Ανάκτηση πληροφορίας

Αντωνίου Σωτήριος 1067512 6ο έτος
ε-μαιλ: basotos@hotmail.com

up1067492@ac.upatras.gr

γλώσσα προγραμματισμού : python
περιβάλλον: pycharm
βιβλιοθήκες : aiohttp==3.9.5

```
aiosignal==1.3.1
async-timeout==4.0.3
attrs==23.2.0
bitarray==2.9.2
blinker==1.8.2
certifi==2024.6.2
charset-normalizer==3.3.2
click==8.1.7
-e
git+https://github.com/stanford-futuredata/ColBERT.git@a45bc64985ffecf55ba710
6ed78df8b33f783f73#egg=colbert_ai
datasets==2.20.0
dill==0.3.8
faiss-cpu==1.8.0.post1
filelock==3.15.1
Flask==3.0.3
frozenlist==1.4.1
fsspec==2024.5.0
git-python==1.0.3
gitdb==4.0.11
GitPython==3.1.43
huggingface-hub==0.23.4
icalendar==5.0.13
idna==3.7
importlib_metadata==7.1.0
itsdangerous==2.2.0
Jinja2==3.1.4
joblib==1.4.2
MarkupSafe==2.1.5
multidict==6.0.5
multiprocess==0.70.16
ninja==1.11.1.1
nltk==3.8.1
numpy==1.26.4
packaging==24.1
pandas==2.2.2
pyarrow==16.1.0
pyarrow-hotfix==0.6
python-dateutil==2.9.0.post0
```

```
python-dotenv==1.0.1
pytz==2024.1
PyYAML==6.0.1
regex==2024.5.15
requests==2.32.3
safetensors==0.4.3
scipy==1.13.1
six==1.16.0
smmap==5.0.1
threadpoolctl==3.5.0
tokenizers==0.19.1
torch==1.13.1
tqdm==4.66.4
transformers==4.41.2
typing_extensions==4.12.2
tzdata==2024.1
ujson==5.10.0
urllib3==2.2.2
Werkzeug==3.0.3
xxhash==3.4.1
yarl==1.9.4
zipp==3.19.2
```

Άσκηση 1:

script1.txt

```
import os
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from collections import defaultdict

# Ensure you have the necessary NLTK data files
nltk.download('stopwords')
nltk.download('punkt')

def preprocess_text(text):
    text = re.sub(r'\W', ' ', text)
    text = text.lower()
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in
stopwords.words('english')]
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]
    return tokens

def build_inverted_index(documents):
```

```
    inverted_index = defaultdict(list)
    for doc_id, content in documents.items():
        tokens = preprocess_text(content)
        for token in tokens:
            inverted_index[token].append(doc_id)
    return inverted_index

def read_documents(directory):
    documents = {}
    for filename in os.listdir(directory):
        filepath = os.path.join(directory, filename)
        with open(filepath, 'r', encoding='utf-8') as file:
            documents[filename] = file.read()
    return documents

# Usage
document_directory = 'Collection/docs'
documents = read_documents(document_directory)
inverted_index = build_inverted_index(documents)

# Debugging: Print some of the inverted index entries
for term, doc_ids in list(inverted_index.items())[:10]:  # Print the first 10
terms and their associated doc IDs
    print(f"Term: {term}, Document IDs: {doc_ids}")

# Save the inverted index to a file to inspect it manually
with open('inverted_index.txt', 'w', encoding='utf-8') as f:
    for term, doc_ids in inverted_index.items():
        f.write(f"Term: {term}, Document IDs: {doc_ids}\n")
```

ο παραπάνω κώδικας θα δημιουργήσει ανεστραμένο ευρετήριο και θα το αποθηκεύσει σε ενα .txt file με το όνομα inverted_index.txt παρολο που είναι πολύ μεγάλο αρχειο για να παραθέσω οριστε οι πρωτες 3 γραμμές

```
Term: dri, Document IDs: ['01128', '01128', '00261', '00261', '00261',
'00261', '00805', '00604', '00379', '00976', '00976', '01014', '01219',
'00425', '00425', '00425', '00425', '01133', '00827', '00600', '00522',
'00380', '00380', '00303', '00303', '00303', '01028', '01028', '00137',
'00993', '00417', '00876']
Term: blood, Document IDs: ['01128', '01128', '01126', '00804', '00804',
'00804', '00804', '00235', '00235', '00235', '00235', '00235', '00006',
'00006', '01127', '00030', '00256', '00256', '01189', '00251', '00055',
'01180', '00436', '00436', '00436', '00436', '00924', '00924', '00588',
'00173', '00313', '00313', '00181', '00378', '00985', '00985', '00371',
'00150', '01085', '01217', '00556', '00135', '01221', '00707', '00167',
'01046', '00568', '00753', '00414', '01156', '00874', '01151', '00482',
'00643', '00023', '00023', '00023', '00844', '00843', '00843', '00014',
'01140', '00253', '00253', '00896', '00896', '01178', '00209', '00236',
'00236', '00460', '00458', '00458', '00458', '00263', '00435', '00230',
'00230', '00230', '00230', '00230', '00691', '00034', '00374', '00374',
'01058', '01058', '00148', '00316', '00316', '00943', '01238', '01238',
'01238', '00583', '00122', '00317', '00317', '00751', '00751', '00751',
'00751', '00107', '00992', '00165', '00165', '00539', '01075', '00792',
```

```
'00792', '00792', '00553', '00553', '00553', '00553', '00757', '00562',
'01212', '00509', '00199', '00199', '00199', '00360', '00360', '00360',
'01073', '01073', '00393', '00393', '00393', '00393', '00393', '00393',
'00393', '00958', '00958', '01130', '01130', '00020', '00622', '00613',
'00613', '00613', '00613', '00613', '00426', '00426', '00270', '00270',
'00044']
Term: spot, Document IDs: ['01128', '01128', '01126', '01133']
```

