc_length_50k = calculate_tfidf(df_50k,␣
 ↪inverted_indexes_50k)
#normalize inverted_index
for i in range(len(inverted_indexes_50k)):
    for j in range(len(inverted_indexes_50k[i][1])):
        inverted_indexes_50k[i][1][j][1] = inverted_indexes_50k[i][1][j][1] /␣
 ↪doc_length_50k[inverted_indexes_50k[i][1][j][0]]
```

[19]: 243026

[22]:
```
dict_inverted_indexes_50k = []
for i in range(len(inverted_indexes_50k)):
    temp = []
    temp.append(inverted_indexes_50k[i][0])
    temp.append([inverted_indexes_50k[i][1], inverted_indexes_50k[i][2]])
    dict_inverted_indexes_50k.append(temp)
# inverted_indexes = dict(inverted_indexes)
# print(inverted_indexes['  '([
# dict_championList = []
# for i in range(len(championList)):
#     temp = []
#     temp.append(championList[i][0])
#     temp.append([championList[i][1], championList[i][2]])
#     dict_championList.append(temp)

dict_inverted_indexes_50k = dict(dict_inverted_indexes_50k)
# dict_championList = dict(dict_championList)
```

[57]:
```
# df_50k.loc[df_50k['topic'] == 'sports'] = df_50k.loc[df_50k['topic'] ==␣
 ↪'sports'].topic.replace('sport')
df_50k['topic'] = df_50k['topic'].replace('sports','sport')
df_50k['topic'] = df_50k['topic'].replace('political','politics')
```

[57]:
```
array(['sport', 'politics', 'economy', 'health', 'culture'], dtype=object)
```

[55]:
```
df_50k
```

[56]:
```
len(dict_inverted_indexes_50k)
```

[56]:
```
          id                                              content     topic  \
0          1                                  …       sport
1          2                                   …      sport
2          3                                …         sport
3          4                            …          sport
4          5                                (…      sport
…        …                                                  …        …
50056  50057                          …     ]  politics
50057  50058                               …   politics
50058  50059                                    …  politics
50059  50060                               …     politics
50060  50061                               …   politics

                                                    url
0      https://www.isna.ir/news/99010100077/    --…
1      https://www.isna.ir/news/98122922468/   -… -
```

```
2      https://www.isna.ir/news/99010200541/     -…
3      https://www.isna.ir/news/99010200528/    … -
4      https://www.isna.ir/news/99010200510/   …  -
…                                                   …
50056  https://www.farsnews.ir/news/13990612000494/…
50057  https://www.farsnews.ir/news/13990612000397/…
50058  https://www.farsnews.ir/news/13990612000425/…
50059  https://www.farsnews.ir/news/13990612000435/…
50060  https://www.farsnews.ir/news/13990611001190/…

50061] rows x 4 columns]
```

1.7 KNN

label our unlabeled datasets using another datasets labels using KNN classification algorithm

[88]:

```
%%time
dict_topics = {}
for i in range(len(df):
    query = df.content[i]
    # print(query)
    res = ranked_search(df_50k, dict_inverted_indexes_50k, query,␣
 ↪doc_length_50k, k = 9, using_heap = True)
    className = res['topic'].value_counts().idxmax()
    dict_topics[i+1] = className
    if i%1000 == 0:
        print(i)
```

```
5300
5400
5500
5600
5700
5800
5900
6000
6100
6200
6300
6400
6500
6600
6700
6800
6900
Wall time: 18min 30s
```

[99]:

```
dict_topics[2500]
```

```
[99]: 'economy'
```

```
[141]: path = 'files/'
a_file = open(path + "IR_Spring2021_ph12_7k_lablesDict.pkl", "wb")
pickle.dump(dict_topics, a_file)
a_file.close()
```

```
[28]: path = 'files/'
a_file = open(path + "IR_Spring2021_ph12_7k_lablesDict.pkl", "rb")
dict_topics = pickle.load(a_file)
```

```
[29]: dict_topics[1]
```

```
[29]: 'sport'
```

```
[30]: # duplicates_loaded = len(df_50k[df_50k.duplicated(subset=['url'])])
# print(duplicates_loaded)
```

1.8 Queries and results

```
[35]: query = "            "

# binary search
print("binary search")
print(binary_search(df, dict_inverted_indexes, query))
# rank search using all posting lists
print("rank search using all posting lists")
print(ranked_search(df, dict_inverted_indexes, query, doc_length, k = 10,␣
 ↪using_heap = True))
# rank search using champion list
print("rank search using champion list")
print(ranked_search(df, dict_championList, query, doc_length, k = 10,␣
 ↪using_heap = True))
# rank search using clustering to pure some unrelated docs
print("rank search using clustering")
print(ranked_search_withClustering(df, dict_inverted_indexes, query,␣
 ↪doc_length, k = 10, using_heap = True, b2 = 2))

query = "cat:health            "
print("rank search with classification.\n categories: 'sport', 'politics',␣
 ↪'economy', 'health', 'culture'")
print(ranked_search(df, dict_inverted_indexes, query, doc_length, k = 10,␣
 ↪using_heap = True))
```

