ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Ανάκτηση Πληροφορίας

Εργαστηριακή Άσκηση

Χειμερινό Εξάμηνο 2023

Σεϊτανίδης Νικόλαος 1072553 up1072553@ac.upatras.gr

Οικονομόπουλος Ιωάννης 1072582 up1072582@ac.upatras.gr

Περιεχόμενα

Εισαγωγή

Θεωρητικά για τα μοντέλα

Αρχικά να αναφερθεί πως έχουν υλοποιηθεί όλα τα ζητούμενα. Πιο συγκεκριμένα, έχει δημιουργηθεί ένα ανεστραμμένο αρχείο, το μοντέλο Vector Space, το μοντέλο ColBERT και τέλος υπολογίζονται οι μετρικές MAP, MRR και διαγράμματα ανάκλησης-ακρίβειας για τα δύο μοντέλα. Πιο συγκεκριμένα για την υλοποίηση του κάθε μοντέλου αναφέρεται στο section **Υλοποίηση**.

Τεχνικές και βιβλιοθήκες με αναφορές

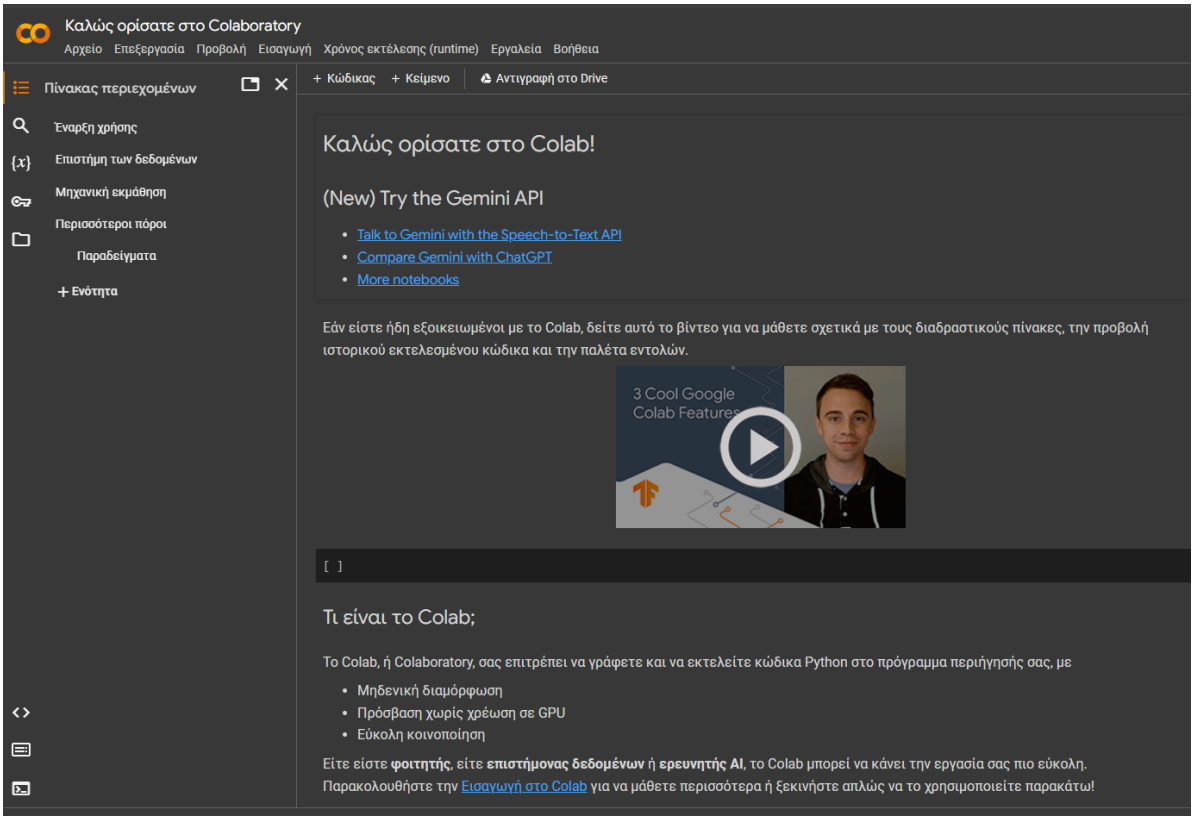
Η εργασία υλοποιήθηκε σε Python στο Visual Studio Code και στο Google Colab. Παραθέτουμε για κάθε ερώτημα τις βιβλιοθήκες που χρειάστηκαν:

- **Collections**: Βιβλιοθήκη απαραίτητη για την δημιουργία ενός counter.
- **Os**: Βιβλιοθήκη απαραίτητη για να πάρουμε πληροφορίες από ένα αρχείο και συγκεκριμένα για να ελεγχθεί εάν υπάρχει ένα αρχείο σε ένα συγκεκριμένο path.
- **Numpy**: Βιβλιοθήκη απαραίτητη για την δημιουργία πινάκων και διανυσμάτων.
- **Pandas**: Βιβλιοθήκη απαραίτητη για την δημιουργία dataframe για την υλοποίηση του colBERT.
- **Math**: Βιβλιοθήκη απαραίτητη για κάθε μαθηματική πράξη.
- **Re**: Βιβλιοθήκη απαραίτητη για τον καθαρισμό του κειμένου.
- **Ntlk**: Βιβλιοθήκη απαραίτητη για να επεξεργαστούμε τις λέξεις των αρχείων και συγκεκριμένα για να κάνουμε import μέσω αυτής τις 2 παρακάτω βιβλιοθήκες.
- **Stopwords**: Βιβλιοθήκη απαραίτητη για την αφαίρεση λέξεων οι οποίες δεν περιέχουν καμία πληροφορία και απλά συνδέουν τις λέξεις σε μια πρόταση που περιέχουν πληροφορία.
- **PorterStemmer, WordNetLemmatizer:** Βιβλιοθήκες απαραίτητη για να αφαιρεθεί το κομμάτι της λέξης (στην περίπτωση μας με τον τρόπο που έχει οριστεί στον κώδικα για λέξεις χωρίς stopwords) το οποίο αφορά καταλήξεις την λέξης ή ενικό και πληθυντικό αριθμό, ώστε να μείνει το μέρος της κάθε λέξης το οποίο περιέχει την μέγιστη πληροφορία την οποία χρειαζόμαστε.

- **Matplotlib:** Βιβλιοθήκη απαραίτητη για την εκτύπωση του διαγράμματος recall-precision για τον υπολογισμό των μετρικών.
- **Sys; sys.path.insert(0, 'ColBERT/'):** Βιβλιοθήκη απαραίτητη για την δημιουργία του ColBERT μοντέλου και την εισαγωγή του σε ένα μονοπάτι ώστε να είναι συμβατό με το google colab.

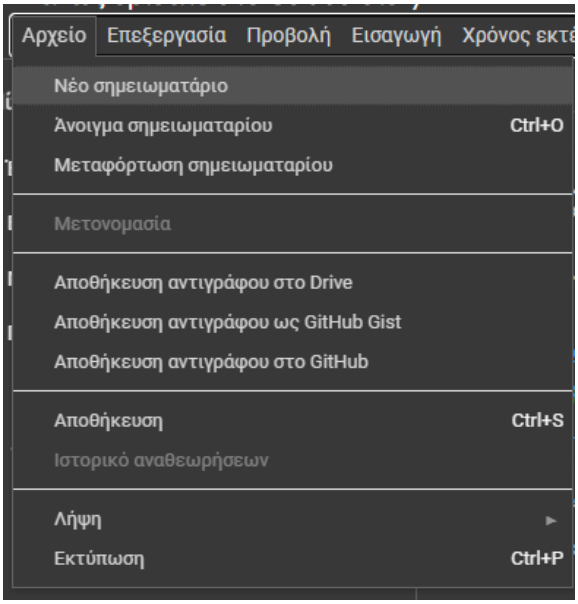
Σχετικά με το περιβάλλον Google Colab

Το περιβάλλον χρησιμοποιήθηκε για την υλοποίηση του μοντέλου ColBERT για λόγους που αναφέρονται στο κεφάλαιο υλοποίηση. Το google colab προσφέρει ένα περιβάλλον τύπου Notebook το οποίο προσδίδει δυνατότητες και εκτελέσεις για προγράμματα Python σαν να εκτελείται σε Linux/Unix λειτουργικό για χρήστες που βρίσκονται σε συστήματα με λειτουργικά Windows. Η πρόσβαση στο περιβάλλον γίνεται μέσω του web browser στην σελίδα: https://colab.research.google.com/
Κατά την είσοδο στην σελίδα εμφανίζεται μια ενημέρωση για το χρήστη προς τις δυνατότητες του περιβάλλοντος.