Άσκηση 2:

rawscript2detailed.txt

logscript2.txt
script2skcheck.txt

κώδικας:

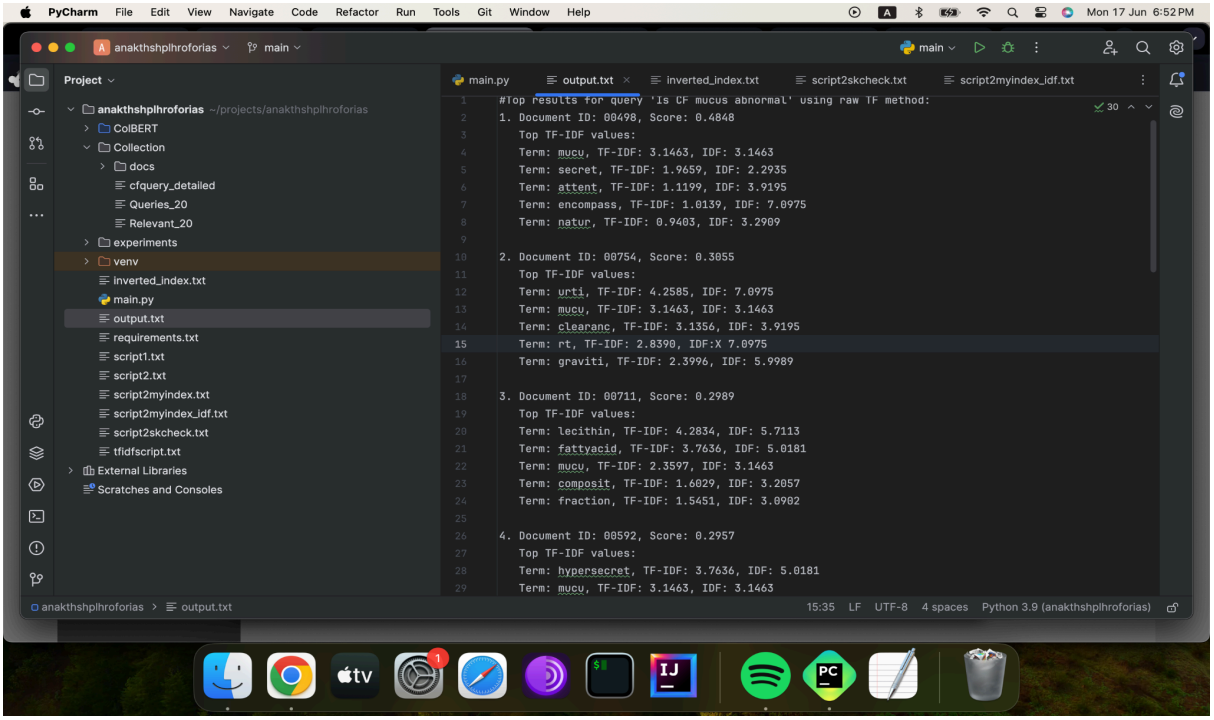
```
import os
import re
import math
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from collections import defaultdict

# Ensure we have the necessary NLTK data files
nltk.download('stopwords')
nltk.download('punkt')


# Function to preprocess text
def preprocess_text(text):
   text = re.sub(r'\W', ' ', text)
   text = text.lower()
   tokens = nltk.word_tokenize(text)
   tokens = [word for word in tokens if word not in
stopwords.words('english')]
   stemmer = PorterStemmer()
   tokens = [stemmer.stem(word) for word in tokens]
   return tokens


# Function to read documents from a directory
def read_documents(directory):
   documents = {}
```

