Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών

Ανάκτηση Πληροφορίας

Εργαστηριακή Άσκηση
Χειμερινό Εξάμηνο 2023

Κουρή Μαρία 1084526 up1084526@ac.webmail.upatras.gr
Μακρυγιάννης Παντελεήμων 1067526 up1067526@ac.webmail.upatras.gr

Εισαγωγή

Η εργασία πραγματοποιήθηκε στο πλαίσιο του μαθήματος «Ανάκτηση Πληροφορίας», κατά την διάρκεια του χειμερινού εξαμήνου, το ακαδημαϊκό έτος 2023-2024. Στόχος είναι η ανάπτυξη δύο μοντέλων ανάκτησης πληροφορίας και η σύγκρισή τους. Το πρώτο μοντέλο που αναπτύχθηκε είναι το Μοντέλου Διανυσματικού Χώρου [1] με δημιουργία ανεστραμμένου ευρετηρίου (και το δεύτερο το του μοντέλου colBERT, στηριζόμενο στην παραλλαγή που πρότεινε το Stanford University [2]. Τέλος, για την σύγκριση και την αξιολόγηση των δύο μοντέλων χρησιμοποιήθηκαν δύο μετρικές εκτίμησης, η Precision-Recall και η MAP (Mean Average Precision).

Και για τα δύο μοντέλα χρησιμοποιήθηκε η συλλογή Cystic Fibrosis (C.F.), η οποία περιλαμβάνει 1209 κείμενα, 20 ερωτήματα και τα σχετικά κείμενα για κάθε ερώτημα.

Υλοποίηση

Βιβλιοθήκες

Για την υλοποίηση των μοντέλων επιλέγεται γλώσσα προγραμματισμού python, καθώς έχει βιβλιοθήκες που χρησιμοποιούνται ευρέως στην Ανάκτηση Πληροφορίας, ενώ ταυτόχρονα ο κώδικας σε python είναι ευανάγνωστος και εύκολα κατανοητός.

Αρχικά, για την δημιουργία και την εκτέλεση των μοντέλων απαιτήθηκαν οι κατάλληλες βιβλιοθήκες και συγκεκριμένες παραμετροποιήσεις στα περιβάλλοντα, όπου εκτελέστηκε το κάθε μοντέλο ανάκτησης πληροφορίας. Το Μοντέλο Διανυσματικού Χώρου που αναπτύχθηκε εκτελέστηκε στο περιβάλλον Visual Code, με εγκατεστημένη την έκδοση Python 3.12.0. Στο μοντέλο αυτό χρησιμοποιήθηκαν οι βιβλιοθήκες:

i. **os**: δίνει πρόσβαση στα αρχεία του υπολογιστικού συστήματος
ii. **collections**: επιτρέπει τη δημιουργία ενός defaultdict, με αποτέλεσμα κάθε φορά που ανακαλύπτεται μια νέα λέξη μέσα στα κείμενα να επιστρέφει μια κενή λίστα, εάν η λέξη δεν υπάρχει ήδη στο ευρετήριο.
iii. **math**: χρησιμοποιείται για μαθηματικούς υπολογισμούς που απαιτούνται, ώστε να υπολογιστούν τα βάρη.
iv. **pandas**: χρησιμοποιείται για τη δημιουργία και επεξεργασία των DataFrame - δομές δεδομένων - όπου κάθε γραμμή αντιστοιχεί σε ένα στοιχείο του λεξικού
v. **numpy**: γίνεται εισαγωγή της συγκεκριμένης βιβλιοθήκης, ώστε να υπολογιστεί η ομοιότητα μεταξύ των Data Frames (cosine similarity).
vi. **matplotlib**.**pyplot**: απαιτείται για την δημιουργία γραφικών παραστάσεων στις μετρικές αξιολόγησης των μοντέλων.
vii. **sklearn**.**metrics**: η βιβλιοθήκη εισάγεται για τον υπολογισμό του auc (υπολογισμός περιοχής κάτω από την καμπύλη precision – recall και όχι για τον υπολογισμό των μετρικών)

Το μοντέλο ColBERT εκτελέστηκε σε Google Colab με ενεργοποιημένη την GPU (T4 GPU). Για την ανάπτυξη του μοντέλου εγκαταστάθηκαν βιβλιοθήκες και έγινε η κατάλληλη παραμετροποίηση του περιβάλλοντος. Ακολουθούν οι βιβλιοθήκες που χρησιμοποιήθηκαν:

i. Εγκαταστάθηκε το περιβάλλον virtualenv και δημιουργήθηκε νέο περιβάλλον myenv (εάν δεν ενεργοποιηθεί εμφανίζεται ειδοποίηση για πιθανή καταστροφή του config του NotePad)
    a. **python3.10-venv**
    b. **virtualenv**
ii. **torch:** είναι βιβλιοθήκη ανοικτού κώδικα, η οποία χρησιμοποιείται κυρίως σε μηχανική μάθηση. Για την εκτέλεση του ColBERT είναι απαραίτητη η εισαγωγή του, καθώς το συγκεκριμένο μοντέλο είναι deep learning
iii. **torchvision και torchaudio**: υποβιβλιοθήκες του torch που γίνονται εισαγωγή, ώστε να υπάρχει δυνατότητα επεξεργασίας εικόνας και ήχου από το μοντέλο που δημιουργείται
iv. **CUDA**: ενεργοποιείται το CUDA - πρότυπο υπολογισμού που δημιουργήθηκε από την NVIDIA για την εκτέλεση παράλληλων υπολογισμών σε γραφικές κάρτες της – καθώς χωρίς αυτό δεν μπορεί να γίνει εισαγωγή της βιβλιοθήκης ColBERT (CUDA not found)
v. **ColBERT**: είναι απαραίτητη για την εκτέλεση του μοντέλου ColBERT, για την αναζήτηση περιεχομένου. Η βιβλιοθήκη αυτή δημιουργήθηκε από το Stanford Future Data Systems και περιέχει υλοποίηση του μοντέλου για αποτελεσματική ανάκτηση πληροφοριών. Από την βιβλιοθήκη ColBERT έγινε χρήση των ακολούθων στοιχείων, ώστε να ρυθμιστεί σωστά το περιβάλλον:
    a. **Indexer, Searcher**
    b. **Run, RunConfig, ColBERTConfig**
    c. **Queries, Collection**
vi. **from google.colab import drive**: απαραίτητη βιβλιοθήκη για την εισαγωγή των δοθέντων αρχείων (docs, queries, etc.). Τα αρχεία που συνόδευαν την εκφώνηση ανέβηκαν σε google drive και από εκεί έγινε κοινοποίηση τους και εισαγωγή τους μέσα στο περιβάλλον Google Colab
vii. **csv**: η βιβλιοθήκη χρησιμοποιείται για την εξαγωγή των αποτελεσμάτων, έπειτα από την εκτέλεση του colBERT, ώστε να χρησιμοποιηθούν στις μετρικές που θα αναπτυχθούν.