```
binary search
        id  rank                                                url
793   6385     3  https://www.isna.ir/news/99111208510/ …   -
351   5535     3  https://www.isna.ir/news/99020906382/    --…
```

```
307   5453     3  https://www.isna.ir/news/99011407171/  …  -
606   6020     3  https://www.isna.ir/news/99062720787/   … -
567   5946     3  https://www.isna.ir/news/99061107994/  -…  -
…     …    …                                              …
357   5548     1  https://www.isna.ir/news/99021410284/   … -
358   5549     1  https://www.isna.ir/news/99021410064/  …  -
359   5550     1  https://www.isna.ir/news/99021410020/    - …
360   5554     1  https://www.isna.ir/news/99021510817/  …  -
1045  7000     1  https://www.isna.ir/news/98092015327/   …  -

1046] rows x 3 columns]
rank search using all posting lists
     id      rank                                      url
2  5453  0.699405  https://www.isna.ir/news/99011407171/  …  -
3  5461  0.699405  https://www.isna.ir/news/99011407171/  …  -
8  6886  0.368826  https://www.isna.ir/news/98071813768/  …  -
9  6889  0.368826  https://www.isna.ir/news/98071813768/  …  -
5  5946  0.353189  https://www.isna.ir/news/99061107994/  -…  -
6  6013  0.351518  https://www.isna.ir/news/99062720787/   … -
7  6020  0.351518  https://www.isna.ir/news/99062720787/   … -
1  4366  0.344058  https://www.isna.ir/news/98012912873/   …  -
0  1606  0.334954  https://www.isna.ir/news/98111309170/    -… -
4  5535  0.331943  https://www.isna.ir/news/99020906382/    --…
rank search using champion list
     id      rank                                      url
4  5453  0.452051  https://www.isna.ir/news/99011407171/  …  -
5  5461  0.452051  https://www.isna.ir/news/99011407171/  …  -
1  1606  0.334954  https://www.isna.ir/news/98111309170/    -… -
2  1921  0.292579  https://www.isna.ir/news/99050100921/  …  -
6  5904  0.275321  https://www.isna.ir/news/99052719974/    -… -
7  6606  0.256339  https://www.isna.ir/news/98012510503/  …  -
3  2913  0.254779  https://www.isna.ir/news/98071511860/   … -
0   756  0.248147  https://www.isna.ir/news/99091814045/     -…
8  6626  0.120213  https://www.isna.ir/news/98022110734/  …  -
9  6629  0.120213  https://www.isna.ir/news/98022110734/  …  -
rank search using clustering
     id      rank                                      url
8  6886  0.368826  https://www.isna.ir/news/98071813768/  …  -
9  6889  0.368826  https://www.isna.ir/news/98071813768/  …  -
5  1606  0.334954  https://www.isna.ir/news/98111309170/    -… -
7  1921  0.292579  https://www.isna.ir/news/99050100921/  …  -
6  1698  0.229151  https://www.isna.ir/news/98121713481/ …   -
4   972  0.225456  https://www.isna.ir/news/99120402801/  …  -
0   597  0.208519  https://www.isna.ir/news/99072820710/   … -
3   819  0.203495  https://www.isna.ir/news/99101612434/ …    -
1   600  0.197149  https://www.isna.ir/news/99072921619/    --…
2   607  0.196446  https://www.isna.ir/news/99080100388/ …    -
rank search with classification.
```

```
 categories: 'sport', 'politics', 'economy', 'health', 'culture'
health
     id      rank                                               url
0  5453  0.699405  https://www.isna.ir/news/99011407171/  …  -
1  5461  0.699405  https://www.isna.ir/news/99011407171/  …  -
8  6886  0.368826  https://www.isna.ir/news/98071813768/  …  -
9  6889  0.368826  https://www.isna.ir/news/98071813768/  …  -
4  5946  0.353189  https://www.isna.ir/news/99061107994/  -…  -
5  6013  0.351518  https://www.isna.ir/news/99062720787/   … -
6  6020  0.351518  https://www.isna.ir/news/99062720787/   … -
2  5535  0.331943  https://www.isna.ir/news/99020906382/    --…
3  5738  0.316212  https://www.isna.ir/news/99041410234/   … -
7  6681  0.311661  https://www.isna.ir/news/98032712929/    … -
```

```
] ]:
```