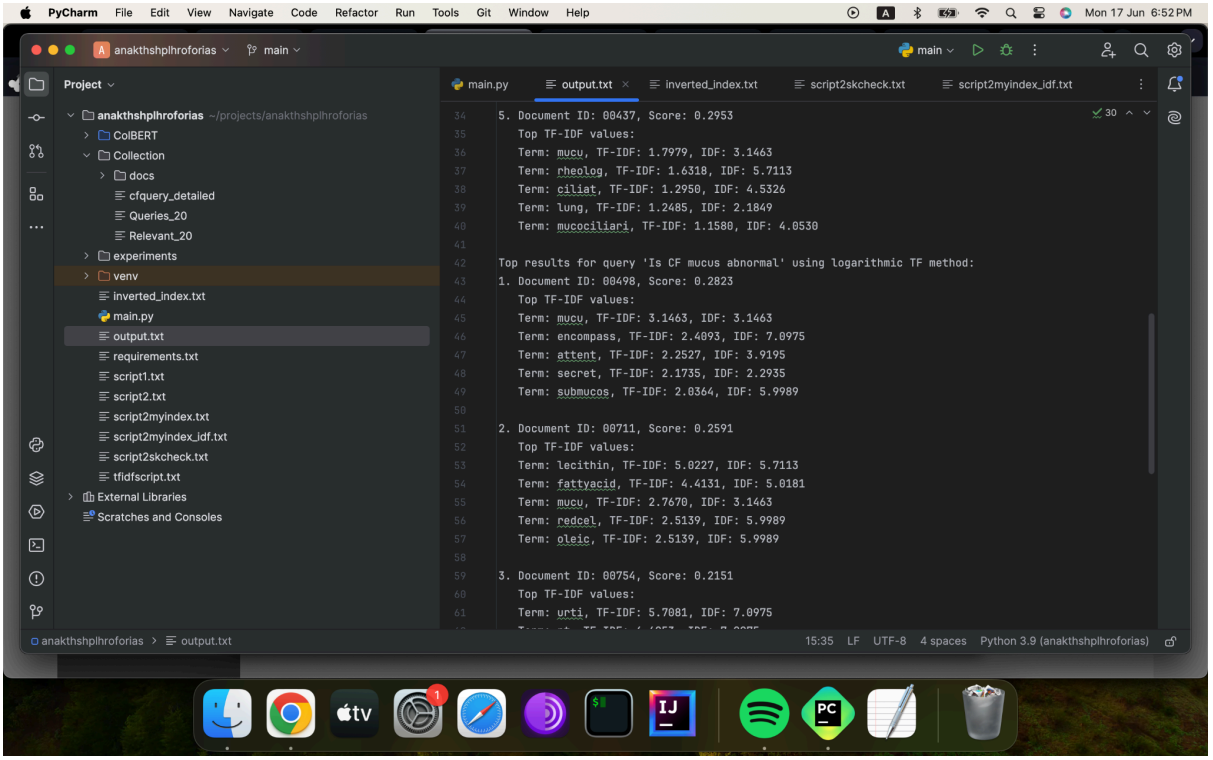
```
    for filename in os.listdir(directory):
        filepath = os.path.join(directory, filename)
        with open(filepath, 'r', encoding='utf-8') as file:
            documents[filename] = file.read()
    return documents


# Function to read inverted index from a file
def read_inverted_index(file_path):
    inverted_index = defaultdict(list)
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip()
            if line.startswith('Term: '):
                parts = line.split(', Document IDs: ')
                term = parts[0].replace('Term: ', '').strip()
                doc_ids_str = parts[1].replace('[', '').replace(']',
'').strip()
                doc_ids = [doc_id.strip() for doc_id in doc_ids_str.split(',
')]
                inverted_index[term] = doc_ids
    return inverted_index


# Function to calculate TF-IDF for documents with a given TF method
def calculate_tf_idf(documents, inverted_index, tf_method='raw'):
    N = len(documents)
    tfidf = defaultdict(lambda: defaultdict(float))
    idf = {}

    # Calculate IDF for each term
    for term, postings in inverted_index.items():
        ni = len(set(postings))
        idf[term] = math.log(N / ni) if ni > 0 else 0.0

    # Calculate TF-IDF for each term in each document
    for doc_id, content in documents.items():
        tokens = preprocess_text(content)
        tf = defaultdict(float)
        for term in tokens:
            tf[term] += 1.0  # Simple term frequency

        if tf_method == 'logarithmic':
            for term in tf:
                tf[term] = 1 + math.log(tf[term])

        max_tf = max(tf.values(), default=1.0)
        for term in tokens:
            tfidf[doc_id][term] = (tf[term] / max_tf) * idf.get(term, 0.0)

    return tfidf, idf
```