Μοντέλο Διανυσματικού Χώρου [3]

Για την υλοποίηση του Μοντέλου Διανυσματικού Χώρου (Vector Space Model – VSM) δημιουργήθηκε αντεστραμμένο ευρετήριο. Για την δημιουργία του ευρετηρίου χρησιμοποιήθηκε η defaultdict(list). Σε πρώτο στάδιο δημιουργείται κενό ευρετήριο, όπου τα κλειδιά θα είναι λέξεις και οι τιμές θα είναι λίστες με τα ονόματα των αρχείων και τις συχνότητες εμφάνισης της λέξης σε κάθε αρχείο. Το αρχείο αυτό διατηρεί πληροφορίες για τους όρους και την συχνότητα εμφάνισής τους, ώστε να βασιστεί σε αυτό το μοντέλο της ΑΠ [4]. Γενικά, το ανεστραμμένο ευρετήριο εξυπηρετεί στα μοντέλα ανάκτησης, καθώς μειώνει σημαντικά τον χρόνο αναζήτησης. [3] Αφού οριστεί ο φάκελος που περιέχει τα αρχεία και γίνει διαχωρισμός του κειμένου σε λέξεις, γίνεται προσπέλαση των αρχείων και ανάγνωσή τους. Για κάθε λέξη όταν εντοπίζεται αυξάνεται ο αντίστοιχος αριθμός εμφάνισης της λέξης στο εκάστοτε έγγραφο και καταγράφεται η πληροφορία ως διατεταγμένο ζεύγος στο αντεστραμμένο ευρετήριο. Έτσι υπολογίζεται η

συχνότητα εμφάνισης κάθε λέξης, καταμετρώντας τον αριθμό των εγγράφων που περιέχουν τη συγκεκριμένη λέξη.

```
inverted_index = defaultdict(list)

folder_path = '/Users/mariakouri/Desktop/Ανάκτηση Πληροφορίας/docs'

for filename in os.listdir(folder_path):
    file_path = os.path.join(folder_path, filename)

    if os.path.isfile(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            text = file.read()

        words = text.split()

        for word in set(words):
            inverted_index[word].append((filename, words.count(word)))

# Υπολογισμός της συχνότητας εμφάνισης κάθε λέξης
word_counts = defaultdict(int)
for word, documents in inverted_index.items():
    word_counts[word] = len(documents)
```

Το ανεστραμμένο ευρετήριο που δημιουργήθηκε χρησιμοποιείται για τον υπολογισμό της Συχνότητας Εμφάνισης (Term Frequency - TF) της κάθε λέξης. Η Συχνότητα Εμφάνισης του κάθε όρου αποθηκεύεται σε ένα DataFrame. Γίνεται προσπέλαση με κατάλληλο βρόχο του αντεστραμμένου ευρετηρίου και για κάθε όρο γίνεται έλεγχος εάν υπάρχει ήδη ως εγγραφή στο DataFrame και εάν δεν υπάρχει δημιουργείται μία νέα εγγραφή. Εάν η εγγραφή υπάρχει, συμπεριλαμβάνεται στην ήδη υπάρχουσα και προστίθεται το TF του. Για τον υπολογισμό της συχνότητας εμφάνισης του κάθε όρου χρησιμοποιείται ο παρακάτω τύπος. Ο τύπος επιλέγεται, διότι διευκολύνει την σύγκριση με της αντίστροφη συχνότητα εγγράφου.

$$TF_{ij} = 1 + \log_{10} F_{ij}$$

```
# Υπολογισμός του tf
tf_values = {}
for word, occurrences in inverted_index.items():
    for document, count in occurrences:
        tf = 1 + math.log10(count)
        if document in tf_values:
            tf_values[document][word] = tf
        else:
            tf_values[document] = {word: tf}
```

Στην συνέχεια, γίνεται υπολογισμός της αντιστροφής συχνότητας εγγράφου (Inverse Document Frequency – IDF) και δημιουργείται αντίστοιχο DataFrame για την αποθήκευση των αποτελεσμάτων. Ο όρος IDF δηλώνει σε πόσα κείμενα υπάρχει ο κάθε όρος και υπολογίζεται με τον ακόλουθο τύπο:

$$IDF_{ij} = \log_{10} \frac{N}{n_i}$$

```
# Υπολογισμός του idf και αποθήκευση σε ένα dictionary
idf_values = {}
total_documents = len(tf_df)
for word in inverted_index.keys():
    df = word_counts[word]
    idf = math.log10(total_documents / df) if df > 0 else 0
    idf_values[word] = idf
```

Στο επόμενο στάδιο, δημιουργείται ένα DataFrame (weight_df) στο οποίο αποθηκεύονται τα βάρη των όρων σε κάθε έγγραφο. Ο υπολογισμός των βαρών (weight) δίνεται από τον ακόλουθο τύπο:

$$weight_{ij} = TF_{ij} * IDF_{ij}$$

```
#βάρη (weights)
weight_values = {}

for document, word_tf_values in tf_values.items():
    weight_values[document] = {}
    for word, tf in word_tf_values.items():
        weight = tf * idf_values[word]
        weight_values[document][word] = weight

# Δημιουργία του DataFrame για τα βάρη (weights)
weight_df = pd.DataFrame.from_dict(weight_values, orient='index')
```

Αφού γίνουν οι παραπάνω υπολογισμοί, γίνεται ανάγνωση των ερωτημάτων του αρχείου Queries_20 και δημιουργείται ένα DataFrame, με γραμμές τα ερωτήματα και στήλες τους όρους, ενώ στα κελιά είναι αρχικοποιημένη η τιμή μηδέν. Έπειτα από έλεγχο με κατάλληλο βρόχο, εάν ο όρος βρεθεί στο αντίστοιχο ερώτημα το κελί τρέπεται σε 1 (TF για τα query).

```
# Επεξεργασία των queries
for i, query in enumerate(queries_list, start=1):
    query_words = query.strip().split()  # Διαχωρισμός του query σε λέξεις
    for word in query_words:
        if word.upper() in query_df.columns:
            query_df.at[i, word.upper()] = 1  # Αν η λέξη υπάρχει στο query, θέτουμε
το κελί σε 1
```

Με αντίστοιχο τρόπο υπολογίζεται και το IDF – TF (query_idf_df) των όρων στα ερωτήματα, πολλαπλασιάζοντας το DataFrame που περιέχει τα TF για τα query, με το IDF των όρων και τα αποτελέσματα αποθηκεύονται σε ένα νέο DataFrame.

```
# Δημιουργία του DataFrame για το query_idf
query_idf_values = {}

for document, word_tf_values in query_df.iterrows():
    query_idf_values[document] = {}
    for word, tf in word_tf_values.items():
        query_idf_values[document][word] = tf * idf_values[word]  #όπου tf η συχνότητα
του όρου στα query με βάση τον πίνακα
```

Τέλος, υπολογίζεται το cosine similarity μεταξύ των ερωτημάτων και των εγγράφων. Για τον υπολογισμό του cosine similarity ακολουθείται ο παρακάτω τύπος:

$$cosine\_similarity(\text{query}_{\text{idf}_{\text{df}}}, \text{weight}_{\text{df}}) = \frac{\text{query}_{\text{idf}_{\text{df}}} * \text{weight}_{\text{df}}}{\left\|\text{query}_{\text{idf}_{\text{df}}}\right\| * \left\|\text{weight}_{\text{df}}\right\|}$$

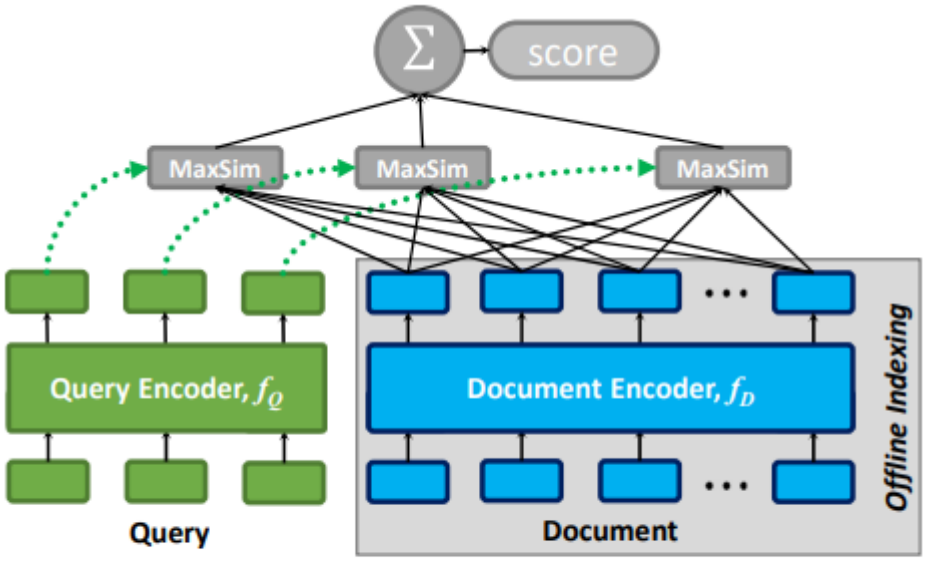
```
# Πολλαπλασιασμός των DataFrames
cosine_sim = query_idf_df.dot(weight_df.T)

# Ευκλείδειες νόρμες των γραμμών των DataFrames
query_idf_norm = np.linalg.norm(query_idf_df.values, axis=1)
weight_norm = np.linalg.norm(weight_df.values, axis=1)

# Διαίρεση με το γινόμενο των ευκλείδειων νορμών
cosine_sim = cosine_sim.divide(np.outer(query_idf_norm, weight_norm))
```

- Παρουσίαση των σημαντικότερων σημείων της υλοποίησης

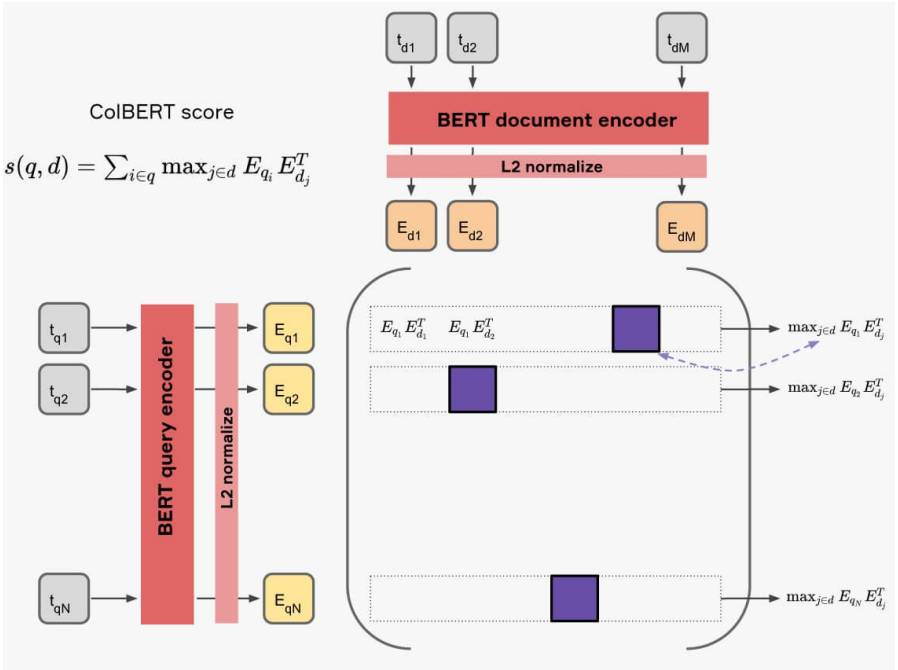
colBERT [Stanford Edition] [5] [6]

Το ColBERT είναι ένα μοντέλο ανάκτησης πληροφορίας, το οποίο χρησιμοποιεί διανυσματικές αναπαραστάσεις και συνδυάζει σε μεγάλο βαθμό την αποτελεσματικότητα του με την δημιουργία συμφραζομένων. Το μοντέλο αυτό εκτελεί μια vector similarity αναζήτηση με αποτέλεσμα να είναι δυνατή η end-to-end ανάκτηση. Η δομή του ColBERT αποτελείται από τον

Query Encoder και τον Document Encoder. Στο ColBERT χρησιμοποιείται το τελεστής μέγιστης ομοιότητας (MaxSim) για τον υπολογισμό των relevance scores. Για κάθε query υπολογίζεται το MaxSim για κάθε όρο ο οποίος περιέχεται μέσα στο έγγραφο και αθροίζεται έτσι ώστε να προκύψει το τελικό score. Τα πλεονεκτήματα του υπολογισμού MaxSim είναι η οικονομική υλοποίηση και η αποτελεσματικότητα στην ανάκτηση των top-k κειμένων.



Για ένα Query και ένα Document το Relevance Score υπολογίζεται με το άθροισμα των τιμών που λαμβάνονται μέσω του cosine similarity.



Κατά τη διαδικασία ευρετηρίασης(indexing), υπολογίζεται η αναπαράσταση (embedding) των εγγράφων σε batches με τη χρήση encoder και αποθηκεύεται. Αφού δημιουργηθεί η αναπαράσταση κάθε εγγράφου, χρησιμοποιείται σε περιπτώσεις ranking και indexing

Για το μοντέλο ColBERT χρησιμοποιήθηκε η υλοποίηση του Standford, όπως προτείνεται. Αρχικά στο Google Colab δημιουργήθηκε ένα νέο Notebook, όπου εγκαταστάθηκε και ενεργοποιήθηκε ένα Virual Environment (system warning for using virtual environment). Μετά τον έλεγχο που γίνεται για τη χρήση GPU εγκαθίσταται η βιβλιοθήκη του ColBERT και γίνονται οι απαραίτητες παραμετροποιήσεις και έλεγχοι. Στην συνέχεια υλοποιείται μια συνάρτηση, η οποία έχει ως στόχο να επιστρέψει τα tuples που θα δημιουργηθούν από την προσπέλαση του εγγράφου όπου περιέχονται τα κείμενα. Τα tuples αυτά, περιέχουν ένα μοναδικό ID για κάθε κείμενο και το περιεχόμενο κάθε κειμένου και με το πέρας της συνάρτησης προστίθενται όλα στην κενή λίστα documents.

```
def read_documents_from_directory(directory_path):
    documents = []
    for filename in os.listdir(directory_path):
        with open(os.path.join(directory_path, filename), 'r', encoding="utf8") as
file:
            text = file.read()
            document_id = len(documents) + 1  # Εκχώρηση ενός μοναδικού ID με βάση τη
θέση
            document_tuple = (document_id, text)
            documents.append(document_tuple)
    return documents
```

Η ίδια διαδικασία ακολουθείται για το αρχείο με τα queries.

```
# Διάβασμα των queries από τον φάκελο
def read_queries_from_file(file_path):
    queries = []
    with open(file_path, 'r', encoding="utf8") as file:
        text = file.read()
        queries_list = text.split('\n')
        queries_list = [query.strip() for query in queries_list if query.strip()]
        queries.extend(queries_list)
    return queries
# Read queries from file
```

Στην συνέχεια πραγματοποιείται το indexing (ευρετηρίαση). Δημιουργείται ευρετήριο όρων, το οποίο διευκολύνει την διαδικασία της αναζήτησης, και αφού τελειώσει η διαδικασία της ευρετηρίασης τα αποτελέσματα θα αποθηκευτούν με το όνομα index_name.

```
# Indexing
with Run().context(RunConfig(nranks=1, experiment='indexing')):
    config = ColBERTConfig(doc_maxlen=doc_maxlen, nbits=nbits, kmeans_niters=4)
    indexer = Indexer(checkpoint=checkpoint, config=config)
    indexer.index(name=index_name, collection=[doc[1] for doc in documents[:max_id]],
overwrite=True)
```

Επίσης με τον ίδιο τρόπο κατασκευάζεται και ο Searcher βασισμένος πάνω στο πλαίσιο του Indexing.

```
# Searching
with Run().context(RunConfig(experiment='indexing')):
    searcher = Searcher(index=index_name, collection=[doc[1] for doc in documents])
```