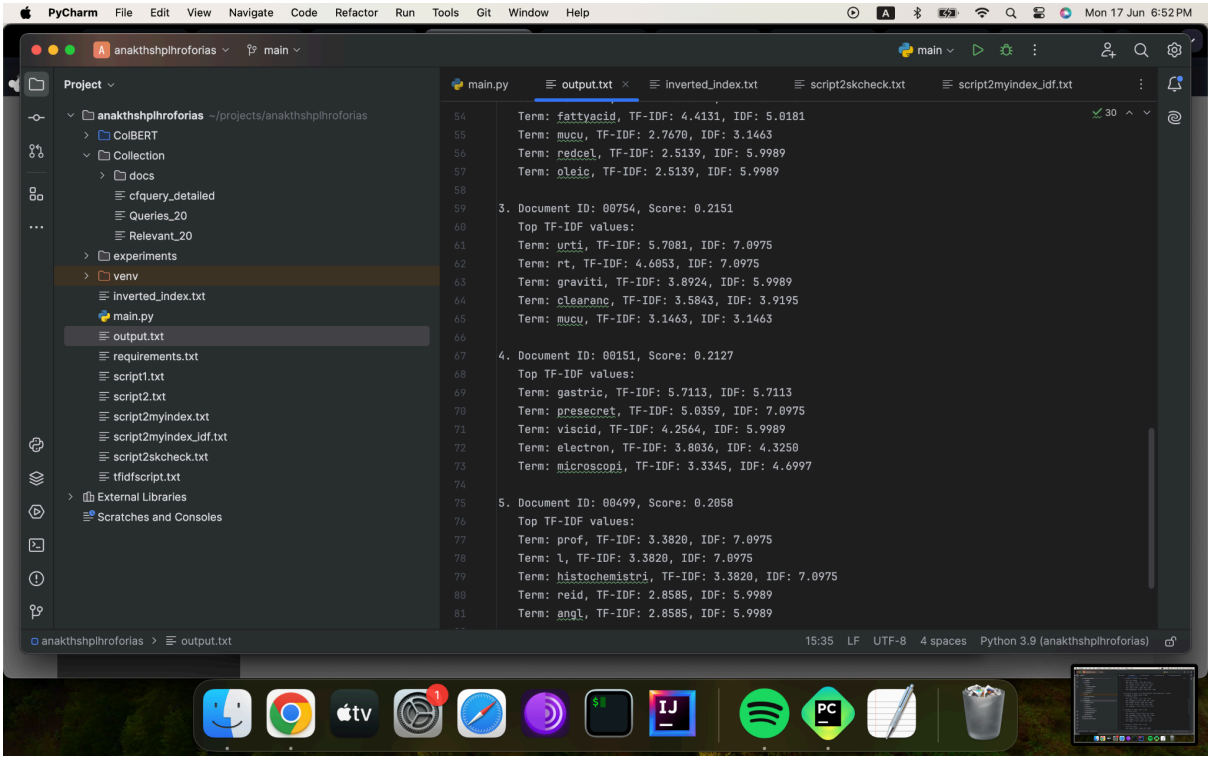
```
# Function to execute a query and retrieve top results
def execute_query(query, documents, inverted_index, tfidf, idf, tf_method):
    query_tokens = preprocess_text(query)
    scores = defaultdict(float)
    query_vector = defaultdict(float)

    # Calculate query vector (simple term frequency)
    for term in query_tokens:
        query_vector[term] += 1.0

    if tf_method == 'logarithmic':
        for term in query_vector:
            query_vector[term] = 1 + math.log(query_vector[term])

    # Calculate TF-IDF for query vector
    max_tf_query = max(query_vector.values(), default=1.0)
    for term in query_tokens:
        query_vector[term] = (query_vector[term] / max_tf_query) *
idf.get(term, 0.0)

    # Calculate cosine similarity between query vector and document vectors
    query_norm = sum(query_vector[term] ** 2 for term in query_vector)
    query_norm = math.sqrt(query_norm) if query_norm != 0 else 1  # Normalize
query vector

    for doc_id in documents:
        doc_vector = defaultdict(float)
        for term, weight in tfidf[doc_id].items():
            doc_vector[term] = weight

        # Calculate document vector norm
        doc_norm = sum(doc_vector[term] ** 2 for term in doc_vector)
        doc_norm = math.sqrt(doc_norm) if doc_norm != 0 else 1  # Normalize
document vector

        # Calculate dot product
        dot_product = sum(query_vector[term] * doc_vector[term] for term in
query_tokens if term in doc_vector)

        # Calculate cosine similarity
        cosine_similarity = dot_product / (query_norm * doc_norm) if
query_norm * doc_norm != 0 else 0
        scores[doc_id] = cosine_similarity

    # Sort scores in descending order
    sorted_scores = sorted(scores.items(), key=lambda x: x[1], reverse=True)

    # Print top results
    print(f"Top results for query '{query}' using {tf_method} TF method:")
    for rank, (doc_id, score) in enumerate(sorted_scores[:5], start=1):
        print(f"{rank}. Document ID: {doc_id}, Score: {score:.4f}")

        # Print top TF-IDF values for the document
```

```
        sorted_tfidf = sorted(tfidf[doc_id].items(), key=lambda x: x[1],
reverse=True)[:5]
        print("   Top TF-IDF values:")
        for term, tfidf_value in sorted_tfidf:
            print(f"   Term: {term}, TF-IDF: {tfidf_value:.4f}, IDF:
{idf.get(term, 0.0):.4f}")

        print()  # Print an empty line for separation

    return sorted_scores


if __name__ == "__main__":
    # Directory containing documents
    documents_directory = 'collection/docs'

    # Read documents
    documents = read_documents(documents_directory)

    # File path for inverted index
    inverted_index_file = 'inverted_index.txt'

    # Read inverted index from file
    inverted_index = read_inverted_index(inverted_index_file)

    # Calculate TF-IDF for documents using raw count TF method
    tfidf_raw, idf_raw = calculate_tf_idf(documents, inverted_index,
tf_method='raw')

    # Calculate TF-IDF for documents using logarithmic TF method
    tfidf_log, idf_log = calculate_tf_idf(documents, inverted_index,
tf_method='logarithmic')

    # Example query
    query = "Is CF mucus abnormal"

    # Execute the query and retrieve top results for raw count TF method
    top_results_raw = execute_query(query, documents, inverted_index,
tfidf_raw, idf_raw, tf_method='raw')

    # Execute the query and retrieve top results for logarithmic TF method
    top_results_log = execute_query(query, documents, inverted_index,
tfidf_log, idf_log, tf_method='logarithmic')
```

ο παραπάνω κώδικας συγκρίνει τις λέξεις στο query που δώσαμε (στο συγκεκριμενο παραδειγμα δινω το: `"Is CF mucus abnormal"` και βρίσκει τα ντοξ με το μεγαλύτερο ποσοστό ομοιότητας. Για τον υπολογισμο του tf χρησιμοποιησα και το raw count και την λογαριθμικη μεθοδο, αφου τα αρχεια ειναι σχετικά απλα (100-200) λέξεις και πολλα (1200+) αλλα αφου μιλάμε για ιατρικά αρχεία δεν θέλουμε οι κοινότυπες λέξεις να μας ανεβάσουν πάρα πολύ την ομοιότητα, αφου συνήθως οι σπάνιες λεξεις ειναι αυτές που κανουν την διαφορά. Για την idf μέθοδο επέλεξα την απλή αναστροφή συχνότητας εμφάνισης, αφου σιγουρεύτηκα οτι η διαίρεση δεν θα γίνει ποτέ με το 0. Τα αποτελέσματα είναι παρακάτω


PyCharm File Edit View Navigate Code Refactor Run Tools Git Window Help
Mon 17 Jun 6:52 PM
anakthshplhroforias main
Project
anakthshplhroforias ~/projects/anakthshplhroforias
ColBERT
Collection
docs
cfquery_detailed
Queries_20
Relevant_20
experiments
venv
inverted_index.txt
main.py
output.txt
requirements.txt
script1.txt
script2.txt
script2myindex.txt
script2myindex_idf.txt
script2skcheck.txt
tfidfscript.txt
External Libraries
Scratches and Consoles
main.py
output.txt
inverted_index.txt
script2skcheck.txt
script2myindex_idf.txt
#Top results for query 'Is CF mucus abnormal' using raw TF method:
1. Document ID: 00498, Score: 0.4848
Top TF-IDF values:
Term: mucu, TF-IDF: 3.1463, IDF: 3.1463
Term: secret, TF-IDF: 1.9659, IDF: 2.2935
Term: attent, TF-IDF: 1.1199, IDF: 3.9195
Term: encompass, TF-IDF: 1.0139, IDF: 7.0975
Term: natur, TF-IDF: 0.9403, IDF: 3.2909
2. Document ID: 00754, Score: 0.3055
Top TF-IDF values:
Term: urti, TF-IDF: 4.2585, IDF: 7.0975
Term: mucu, TF-IDF: 3.1463, IDF: 3.1463
Term: clearanc, TF-IDF: 3.1356, IDF: 3.9195
Term: rt, TF-IDF: 2.8390, IDF:X 7.0975
Term: graviti, TF-IDF: 2.3996, IDF: 5.9989
3. Document ID: 00711, Score: 0.2989
Top TF-IDF values:
Term: lecithin, TF-IDF: 4.2834, IDF: 5.7113
Term: fattyacid, TF-IDF: 3.7636, IDF: 5.0181
Term: mucu, TF-IDF: 2.3597, IDF: 3.1463
Term: composit, TF-IDF: 1.6029, IDF: 3.2057
Term: fraction, TF-IDF: 1.5451, IDF: 3.0902
4. Document ID: 00592, Score: 0.2957
Top TF-IDF values:
Term: hypersecret, TF-IDF: 3.7636, IDF: 5.0181
Term: mucu, TF-IDF: 3.1463, IDF: 3.1463
anakthshplhroforias > output.txt
15:35 LF UTF-8 4 spaces Python 3.9 (anakthshplhroforias)