Με τη χρήση του Searcher επαναλαμβάνεται η διαδικασία της αναζήτησης για κάθε ερώτημα που βρίσκεται στον φάκελο queries. Ο Searcher χρησιμοποιείται για να αναζητήσει τα passages που σχετίζονται με το τρέχον ερώτημα κάθε στιγμή. Το αποτέλεσμα είναι η λίστα results όπου περιέχει τα passage_id, passage_rank και passage_score. Η τελική εκτύπωση αφορά το document_id, το passage_rank, όπου αφορά τον βαθμό σχετικότητας του κειμένου και το passage_id. Σε αυτό το σημείο τονίζεται πως το passage_id αντιστοιχίζεται με το document_id λόγω του κατακερματισμού που μπορεί να συμβαίνει κατά την διάρκεια του indexing. Το passage_id αποτελεί μοναδικό αναγνωριστικό για κάθε passage που δημιουργείται, αλλά δεν αντιπροσωπεύει την τιμή του document_id. Για αυτό απαιτείται αντιστοίχιση του passage_id με το document_id, με την εντολή `document_id = documents[passage_id][0]`.

```
# Επανάληψη για όλα τα queries
for query in queries:
    print(f"#> {query}")

    # Εύρεση των passages για κάθε query
    results = searcher.search(query, k=1209)  # Retrieve all passages

    # Εκτύπωση των passages που ανακτήθηκαν με το σχετικό score
    for i, (passage_id, passage_rank, passage_score) in enumerate(zip(*results)):
        document_id = documents[passage_id][0]
        print(f"\t [{document_id}] \t\t {passage_score:.1f} \t\t {passage_id}")
```

Τέλος δημιουργείται ένα αρχείο csv με τα αποτελέσματα του μοντέλου colBERT , ώστε να μπορέσει να γίνει εξαγωγή και να αναπτυχθεί κώδικας για τις Μετρικές Αξιολόγησης, σε περιβάλλον VS Studio.

```
# Δημιουργία dictionary για αποθήκευση λιστών των document IDs για κάθε query
query_document_lists = {}


# Επανάληψη για όλα τα queries
for query in queries:
    print(f"#> {query}")

    # Εύρεση των passages για κάθε query
```

```
    results = searcher.search(query, k=1209)  # Ανάκτση όλων των passages

    # Προσθήκη των αποτελεσμάτων στην λίστα
    result_list = []
    for i, (passage_id, passage_rank, passage_score) in    enumerate(zip(*results)):
        result_list.append(passage_id)

    # Καταχώρηση της λίστας για το τέχον querie στο dictionary
    query_document_lists[query] = result_list

    # Αποθήκευση του dictionary σε ένα CSV έγγραφο
    csv_file_path = '/content/colbert_result.csv'
    with open(csv_file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)

    # Εγγραφή του header στο αρχείο
    writer.writerow(['Query', 'DocumentIDs'])

    # Εγγραφή των data στο αρχείο
    for query, document_list in query_document_lists.items():
        writer.writerow([query, document_list])
```

Μετρικές σύγκρισης

Για να συγκριθεί η απόδοση των δύο μοντέλων και να αξιολογηθεί η ανάκτηση τους, σε σχέση με τα σχετικά κείμενα για κάθε ερώτημα, χρησιμοποιήθηκαν δύο μετρικές. Η πρώτη μετρική είναι αυτή της Ανάκλησης και της Ακρίβειας (Precision – Recall) και η δεύτερη αυτή της Mean Average Precision (MAP).

Στην μετρική Ανάκλησης – Ακρίβειας, η ανάκληση υπολογίζει το ποσοστό των σχετικών κειμένων που έχει ανακτηθεί και η ακρίβεια υπολογίζει το ποσοστό των ανακτηθέντων κειμένων που είναι σχετικό. Έστω R το σύνολο των σχετικών κειμένων και |R| ο πληθάριθμος του R. Έστω ότι το μοντέλο ΑΠ ανακτά ένα σύνολο κειμένων A και |A| ο πληθάριθμός του. Αν υποθέσουμε ότι |Ra| είναι τα κοινά κείμενα των συνόλων R και A, τότε η ακρίβεια και η ανάκληση ορίζονται ως εξής:

$$Ανάκληση = \frac{|Ra|}{|R|} \quad , \quad Ακρίβεια = \frac{|Ra|}{|A|}$$

Για τον υπολογισμό της μετρικής Mean Average Precision (MAP), αρχικά υπολογίζεται το Average Precision (AP), το οποίο ορίζεται ως η μέση τιμή των τιμών της ακρίβειας για όλα τα σχετικά κείμενα της συλλογής για κάθε ερώτημα. Στην συνέχεια, υπολογίζεται η μέση τιμή όλων των τιμών του AP και προκύπτει η μετρική MAP.

$$MAP = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{Q_j}\left(\sum_{i=1}^{Q_j} P(doc_i)\right)$$

Τέλος, δημιουργούνται διαγράμματα και για τις δύο μετρικές και συγκρίνονται μεταξύ τους με υπολογισμό του AUC – Area Under the Curve.

Αποτελέσματα - Παρατηρήσεις

Η συλλογή που δίνεται έχει τίτλο Cystic Fibrosis (C.F.), που αποτελείται από 1209 κείμενα και τα αρχεία queries_20, relevant_20, cfquery_detailed, τα οποία περιλαμβάνουν τα ερωτήματα, τα σχετικά κείμενα ανά ερώτημα και τα ερωτήματα με τα αντίστοιχα αποτελέσματα.

Σε κάθε βήμα εκτυπώνονται και στα δύο μοντέλα οι επιμέρους υπολογισμοί, οι οποίοι είναι διαθέσιμοι στο terminal, κατά την εκτέλεση του κάθε κώδικα. Για ορισμένους υπολογισμούς, οι οποίοι δεν χωρούσαν στο terminal, είτε γίνεται εξαγωγή σε csv, είτε ειδικοί έλεγχοι. Για παράδειγμα, στο tf των query στο VSM, γίνεται εκτύπωση του συνόλου των «0» και «1» στον πίνακα που δημιουργείται.

Ερώτημα 1ο (Inverted Index): Ακολουθεί εκτύπωση από το inverted index.

```
EACTIVE': [('00986', 2)], 'AUREUSSPECIFIC': [('00986', 1)], 'GENUS': [
('00986', 1), ('00270', 1)], 'SERINE': [('01068', 1), ('00139', 1), ('
01136', 1)], '4METHYLUMBELLIFERYL': [('01068', 1), ('01130', 1), ('011
36', 1)], 'GUANIDINOBENZOATE': [('01068', 1), ('01130', 1), ('01136',
1)], 'GLUCOCORTICOID': [('01068', 1), ('00840', 1)], 'SENSITIVITYTEST'
: [('00718', 1)], 'LATELY': [('00718', 1)], '14CSPERMIDINE': [('00972'
, 3), ('01198', 1)], 'CONJUGATION': [('00972', 2)], 'STRICTLY': [('007
20', 1)], 'QUARTERS': [('00512', 1)], 'CROSSINFECTION': [('00176', 1)]
, 'CONTINUOUSLY': [('00176', 1), ('01154', 1)], 'SEROGROUP': [('00176'
, 1)], 'TENDING': [('00176', 1)], 'UNSTABLE': [('00176', 2)], 'NM': [(
'00176', 5), ('00766', 1)], 'DISSOCIANTS': [('00176', 1), ('01086', 1)
], 'SELECT': [('00176', 1), ('01086', 1)], 'ELECTROENDOSMOTIC': [('003
44', 1)], 'COMMERCIALLY': [('00344', 1), ('00395', 1)], 'PHADEBAS': [(
'00344', 1)], 'DYESTARCH': [('00344', 1)], 'DETECTS': [('00344', 1), (
'00225', 1)], 'MACROAMYLASEMIA': [('00344', 1)], 'MAXILLOFACIAL': [('0
0344', 1)], 'ACETYLCHOLINESTERASE': [('01050', 3)], 'ADENOSINETRIPHOSP
HATASE': [('01050', 2)], 'VAGUE': [('00182', 1)], 'ASSOCITED': [('0018
2', 1)], 'POLYCATION': [('00975', 1)], 'POLYELECTROLYTES': [('00975',
1)], 'MGM': [('00975', 1)], 'LINKING': [('00975', 1)], 'POLYANIONIC':
[('00975', 1)], 'CROSSLINKING': [('00975', 1)], 'GLOBULE': [('00975',
1)], 'GOVERNS': [('00975', 1)], 'POLY': [('00975', 1)], 'POLYCATIONS':
 [('00975', 1)], 'FEBRILE': [('00149', 1)], 'TECHNETIUM99MSULFUR': [('
00149', 1)], 'TUMOR': [('00149', 1), ('01191', 1)], 'PITFALLS': [('001
49', 1)], 'RETROSPECTIVELY': [('00149', 1)], 'SPLENIC': [('00149', 1)]
, 'IODINE131ROSE': [('00149', 1)], '254': [('00149', 1)], 'NONSPECIFIC
```

Inverted Index

Ερώτημα 2ο (VSM):

```
                          IDF
WE                   0.675886
IS                   0.216730
BLOOD                1.137944
SECRETION            1.207365
THIS                 0.317503
...                       ...
BRONCHORRHEIC        3.082426
THANKS               3.082426
RADIOCINEMATOGRAPHIC 3.082426
INEFFICACY           3.082426
AUTOGENIC            3.082426

[11367 rows x 1 columns]
             WE        IS     BLOOD  SECRETION      THIS  ...  TOMOGRAPHIC     HUMID  CLIMATES  DERANGEMENTS  FIBROSISMECONIUM
01128  0.879348  0.216730  1.480499   1.207365  0.317503  ...     0.000000  0.000000  0.000000      0.000000          0.000000
00658  0.879348  0.281973  0.000000   0.000000  0.317503  ...     0.000000  0.000000  0.000000      0.000000          0.000000
01117  0.879348  0.320137  0.000000   0.000000  0.317503  ...     0.000000  0.000000  0.000000      0.000000          0.000000
00694  1.148310  0.216730  0.000000   1.570818  0.317503  ...     0.000000  0.000000  0.000000      0.000000          0.000000
00497  0.675886  0.000000  0.000000   1.207365  0.317503  ...     0.000000  0.000000  0.000000      0.000000          0.000000
...         ...       ...       ...        ...       ...  ...          ...       ...       ...           ...               ...
00624  0.000000  0.000000  0.000000   0.000000  0.000000  ...     0.000000  0.000000  0.000000      0.000000          0.000000
01097  0.000000  0.000000  0.000000   0.000000  0.000000  ...     0.000000  0.000000  0.000000      0.000000          0.000000
00558  0.000000  0.000000  0.000000   0.000000  0.000000  ...     3.082426  0.000000  0.000000      0.000000          0.000000
01219  0.000000  0.000000  0.000000   0.000000  0.000000  ...     0.000000  3.082426  3.082426      3.082426          0.000000
```

IDF και Weights

```
    WE  IS  BLOOD  SECRETION  THIS  INVESTIGATION  ...  EXHAUSTING  BRONCHORRHEIC  THANKS  RADIOCINEMATOGRAPHIC  INEFFICACY  AUTOGENIC
1    0   0      0          0     0              0  ...           0              0       0                      0           0          0
2    0   0      0          0     0              0  ...           0              0       0                      0           0          0
3    0   0      0          0     0              0  ...           0              0       0                      0           0          0
4    0   1      0          0     0              0  ...           0              0       0                      0           0          0
5    0   1      0          0     0              0  ...           0              0       0                      0           0          0
6    0   1      0          0     0              0  ...           0              0       0                      0           0          0
7    0   0      0          0     0              0  ...           0              0       0                      0           0          0
8    0   0      0          0     0              0  ...           0              0       0                      0           0          0
9    0   1      0          0     0              0  ...           0              0       0                      0           0          0
10   0   1      0          0     0              0  ...           0              0       0                      0           0          0
11   0   1      0          0     0              0  ...           0              0       0                      0           0          0
12   0   0      0          0     0              0  ...           0              0       0                      0           0          0
13   0   0      0          0     0              0  ...           0              0       0                      0           0          0
14   0   0      0          0     0              0  ...           0              0       0                      0           0          0
15   0   0      0          0     0              0  ...           0              0       0                      0           0          0
16   0   0      0          0     0              0  ...           0              0       0                      0           0          0
17   0   1      0          0     0              0  ...           0              0       0                      0           0          0
18   0   1      0          0     0              0  ...           0              0       0                      0           0          0
19   0   0      0          0     0              0  ...           0              0       0                      0           0          0
20   0   1      0          0     0              0  ...           0              0       0                      0           0          0

[20 rows x 11367 columns]
Συνολικά 0: 227079
Συνολικά 1: 261
```

Queries TF

```
[20 rows x 11367 columns]
      01128     00658     01117     00694     00497     01121  ...     00878     00624     01097         00558         01219     00772
1  0.009568  0.041137  0.006936  0.010610  0.085707  0.020662  ...  0.030684  0.008208  0.000001  2.375910e-03  1.309088e-06  0.000000
2  0.004126  0.031456  0.029349  0.018968  0.050053  0.031663  ...  0.021973  0.014093  0.009290  1.312716e-02  1.388032e-02  0.000008
3  0.010532  0.034364  0.031743  0.007447  0.098352  0.054633  ...  0.004493  0.021045  0.000001  2.615770e-03  1.785886e-02  0.000000
4  0.006952  0.007140  0.007471  0.004537  0.096069  0.000742  ...  0.004317  0.008678  0.000001  1.153293e-06  1.384086e-06  0.000000
5  0.013598  0.071256  0.014612  0.008872  0.079317  0.040394  ...  0.008441  0.016975  0.000000  0.000000e+00  0.000000e+00  0.000000
6  0.004718  0.013450  0.005279  0.009995  0.101111  0.003445  ...  0.019054  0.007899  0.002755  1.509996e-03  8.319834e-07  0.000000
7  0.010277  0.035079  0.005329  0.007923  0.094974  0.024899  ...  0.006185  0.014787  0.000113  1.915957e-03  1.245839e-02  0.000009
8  0.008904  0.013449  0.037624  0.008184  0.006059  0.011409  ...  0.003131  0.048533  0.000009  3.556486e-03  3.757885e-03  0.000000
9  0.006246  0.055913  0.015124  0.004090  0.003799  0.007081  ...  0.090956  0.018292  0.000018  1.436078e-05  2.058875e-05  0.000011
10 0.008287  0.008278  0.009235  0.005754  0.005546  0.000771  ...  0.191724  0.008965  0.000197  1.190008e-06  1.808891e-04  0.000013
11 0.000764  0.007044  0.029265  0.023254  0.000016  0.016953  ...  0.000012  0.010503  0.000011  1.414877e-05  1.317300e-05  0.108204
12 0.011006  0.032156  0.036479  0.010230  0.007755  0.024540  ...  0.003606  0.056161  0.000008  4.084271e-03  4.324258e-03  0.000010
13 0.009935  0.010201  0.007205  0.007026  0.011575  0.003000  ...  0.004244  0.012906  0.004506  1.131744e-06  9.420160e-06  0.000012
14 0.053322  0.019524  0.115474  0.018205  0.005283  0.001750  ...  0.005456  0.033947  0.002677  2.118815e-02  2.030285e-02  0.000012
15 0.009500  0.044325  0.006205  0.006377  0.011182  0.003144  ...  0.004448  0.013540  0.004723  1.188283e-06  1.426079e-06  0.000000
16 0.006466  0.009410  0.004230  0.014093  0.005326  0.000008  ...  0.017114  0.006095  0.000010  1.225875e-05  1.163073e-05  0.087571
17 0.054411  0.016347  0.021327  0.012892  0.004142  0.001207  ...  0.004281  0.008040  0.000007  9.106723e-07  7.580050e-06  0.000010
18 0.005875  0.067931  0.006417  0.004051  0.003929  0.000611  ...  0.003290  0.006332  0.000238  9.252641e-05  1.199511e-04  0.000000
19 0.078229  0.021243  0.032343  0.013200  0.010005  0.019098  ...  0.004651  0.009347  0.000010  5.270581e-03  5.580274e-03  0.000013
20 0.007316  0.036136  0.008152  0.005079  0.004896  0.000677  ...  0.024687  0.007912  0.000167  1.051477e-06  1.523414e-04  0.000000
```