PyCharm File Edit View Navigate Code Refactor Run Tools Git Window Help
Mon 17 Jun 6:52 PM
anakthshplhroforias main
Project
anakthshplhroforias ~/projects/anakthshplhroforias
ColBERT
Collection
docs
cfquery_detailed
Queries_20
Relevant_20
experiments
venv
inverted_index.txt
main.py
output.txt
requirements.txt
script1.txt
script2.txt
script2myindex.txt
script2myindex_idf.txt
script2skcheck.txt
tfidfscript.txt
External Libraries
Scratches and Consoles
main.py
output.txt
inverted_index.txt
script2skcheck.txt
script2myindex_idf.txt
5. Document ID: 00437, Score: 0.2953
Top TF-IDF values:
Term: mucu, TF-IDF: 1.7979, IDF: 3.1463
Term: rheolog, TF-IDF: 1.6318, IDF: 5.7113
Term: ciliat, TF-IDF: 1.2950, IDF: 4.5326
Term: lung, TF-IDF: 1.2485, IDF: 2.1849
Term: mucociliari, TF-IDF: 1.1580, IDF: 4.0530
Top results for query 'Is CF mucus abnormal' using logarithmic TF method:
1. Document ID: 00498, Score: 0.2823
Top TF-IDF values:
Term: mucu, TF-IDF: 3.1463, IDF: 3.1463
Term: encompass, TF-IDF: 2.4093, IDF: 7.0975
Term: attent, TF-IDF: 2.2527, IDF: 3.9195
Term: secret, TF-IDF: 2.1735, IDF: 2.2935
Term: submucos, TF-IDF: 2.0364, IDF: 5.9989
2. Document ID: 00711, Score: 0.2591
Top TF-IDF values:
Term: lecithin, TF-IDF: 5.0227, IDF: 5.7113
Term: fattyacid, TF-IDF: 4.4131, IDF: 5.0181
Term: mucu, TF-IDF: 2.7670, IDF: 3.1463
Term: redcel, TF-IDF: 2.5139, IDF: 5.9989
Term: oleic, TF-IDF: 2.5139, IDF: 5.9989
3. Document ID: 00754, Score: 0.2151
Top TF-IDF values:
Term: urti, TF-IDF: 5.7081, IDF: 7.0975
anakthshplhroforias > output.txt
15:35 LF UTF-8 4 spaces Python 3.9 (anakthshplhroforias)



Στην συνέχεια έλεγξα τον κώδικα με αποτελέσματα από sklearn. Εδώ ειναι ο κώδικας που χρησιμοποίησα:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import os
import numpy as np

# Function to read documents from a directory
def read_documents_sklearn(directory):
    documents = []
    filenames = []
    for filename in os.listdir(directory):
        filepath = os.path.join(directory, filename)
        with open(filepath, 'r', encoding='utf-8') as file:
            documents.append(file.read())
            filenames.append(filename)
    return documents, filenames

# Directory containing documents
documents_directory = 'Collection/docs'

# Read documents
documents_sklearn, filenames = read_documents_sklearn(documents_directory)

# Example query
query = "Is CF mucus abnormal"

# Initialize TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', use_idf=True)
```

```
# Fit and transform documents
tfidf_matrix = vectorizer.fit_transform(documents_sklearn)

# Transform query
query_vector = vectorizer.transform([query])

# Calculate cosine similarity between query and documents
cosine_similarities = cosine_similarity(query_vector, tfidf_matrix).flatten()

# Get indices of top 10 scores
top_indices = np.argsort(cosine_similarities)[::-1][:10]

# Print top 10 results
print("Top 10 results for query '{}':".format(query))
for rank, idx in enumerate(top_indices, start=1):
   print(f"{rank}. Document ID: {filenames[idx]}, Score:
{cosine_similarities[idx]:.4f}")
```

Άσκηση 3

script3.txt
script32.txt
csvtotsv.txt

για το 3ο ερωτημα χρησιμοποιησα το faiss-cpu αφου ο φορητος μου υπολογιστης δεν διαθετει gpu. Παραθετω τον κωδικα υλοποιησης και την μετατροπη των αρχειων .csv σε ενα αρχειο .tsv
εντολες terminal:

```
!git -C ColBERT/ pull || git clone
https://github.com/stanford-futuredata/ColBERT.git
import sys; sys.path.insert(0, 'ColBERT/')
```

```
!pip install -U pip
!pip install -e ColBERT
```

!pip install faiss-cpu

δημιουργια tsv (ο κώδικας για να μετατρέψω τα αρχεία σε csv υπάρχει στο zip):

```
import os

def convert_csv_to_tsv(csv_directory, tsv_output_file):
```

```
    with open(tsv_output_file, 'w', encoding='utf-8') as tsv_file:
        # Κεφαλίδα στο TSV αρχείο
        tsv_file.write("id\ttext\n")

        # Λήψη όλων των αριθμών από τα ονόματα των αρχείων .csv
        file_numbers = sorted([int(f.split('.')[0]) for f in
os.listdir(csv_directory) if f.endswith('.csv')])

        # Βρίσκουμε το μέγιστο ID που θα πρέπει να χρησιμοποιηθεί
        max_id = max(file_numbers) if file_numbers else 0

        # Βρόχος για όλα τα IDs από 1 έως max_id
        for file_id in range(1, max_id + 1):
            filename = f"{file_id:05d}.csv"  # Ονομασία αρχείου στη μορφή
00001.csv, 00002.csv κλπ.
            csv_path = os.path.join(csv_directory, filename)

            # Αν το αρχείο υπάρχει, διαβάζουμε το περιεχόμενό του
            if os.path.exists(csv_path):
                with open(csv_path, 'r', encoding='utf-8') as csv_file:
                    lines = csv_file.readlines()
                    text = ' '.join(line.strip() for line in lines if
line.strip())  # Συνδυάζουμε τις γραμμές σε μία
            else:
                # Αν το αρχείο δεν υπάρχει, χρησιμοποιούμε το "nothing"
                text = "EMPTY"

            # Γράφουμε το ID και το κείμενο στο αρχείο .tsv
            tsv_file.write(f"{file_id}\t{text}\n")

    print(f"Converted TSV file saved at: {tsv_output_file}")

if __name__ == '__main__':
    # Ο φάκελος που περιέχει τα αρχεία .csv
    csv_directory = 'Collection/docs-02'  # Αντικατάστησε με το σωστό path
    # Το αρχείο εξόδου .tsv
    tsv_output_file = 'collection.tsv'  # Αντικατάστησε με το σωστό path

    convert_csv_to_tsv(csv_directory, tsv_output_file)
```

```
```

Εδω μετέτρεψα τα αρχεία μου απο .csv σε .tsv και στην συνέχεια τα αποθήκευσα στο `'/content/collection.tsv'` για να μπορέσει να τα επεξεργαστεί το ColBERT.
Ταυτόχρονα συνάντησα ένα πρόβλημα καθώς τα αρχεία που μας έχουν δωθεί δεν είναι σε σειρά (δηλαδή κάποια λείπουν) και εγώ θέλω το id να αντικατοπτρίζει το όνομα του αρχικού αρχείου, αλλά ταυτόχρονα τα ids να είναι σε αύξοντα σειρά, αλλιώς παρουσιάζει error το

ColBERT. Για αυτό τον λόγο κατά την δημιουργία των csv αρχείων, οσα αρχεία έλλειπαν τα συμπεριέλαβα στο .tsv αλλά στο περιεχόμενο τους γράφει απλά “EMPTY” και προχωράει στο επόμενο id.
στην συνέχεια κατέβασα το προεκπαιδευμένο μοντέλο του colbert και το έκανα unzip:
colbert --checkpoint /path/to/colbertv2.dnn

Παρακάτω παραθέτω την λειτουργία indexing:

```
from colbert.infra import Run, RunConfig, ColBERTConfig
from colbert import Indexer

if __name__ == '__main__':
   with Run().context(RunConfig(nranks=1, experiment="colbert_experiment")):
       config = ColBERTConfig(
           nbits=2,
           root="experiments",  # Ο φάκελος όπου θα αποθηκευτούν τα
αποτελέσματα του indexing
       )

       # Διαδρομή προς το φάκελο με το checkpoint
       checkpoint_path = "colbertv2.0"

       indexer = Indexer(checkpoint=checkpoint_path, config=config)
       indexer.index(name="colbert_index", collection="collection.tsv")
```

παραθετω και την λειτουργια αναζητησης:

```
from colbert.infra import Run, RunConfig, ColBERTConfig
from colbert import Searcher
from colbert.data import Queries

if __name__ == '__main__':
   with Run().context(RunConfig(nranks=1, experiment="colbert_experiment")):
       config = ColBERTConfig(
           root="experiments"  # Διαδρομή για τα αποτελέσματα
       )

       searcher = Searcher(index="colbert_index", config=config)

       queries = Queries("queries.tsv")

       ranking = searcher.search_all(queries, k=100)
       ranking.save("colbert_index.ranking.tsv")
```

Οι παραπάνω γραμμές κώδικα δημιουργούν πολλά διαφορετικά αρχεία, ο πρώτος κώδικας φτιάχνει το index και ο 2ος το search. Για αριθμό clusters και επαναλήψεις στο indexing δεν άλλαξα κάτι, αλλά χρησιμοποίησα αυτά που δίνει το ColBERT by default.

Ασκηση 4

script4.txt

rawscript2.txt

Ξεκινάμε αλλάζοντας λίγο την άσκηση 2 έτσι ώστε τα αποτελέσματα tf-idf να είναι πιο καθαρά για ανάγνωση και να αποθηκεύονται σε ένα αρχείο “query_results.txt”

```
import os
import re
import math
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from collections import defaultdict

# Ensure you have the necessary NLTK data files
nltk.download('stopwords')
nltk.download('punkt')


# Function to preprocess text
def preprocess_text(text):
    text = re.sub(r'\W', ' ', text)
    text = text.lower()
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in
stopwords.words('english')]
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]
    return tokens


# Function to read documents from a directory
def read_documents(directory):
    documents = {}
    for filename in os.listdir(directory):
        filepath = os.path.join(directory, filename)
        with open(filepath, 'r', encoding='utf-8') as file:
            documents[filename] = file.read()
    return documents
```

```
# Function to read inverted index from a file
def read_inverted_index(file_path):
    inverted_index = defaultdict(list)
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip()
            if line.startswith('Term: '):
                parts = line.split(', Document IDs: ')
                term = parts[0].replace('Term: ', '').strip()
                doc_ids_str = parts[1].replace('[', '').replace(']',
'').strip()
                doc_ids = [doc_id.strip() for doc_id in doc_ids_str.split(',
')]
                inverted_index[term] = doc_ids
    return inverted_index