Cosine Similarity

Ερώτημα 3ο (colBERT):

```
#> What are the hepatic complications or manifestations of CF
        [796]          21.4           795
        [107]          20.5           106
        [1032]                20.5           1031
        [135]          19.9           134
        [781]          19.5           780
        [248]          18.9           247
        [124]          18.8           123
        [860]          18.8           859
        [1176]                18.7           1175
        [673]          18.6           672
        [751]          18.6           750
        [644]          18.5           643
        [704]          18.4           703
        [641]          18.3           640
        [5]            18.3           4
        [913]          18.2           912
        [792]          18.1           791
        [215]          17.8           214
        [929]          17.7           928
        [177]          17.2           176
        [353]          16.9           352
        [446]          16.8           445
```

Ενδεικτικό αποτέλεσμα colBERT

```
#> What are the gastrointestinal complications of CF after the ne
        [796]          17.5           795
        [443]          17.3           442
        [1093]                17.3           1092
        [170]          17.3           169
        [46]           17.0           45
        [266]          17.0           265
        [851]          17.0           850
        [1163]                16.8           1162
        [774]          16.7           773
        [1089]                16.2           1088
        [387]          15.5           386
        [1031]                15.4           1030
        [242]          15.3           241
        [135]          15.2           134
        [878]          15.1           877
        [237]          14.9           236
        [55]           14.9           54
        [1137]                14.5           1136
        [81]           14.5           80
        [781]          14.5           780
        [492]          14.4           491
        [707]          14.2           706
```

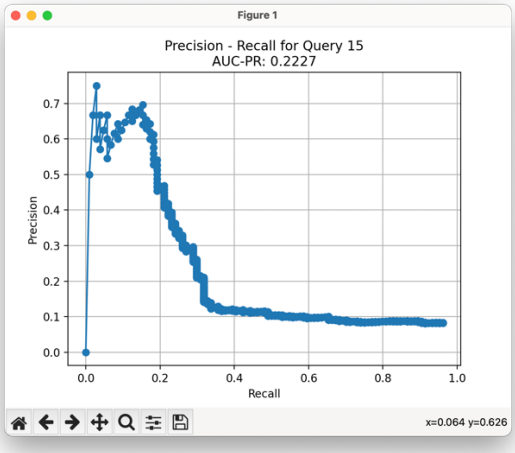
Ενδεικτικό Αποτέλεσμα colBERT (2)

```
#> What are the effects of calcium on the physical properties of mucus from CF patients
#> Can one distinguish between the effects of mucus hypersecretion and infection on the submucosal glands of the respiratory tract in CF
#> How are salivary glycoproteins from CF patients different from those of normal subjects
#> What is the lipid composition of CF respiratory secretions
#> Is CF mucus abnormal
#> What is the effect of water or other therapeutic agents on the physical properties viscosity elasticity of sputum or bronchial secretions from CF patients
#> Are mucus glycoproteins degraded differently in CF patients as compared to those from normal subjects
#> What histochemical differences have been described between normal and CF respiratory epithelia
#> What is the association between liver disease cirrhosis and vitamin A metabolism in CF
#> What is the role of Vitamin E in the therapy of patients with CF
#> What is the difference between meconium ileus and meconium plug syndrome
#> What abnormalities of amino acid transport have been described in the small bowel of CF patients
#> What are the clinical or biochemical features of pancreatitis in CF patients
#> What non-invasive tests can be performed for the evaluation of exocrine pancreatic function in patients with CF
#> What are the hepatic complications or manifestations of CF
#> What are the gastrointestinal complications of CF after the neonatal period exclude liver disease and meconium ileus
#> What is the most effective regimen for the use of pancreatic enzyme supplements in the treatment of CF patients
#> Is dietary supplementation with bile salts of therapeutic benefit to CF patients
#> What complications of pancreatic enzyme therapy have been reported in CF patients
```

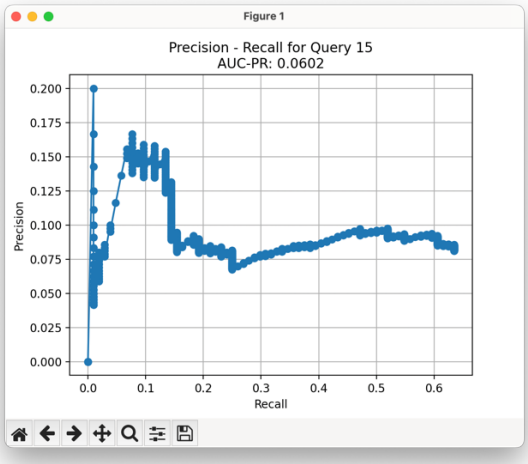
Εισαγωγή και εκτύπωση των Queries

Ερώτημα 4ο (Μετρικές Αξιολόγησης):

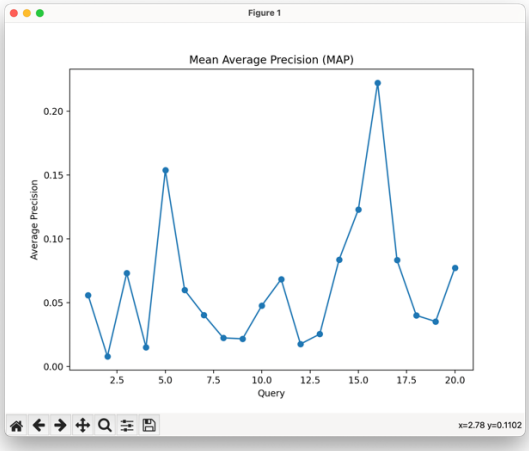
Διαισθητικά, οι μετρικές του colBERT θα έπρεπε να είναι καλύτερες και το AUC για το colBERT μεγαλύτερο, αλλά από τα αποτελέσματα υπερέχει το VSM. Ενδεικτικά ακολουθούν τα αποτελέσματα των μετρικών.



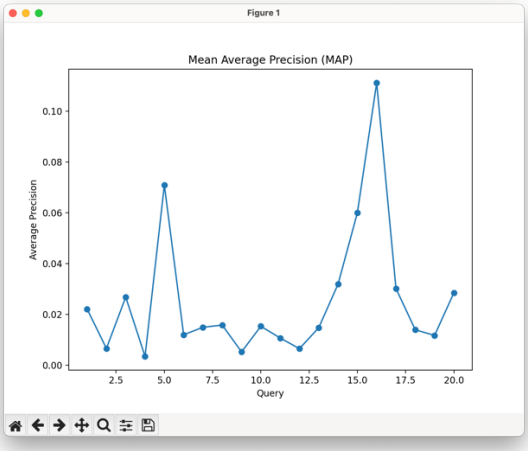
VSM – Precion/Recall



colBERT – Precision/Recall



VSM – MAP



colBERT – MAP

Πίνακας AUC για VSM	Πίνακας AUC για colBERT
Query 1: AUC-PR = 0.2208	Query 1: AUC-PR = 0.0242
Query 2: AUC-PR = 0.1536	Query 2: AUC-PR = 0.0078
Query 3: AUC-PR = 0.1556	Query 3: AUC-PR = 0.0247
Query 4: AUC-PR = 0.1063	Query 4: AUC-PR = 0.0049
Query 5: AUC-PR = 0.2665	Query 5: AUC-PR = 0.0686
Query 6: AUC-PR = 0.2033	Query 6: AUC-PR = 0.0140
Query 7: AUC-PR = 0.1091	Query 7: AUC-PR = 0.0198
Query 8: AUC-PR = 0.0315	Query 8: AUC-PR = 0.0247
Query 9: AUC-PR = 0.1506	Query 9: AUC-PR = 0.0073
Query 10: AUC-PR = 0.4064	Query 10: AUC-PR = 0.0174
Query 11: AUC-PR = 0.5443	Query 11: AUC-PR = 0.0097
Query 12: AUC-PR = 0.1110	Query 12: AUC-PR = 0.0074
Query 13: AUC-PR = 0.0932	Query 13: AUC-PR = 0.0215
Query 14: AUC-PR = 0.1394	Query 14: AUC-PR = 0.0273
Query 15: AUC-PR = 0.2227	Query 15: AUC-PR = 0.0602
Query 16: AUC-PR = 0.2760	Query 16: AUC-PR = 0.1100
Query 17: AUC-PR = 0.1442	Query 17: AUC-PR = 0.0366
Query 18: AUC-PR = 0.2487	Query 18: AUC-PR = 0.0150
Query 19: AUC-PR = 0.0700	Query 19: AUC-PR = 0.0124
Query 20: AUC-PR = 0.4468	Query 20: AUC-PR = 0.0307
Mean Average Precision (MAP): 0.06368606844635437	Mean Average Precision (MAP): 0.025596822102431464

Βιβλιογραφία

[1] A. W. C. S. Y. G. Salton, «ACM Digital Library,» Νοέμβριος 1975. [Ηλεκτρονικό]. Available: https://dl.acm.org/doi/pdf/10.1145/361219.361220.