# Function to calculate TF-IDF for documents
def calculate_tf_idf(documents, inverted_index):
    N = len(documents)
    tfidf = defaultdict(lambda: defaultdict(float))
    idf = {}

    # Calculate IDF for each term
    for term, postings in inverted_index.items():
        ni = len(set(postings))
        idf[term] = math.log(N / ni) if ni > 0 else 0.0

    # Calculate TF-IDF for each term in each document
    for doc_id, content in documents.items():
        tokens = preprocess_text(content)
        tf = defaultdict(float)
        for term in tokens:
            tf[term] += 1.0  # Simple term frequency

        max_tf = max(tf.values(), default=1.0)
        for term in tokens:
            tfidf[doc_id][term] = (tf[term] / max_tf) * idf.get(term, 0.0)

    return tfidf, idf


# Function to execute a query and retrieve top results
def execute_query(query, documents, inverted_index, tfidf, idf, output_file):
    query_tokens = preprocess_text(query)
    scores = defaultdict(float)
    query_vector = defaultdict(float)

    # Calculate query vector (simple term frequency)
    for term in query_tokens:
        query_vector[term] += 1.0

    # Calculate TF-IDF for query vector
```

```
    max_tf_query = max(query_vector.values(), default=1.0)
    for term in query_tokens:
        query_vector[term] = (query_vector[term] / max_tf_query) *
idf.get(term, 0.0)

    # Calculate cosine similarity between query vector and document vectors
    query_norm = sum(query_vector[term] ** 2 for term in query_vector)
    query_norm = math.sqrt(query_norm) if query_norm != 0 else 1  # Normalize
query vector

    for doc_id in documents:
        doc_vector = defaultdict(float)
        for term, weight in tfidf[doc_id].items():
            doc_vector[term] = weight

        # Calculate document vector norm
        doc_norm = sum(doc_vector[term] ** 2 for term in doc_vector)
        doc_norm = math.sqrt(doc_norm) if doc_norm != 0 else 1  # Normalize
document vector

        # Calculate dot product
        dot_product = sum(query_vector[term] * doc_vector[term] for term in
query_tokens if term in doc_vector)

        # Calculate cosine similarity
        cosine_similarity = dot_product / (query_norm * doc_norm) if
query_norm * doc_norm != 0 else 0
        scores[doc_id] = cosine_similarity

    # Sort scores in descending order
    sorted_scores = sorted(scores.items(), key=lambda x: x[1], reverse=True)

    # Write results to file
    with open(output_file, 'w', encoding='utf-8') as file:
        file.write(f"Top results for query '{query}':\n")
        for rank, (doc_id, score) in enumerate(sorted_scores[:5], start=1):
            file.write(f"{rank}. Document ID: {doc_id}, Score: {score:.4f}\n")

    print(f"Results written to {output_file}")

if __name__ == "__main__":
    # Directory containing documents
    documents_directory = 'collection/docs'

    # Read documents
    documents = read_documents(documents_directory)

    # File path for inverted index
    inverted_index_file = 'inverted_index.txt'

    # Read inverted index from file
    inverted_index = read_inverted_index(inverted_index_file)
```

```
    # Calculate TF-IDF for documents using the provided inverted index
    tfidf, idf = calculate_tf_idf(documents, inverted_index)

    # Example query
    query = "Is CF mucus abnormal"

    # Output file for results
    output_file = 'query_results.txt'

    # Execute the query and write top results to the output file
    execute_query(query, documents, inverted_index, tfidf, idf, output_file)
```

Οι 2 τεχνικές που χρησιμοποιώ για να συγκρίνω τα αποτελέσματα είναι η precision και η recall. Η precision συγκρίνει πόσες από τις τοπ 5 τιμές που θα επιστραφούν ανήκουν όντως στα σχετικά έγγραφα (αριθμός σχετικών εγγράφων/5) και η recall θα συγκρίνει το πλήθος αυτών των εγγράφων σε σχέση με το συνολικό πλήθος των εγγράφων.

```
def read_query_relevant_docs(file_path, query_index):
    """Read relevant document IDs for a specific query."""
    with open(file_path, 'r') as file:
        lines = file.readlines()
        if 0 <= query_index < len(lines):
            relevant_docs = lines[query_index].strip().split()
            # Remove leading zeros from relevant document IDs
            relevant_docs = {doc_id.lstrip('0') for doc_id in relevant_docs}
            return relevant_docs
    return set()


def read_query_results(file_path):
    """Read query results from the file."""
    query_results = []
    with open(file_path, 'r') as file:
        for line in file:
            if line.startswith("Top results for query") or
line.startswith("Document ID"):
                continue
            try:
                parts = line.strip().split(', ')
                if len(parts) == 2:
                    doc_id = parts[0].split(': ')[1].strip().lstrip('0')  #
Remove leading zeros
                    score = float(parts[1].split(': ')[1].strip())
                    query_results.append((doc_id, score))
            except ValueError:
                print(f"Skipping line due to ValueError: {line.strip()}")
    print(f"Query Results Read: {query_results}")  # Display results for
debugging
    return query_results


def read_colbert_results(file_path):
    """Read ColBERT results from the file."""
    colbert_results = []
```

```
    with open(file_path, 'r') as file:
        for line in file:
            parts = line.strip().split('\t')
            if len(parts) >= 4:
                doc_id = parts[1].strip().lstrip('0')  # Remove leading zeros
                score = float(parts[3].strip())
                colbert_results.append((doc_id, score))
    return colbert_results


def get_top_n(results, n):
    """Get top N results sorted by score."""
    return sorted(results, key=lambda x: x[1], reverse=True)[:n]


def compute_precision_recall(query_relevant_docs, query_results, top_n=5):
    """Compute Precision@5 and Recall@5 for query results."""
    top_docs = {doc_id for doc_id, _ in get_top_n(query_results, top_n)}
    intersection = query_relevant_docs.intersection(top_docs)

    # Debug prints
    print(f"Top docs: {top_docs}")
    print(f"Intersection: {intersection}")

    # Precision calculation
    precision_at_n = len(intersection) / top_n if top_n > 0 else 0.0

    # Recall calculation
    recall_at_n = len(intersection) / len(query_relevant_docs) if
query_relevant_docs else 0.0

    return precision_at_n, recall_at_n


def main():
    query_results_path = 'query_results.txt'
    colbert_results_path =
'experiments/colbert_experiment/main/2024-09/01/03.36.17/colbert_index.rankin
g.tsv'
    relevant_docs_path = 'collection/Relevant_20'
    query_index = 4  # 5th query (0-based index)

    # Read relevant document IDs for the query
    query_relevant_docs = read_query_relevant_docs(relevant_docs_path,
query_index)

    # Debug print
    print(f"Relevant docs: {query_relevant_docs}")

    # Read query results and ColBERT results
    query_results = read_query_results(query_results_path)
    colbert_results = read_colbert_results(colbert_results_path)
```

```
    # Compute top N results
    top_query_results = get_top_n(query_results, 5)
    top_colbert_results = get_top_n(colbert_results, 5)

    # Compute Precision@5 and Recall@5 for both results
    precision_query, recall_query =
compute_precision_recall(query_relevant_docs, top_query_results)
    precision_colbert, recall_colbert =
compute_precision_recall(query_relevant_docs, top_colbert_results)

    # Print results
    print(f"Top 5 query results: {top_query_results}")
    print(f"Top 5 ColBERT results: {[doc_id for doc_id, _ in
top_colbert_results]}")
    print(f"Precision@5 for query results: {precision_query:.4f}")
    print(f"Recall@5 for query results: {recall_query:.4f}")
    print(f"Precision@5 for ColBERT results: {precision_colbert:.4f}")
    print(f"Recall@5 for ColBERT results: {recall_colbert:.4f}")


if __name__ == "__main__":
    main()
```

χρησιμοποίησα πάλι το query με αριθμό 5 “Is CF Mucus Abnormal”
παραθετω και τα αποτελέσματα:
Relevant docs: {'1156', '876', '593', '437', '943', '298', '114', '1188', '478', '496', '864', '151', '856', '511', '200', '312', '861', '895', '499', '1226', '520', '733', '561', '297', '420', '702', '925', '845', '441', '531', '982', '1098', '761', '1091', '549', '980', '729', '434', '503', '1076', '779', '944', '592', '513', '430', '370', '568', '788', '1040', '299', '427', '935', '731', '533', '386', '669', '711', '867', '1092', '633', '132', '888', '975', '189', '47', '450', '502', '1000', '1080', '139', '392', '772', '750', '805', '333', '1038', '857', '265', '1196', '349', '722', '197', '1093', '553', '371', '497', '1019', '516', '1144', '604', '843', '605', '465', '524', '343', '559', '461', '724', '256', '60', '169', '875', '23', '311', '190', '701', '226', '347', '500', '498', '439', '889', '1064', '410', '369', '1185', '590', '505', '1088', '763', '50', '710', '428', '325', '374', '1223', '990', '1175', '440', '135', '501'}
Query Results Read: [('498', 0.4848), ('754', 0.3055), ('711', 0.2989), ('592', 0.2957), ('437', 0.2953)]
Top docs: {'592', '754', '498', '437', '711'}
Intersection: {'437', '711', '498', '592'}
Top docs: {'711', '499', '151', '568', '501'}
Intersection: {'711', '151', '499', '568', '501'}
Top 5 query results: [('498', 0.4848), ('754', 0.3055), ('711', 0.2989), ('592', 0.2957), ('437', 0.2953)]
Top 5 ColBERT results: ['499', '711', '501', '568', '151']
Precision@5 for query results: 0.8000
Recall@5 for query results: 0.0305
Precision@5 for ColBERT results: 1.0000
Recall@5 for ColBERT results: 0.0382

παρατηρούμε οτι τα αποτελέσματα του ColBERT είναι πιο εύστοχα και στις 2 τεχνικές. Ενώ το δικό μου search έπιασε 4 στα 5 του ColBERT όσα αρχεία έβγαλε είναι τα ίδια με τα αυτά

το αρχείο relevant. Βέβαια εγώ περιόρισα την αναζήτηση μου στο τοπ 5 αλλά έκανα την ίδια λειτουργία για τα τοπ 10 σε αντιστοίχιση αρχεία και αυτά είναι τα αποτελέσματα που πήρα
Precision@10 for query results: 0.4000
Recall@10 for query results: 0.0305
Precision@10 for ColBERT results: 0.8000
Recall@10 for ColBERT results: 0.0611

παρατηρούμε ότι το ColBERT είναι πολύ πιο αποτελεσματικό και σε αρχεία τα οποία δεν έχουν τόσο μεγάλη αλλα έχουν αντιστοιχία ενώ ο δικός μου κώδικας αναζήτησης όσο απομακρύνονται τα score tf-idf τόσο πιο αναξιόπιστος γίνεται.

ΣΗΜΑΝΤΙΚΟ: Τα αρχεία που εμφανίζονται από το search στην άσκηση 3 με path:
“`'experiments/colbert_experiment/main/2024-09/01/03.36.17/colbert_index.ranking.tsv'`”
αλλάζουν path ανάλογα με το πότε έγινε η αναζήτηση οπότε σε μελλοντικές κλήσεις του κώδικα search θα πρεπει να αλλάξουμε το path και στην άσκηση 4.