[2] O. K. J. S.-F. C. P. M. Z. Keshav Santhanam, «ACL Anthology,» 10 Ιούλιος 2022. [Ηλεκτρονικό]. Available: https://aclanthology.org/2022.naacl-main.272.pdf.

[3] B. R. -. N. Ricardo Baeza - Yates, Ανάκτηση Πληροφορίας, Οι έννοιες και η τεχνολογία πίσω από την ανάκτηση, Εκδήσεις Τζίολα, 2015.

[4] S. V. T. R. S. W.-K. H. J. S. V. S. C. Manish Patil, «ACM Digital Library,» Ιούλιος 2011. [Ηλεκτρονικό]. Available: https://dl.acm.org/doi/pdf/10.1145/2009916.2009992.

[5] «GitHub,» [Ηλεκτρονικό]. Available: https://github.com/stanford-futuredata/ColBERT.

[6] «Google Colab,» [Ηλεκτρονικό]. Available: https://colab.research.google.com/github/stanford-futuredata/ColBERT/blob/main/docs/intro2new.ipynb.

Παράρτημα

Link for colBERT: https://colab.research.google.com/drive/1vD5JI9yTX3a6B1L-TK7rlHe9mB_S3-Jb?usp=sharing

(Ο κώδικας υπάρχει και παρακάτω)

Vector Space Model (Inverted index and metrics included)

```
import os
from collections import defaultdict
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import auc

inverted_index = defaultdict(list)

folder_path = '/Users/mariakouri/Desktop/Ανάκτηση Πληροφορίας/docs'

for filename in os.listdir(folder_path):
    file_path = os.path.join(folder_path, filename)

    if os.path.isfile(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            text = file.read()

        words = text.split()

        for word in set(words):
            inverted_index[word].append((filename, words.count(word)))

# Υπολογισμός της συχνότητας εμφάνισης κάθε λέξης
word_counts = defaultdict(int)
for word, documents in inverted_index.items():
    word_counts[word] = len(documents)

print(inverted_index)

# Υπολογισμός του tf και αποθήκευση σε ένα dictionary
tf_values = {}
for word, occurrences in inverted_index.items():
    for document, count in occurrences:
        tf = 1 + math.log10(count)
```

```
        if document in tf_values:
            tf_values[document][word] = tf
        else:
            tf_values[document] = {word: tf}

# Δημιουργία του DataFrame για το tf
tf_df = pd.DataFrame.from_dict(tf_values, orient='index')
tf_df.fillna(0, inplace=True)

# Υπολογισμός του idf και αποθήκευση σε ένα dictionary
idf_values = {}
total_documents = len(tf_df)
for word in inverted_index.keys():
    df = word_counts[word]
    idf = math.log10(total_documents / df) if df > 0 else 0
    idf_values[word] = idf

# Δημιουργία του DataFrame για το idf
idf_df = pd.DataFrame.from_dict(idf_values, orient='index', columns=['IDF'])

# Εμφάνιση του DataFrame
print(idf_df)

# Βάρη (weights)
weight_values = {}

for document, word_tf_values in tf_values.items():
    weight_values[document] = {}
    for word, tf in word_tf_values.items():
        weight = tf * idf_values[word]
        weight_values[document][word] = weight

# Δημιουργία του DataFrame για τα βάρη (weights)
weight_df = pd.DataFrame.from_dict(weight_values, orient='index')

# Αντικατάσταση NaN τιμών με 0
weight_df.fillna(0, inplace=True)

# Εμφάνιση του DataFrame με τα βάρη
print(weight_df)

# Διαδρομή του αρχείου ερωτημάτων
query_file_path = '/Users/mariakouri/Desktop/Ανάκτηση Πληροφορίας/Queries_20'

# Διάβασμα του αρχείου ερωτημάτων
with open(query_file_path, 'r', encoding='utf-8') as query_file:
    queries_list = query_file.readlines()
```

```
# Αρχικοποίηση του DataFrame
query_df = pd.DataFrame(index=range(1, len(queries_list) + 1),
columns=inverted_index.keys())
query_df = query_df.fillna(0)  # Αρχικά, όλα τα κελιά είναι 0

# Επεξεργασία των queries
for i, query in enumerate(queries_list, start=1):
    query_words = query.strip().split()  # Διαχωρισμός του query σε λέξεις
    for word in query_words:
        if word.upper() in query_df.columns:
            query_df.at[i, word.upper()] = 1  # Αν η λέξη υπάρχει στο query, θέτουμε
το κελί σε 1

# Εμφάνιση του DataFrame
print(query_df)

# Ονομασία του αρχείου εξόδου
output_file_path = '/Users/mariakouri/Desktop/Ανάκτηση Πληροφορίας/QueryResults.txt'

# Αποθήκευση του DataFrame στο αρχείο κειμένου
with open(output_file_path, 'w', encoding='utf-8') as output_file:
    output_file.write(query_df.to_string())

# Υπολογισμός συνολικού αριθμού μηδενικών και μονάδων
total_zeros = query_df.eq(0).sum().sum()
total_ones = query_df.eq(1).sum().sum()

# Εμφάνιση των αποτελεσμάτων
print(f"Συνολικά 0: {total_zeros}")
print(f"Συνολικά 1: {total_ones}\n")

# Δημιουργία του DataFrame για το query_idf
query_idf_values = {}

for document, word_tf_values in query_df.iterrows():
    query_idf_values[document] = {}
    for word, tf in word_tf_values.items():
        query_idf_values[document][word] = tf * idf_values[word]  #όπου tf η συχνότητα
του όρου στα query με βάση τον πίνακα

# Δημιουργία του DataFrame για το query_idf
query_idf_df = pd.DataFrame.from_dict(query_idf_values, orient='index')

# Αντικατάσταση NaN τιμών με 0
query_idf_df.fillna(0, inplace=True)

# Εμφάνιση του DataFrame με το query_idf
print(query_idf_df)
```

```
# Πολλαπλασιασμός των DataFrames
cosine_sim = query_idf_df.dot(weight_df.T)

# Ευκλείδειες νόρμες των γραμμών των DataFrames
query_idf_norm = np.linalg.norm(query_idf_df.values, axis=1)
weight_norm = np.linalg.norm(weight_df.values, axis=1)

# Διαίρεση με το γινόμενο των ευκλείδιων νορμών
cosine_sim = cosine_sim.divide(np.outer(query_idf_norm, weight_norm))

# Εμφάνιση του αποτελέσματος
print(cosine_sim)


# Δημιουργία dictionary για αποθήκευση των relevant documents για κάθε query
relevant_docs_dict = {}

# Διαδρομή του αρχείου
relevant_documents_path = '/Users/mariakouri/Desktop/Ανάκτηση Πληροφορίας/Relevant_20'

# Συμπλήρωση του relevant_docs_dict με relevant documents για κάθε query
with open(relevant_documents_path, 'r', encoding='utf-8') as relevant_file:
    for query_id, line in enumerate(relevant_file, start=1):
        parts = line.strip().split()
        if not parts:
            continue  # Παράληψη κενών γραμμών
        relevant_docs = set(map(int, parts))  # Μετατροπή των relevant document IDs σε
ακεραίους
        relevant_docs_dict[query_id] = relevant_docs


# Επανάληψη μέσω των queries στο relevant_docs_dict
for query_id, relevant_docs_set in relevant_docs_dict.items():
    # Λήψη των ταξινομημένων τιμών και των αντίστοιχων κειμένων για το τρέχον query
    row = cosine_sim.loc[query_id]
    sorted_values = row.sort_values(ascending=False).values
    sorted_texts = row.sort_values(ascending=False).index

# Δημιουργία λιστών για την αποθήκευση αποτελεσμάτων των precision και recall για κάθε
query
precision_results = []
recall_results = []

# Επανάληψη μέσω των queries στο relevant_docs_dict
for query_id, relevant_docs_set in relevant_docs_dict.items():
    # Λήψη των ταξινομημένων τιμών και των αντίστοιχων κειμένων για το τρέχον query
    row = cosine_sim.loc[query_id]
```

```
    sorted_values = row.sort_values(ascending=False).values
    sorted_texts = row.sort_values(ascending=False).index

    # Υπολογισμός των precision και recall για διάφορα thresholds
    precisions = []
    recalls = []

    for threshold in range(1, len(sorted_texts) + 1):  # Χρήση όλων των ανακτημένων
εγγράφων
        # Ανάκτηση του συνόλου των ανακτημένων εγγράφων με βάση το threshold
        retrieved_docs = set(map(int, sorted_texts[:threshold]))

        # Υπολογισμός precision and recall
        if len(relevant_docs_set) > 0:
            precision = len(relevant_docs_set.intersection(retrieved_docs)) /
len(retrieved_docs)
        else:
            precision = 0

        recall = len(relevant_docs_set.intersection(retrieved_docs)) /
len(relevant_docs_set) if len(relevant_docs_set) > 0 else 0

        # Προσθήκη των precision and recall στις λίστες
        precisions.append(precision)
        recalls.append(recall)

    # Προσθήκη των λιστών precision and recall στα αποτελέσματα
    precision_results.append(precisions)
    recall_results.append(recalls)

# Δημιουργία λίστας για την αποθήκευση του εμβαδού (AUC-PR) για κάθε query
auc_pr_results = []

# Επανάληψη μέσω των queries στο relevant_docs_dict
for query_id, (precision_values, recall_values) in enumerate(zip(precision_results,
recall_results), start=1):

    # Υπολογισμός του εμβαδού κάτω την precision-recall καμπύλη
    auc_pr = auc(recall_values, precision_values)

    # Προσθήκη του AUC-PR στα αποτελέσματα
    auc_pr_results.append(auc_pr)

    # Δημιουργία γραφήματος για κάθε query
    plt.figure()
    plt.plot(recall_values, precision_values, marker='o')
    plt.title(f"Precision - Recall for Query {query_id}\nAUC-PR: {auc_pr:.4f}")
    plt.xlabel("Recall")
```

```
    plt.ylabel("Precision")
    plt.grid(True)
    plt.show()

# Εκτύπωση των αποτελεσμάτων AUC-PR για κάθε query
for query_id, auc_pr in enumerate(auc_pr_results, start=1):
    print(f"Query {query_id}: AUC-PR = {auc_pr:.4f}")


# Υπολογισμός του Mean Average Precision (MAP)
average_precision_values = []

for precisions, relevant_docs_set in zip(precision_results,
relevant_docs_dict.values()):
    # Υπολογισμός του AP για το query
    average_precision = 0
    for i, precision in enumerate(precisions):
        if i + 1 in relevant_docs_set:
            average_precision += precision

    average_precision /= len(relevant_docs_set)

    # Καταχώρηση της τιμής AP στη λίστα
    average_precision_values.append(average_precision)

# Υπολογισμός του Mean Average Precision (MAP)
map_value = np.mean(average_precision_values)

# Εκτύπωση της τιμής MAP
print(f"Mean Average Precision (MAP): {map_value}")

# Σχεδίαση του MAP ως γραμμή
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(average_precision_values) + 1), average_precision_values,
marker='o', linestyle='-')
plt.xlabel('Query')
plt.ylabel('Average Precision')
plt.title('Mean Average Precision (MAP)')
plt.show()
```

colBERT (running: google colab)

```
import os

def read_documents_from_directory(directory_path):
```

```
    documents = []
    for filename in os.listdir(directory_path):
        with open(os.path.join(directory_path, filename), 'r', encoding="utf8") as
file:
            text = file.read()
            document_id = len(documents) + 1  # Εκχώρηση ενός μοναδικού ID με βάση τη
θέση
            document_tuple = (document_id, text)
            documents.append(document_tuple)
    return documents

# Διαδρομή των αρχείων
docs_directory = '/content/docs'
queries_file = '/content/drive/MyDrive/Queries_20'

# Διάβασμα των documents και queries
documents = read_documents_from_directory(docs_directory)

# Διάβασμα των queries από τον φάκελο
def read_queries_from_file(file_path):
    queries = []
    with open(file_path, 'r', encoding="utf8") as file:
        text = file.read()
        queries_list = text.split('\n')
        queries_list = [query.strip() for query in queries_list if query.strip()]
        queries.extend(queries_list)
    return queries
# Read queries from file
queries = read_queries_from_file(queries_file)

# Ρύθμιση των απαιτήσεων του ColBERT
nbits = 2
doc_maxlen = 300
max_id = 1209
index_name = 'index'
checkpoint = '/content/drive/MyDrive/colbertv2.0'

# Indexing
with Run().context(RunConfig(nranks=1, experiment='indexing')):
    config = ColBERTConfig(doc_maxlen=doc_maxlen, nbits=nbits, kmeans_niters=4)
    indexer = Indexer(checkpoint=checkpoint, config=config)
    indexer.index(name=index_name, collection=[doc[1] for doc in documents[:max_id]],
overwrite=True)

# Searching
with Run().context(RunConfig(experiment='indexing')):
    searcher = Searcher(index=index_name, collection=[doc[1] for doc in documents])
```

```python
# Επανάληψη για όλα τα queries
for query in queries:
    print(f"#> {query}")

    # Εύρεση των top-1209 passages για κάθε query
    results = searcher.search(query, k=1209)
# Επανάληψη για όλα τα queries
for query in queries:
    print(f"#> {query}")

    # Εύρεση των passages για κάθε query
    results = searcher.search(query, k=1209)  # Ανάκτση όλων των passages

    # Εκτύπωση των passages που ανακτήθηκαν με σχετικό score
    for i, (passage_id, passage_rank, passage_score) in enumerate(zip(*results)):
        document_id = documents[passage_id][0]
        print(f"\t [{document_id}] \t\t {passage_score:.1f} \t\t {passage_id}")


import csv

# Δημιουργία dictionary για αποθήκευση λιστών των document IDs για κάθε query
query_document_lists = {}


# Επανάληψη για όλα τα queries
for query in queries:
    print(f"#> {query}")

    # Εύρεση των passages για κάθε query
    results = searcher.search(query, k=1209)  # Ανάκτση όλων των passages

    # Προσθήκη των αποτελεσμάτων στην λίστα
    result_list = []
    for i, (passage_id, passage_rank, passage_score) in enumerate(zip(*results)):
        result_list.append(passage_id)

    # Καταχώρηση της λίστας για το τέχον querie στο dictionary
    query_document_lists[query] = result_list

# Εκτύπωση του dictionary
print(query_document_lists)

# Αποθήκευση του dictionary σε ένα CSV έγγραφο
csv_file_path = '/content/colbert_result.csv'
with open(csv_file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
```

```
    # Εγγραφή του header στο αρχείο
    writer.writerow(['Query', 'DocumentIDs'])

    # Εγγραφή των data στο αρχείο
    for query, document_list in query_document_lists.items():
        writer.writerow([query, document_list])

print(f"Query document lists saved to {csv_file_path}")
```

colBERT’s Metrics (running: VS Code)