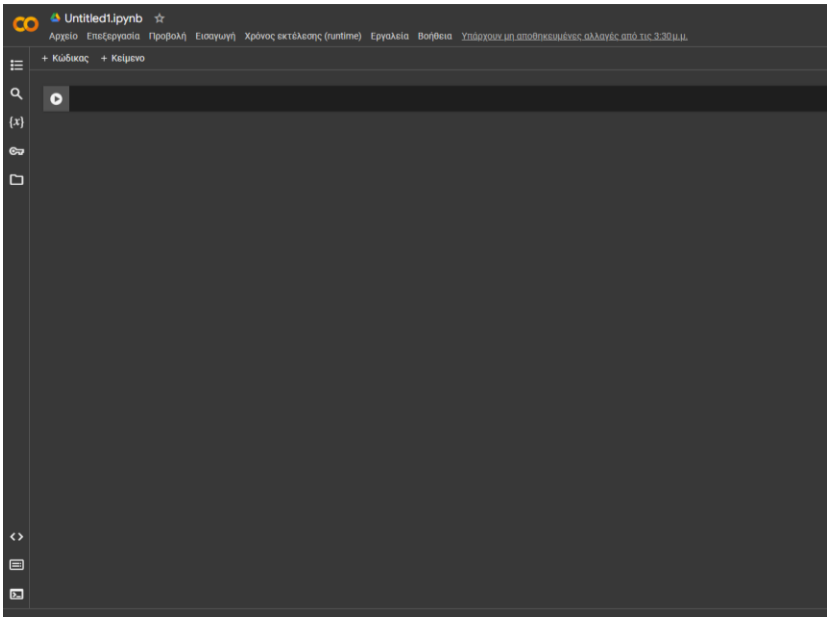
***Εικόνα 1**: Αρχική σελίδα του Google colab*

Για την δημιουργία ενός νέου σημειωματάριου ο χρήστης επιλέγει από το μενού το Αρχείο και έπειτα την επιλογή Νέο σημειωματάριο.



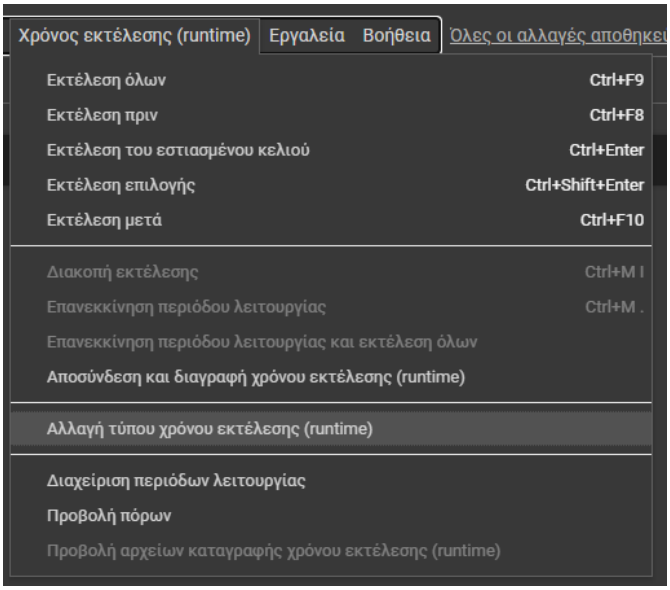
***Εικόνα 2**: Δημιουργία νέου σημειωματάριου*

Στη συνέχεια εμφανίζεται η σελίδα του κενού σημειωματάριου:



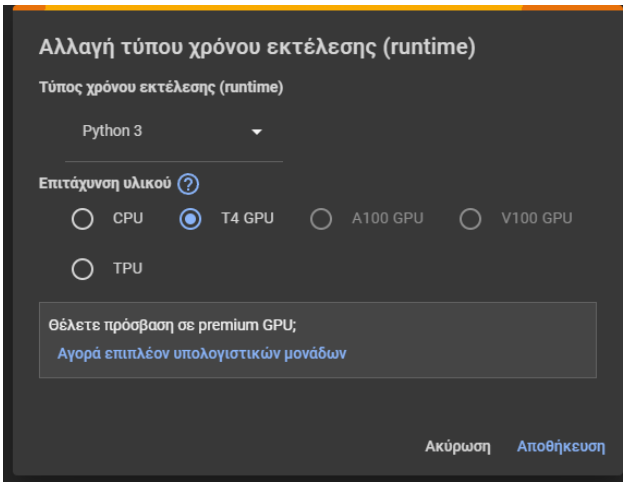
***Εικόνα 3**: Κενό σημειωματάριο*

Στη συνέχεια για την δημιουργία του ColBERT μοντέλο πρέπει να ρυθμιστεί ο χρόνος εκτέλεσης (Runtime) επιλέγοντας από το μενού την επιλογή αλλαγή τύπου χρόνου εκτέλεσης όπως φαίνεται παρακάτω:



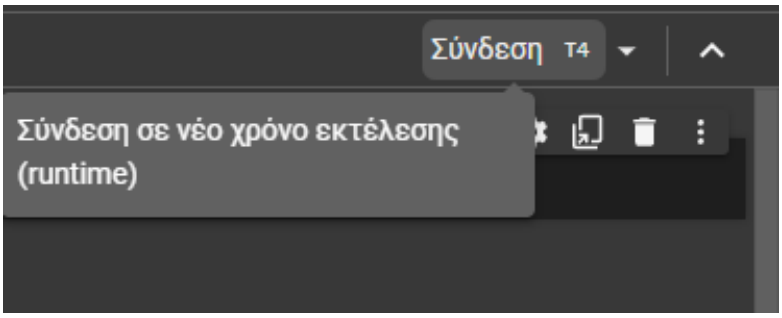
***Εικόνα 4**: Επιλογή για αλλαγή χρόνου εκτέλεσης*

Στην συνέχεια από τις επιλογές που εμφανίζονται επιλέγουμε για τύπο την Python 3 και για επιτάχυνση υλικού την επιλογή T4 GPU:



***Εικόνα 5**: Επιλογές χρόνου εκτέλεσης*

Τέλος επιλέγουμε την επιλογή σύνδεση από την πάνω δεξιά για να συνδεθούμε σε χρόνο εκτέλεσης και το περιβάλλον είναι έτοιμο για συγγραφή και εκτέλεση κώδικα.



***Εικόνα 6**: Σύνδεση χρόνου εκτέλεσης*

Υλοποίηση

Inverted Index

Αρχικά εισάγουμε τις απαραίτητες βιβλιοθήκες που αναφέρουμε παραπάνω. Ορίζουμε τον αριθμό από αρχεία τα οποία θα επεξεργαστούμε “N=7”, εννοώντας πως θα επεξεργαστούμε n+1 docs αρχεία, οπότε για N=7 επεξεργαζόμαστε 6 αρχεία (δουλεύει σωστά για όσα θέσουμε).
Για την δημιουργία του αντεστραμμένου ευρετηρίου δημιουργούνται 2 κλάσεις, η Appearance και η InvertedIndex. Ο σκοπός της Appearance ουσιαστικά είναι ώστε να καθορίσει την δομή δεδομένων σε κάθε document. Δηλαδή με αυτόν τον τρόπο μας εμφανίζει το docId και την συχνότητα στην οποία εμφανίζεται το συγκεκριμένο έγγραφο και ότι θα τα εκτυπώνει ως strings (μέσω της repr). Όσον αφορά την κλάση InvertedIndex, σε αυτή γίνεται ουσιαστικά η δημιουργία του ανεστραμμένου ευρετηρίου. Αρχικά ορίζονται οι συναρτήσεις init() και repr() όπου αρχικοποιούνται και υπολογίζονται ως string τα αντικείμενα. Αρχικά, για εξοικονόμηση χώρου και χρόνου, μας ενδιαφέρει να μειώσουμε τους χαρακτήρες και τις λέξεις σε κάθε αρχείο (και σε κάθε query όπως θα δούμε αργότερα). Για αυτόν τον λόγο, στην index_document() κάθε λέξη που διαβάζεται από κάθε αρχείο περνιέται ως όρισμα στην συνάρτηση clean_text() που καλούμε, η οποία ουσιαστικά καθαρίζει το κείμενο από λέξεις οι οποίες δεν περιέχουν κάποια πληροφορία, τα λεγόμενα “stopwords”, καθώς και από περιττούς χαρακτήρες (regular expression-re). Στην συνέχεια στην clean_text(), φιλτράρεται το περιεχόμενο χωρίς τα stopwords ώστε κάθε λέξη να μην περιέχει περιττή πληροφορία, κρατώντας την ρίζα της λέξης και πετώντας τις καταλήξεις οι οποίες δεν περιέχουν σημαντική πληροφορία για να χαρακτηρίζει την λέξη και την σημασία της. Αυτό γίνεται μέσω του Stemming και του Lemmatizer. Ύστερα στην index_document(), αφού έχουμε ξεχωρίσει κάθε λέξη από το κείμενο ενώ είναι “καθαρές” οι λέξεις, ορίζουμε έναν μετρητή που θα υπολογίζει την συχνότητα εμφάνισης κάθε λέξης εντός της λίστας. Για κάθε λέξη στην λίστα, αν ήδη υπάρχει στο ευρετήριο, έλεγξε μέσω του doc_id, αλλιώς πρόσθεσε την. Η μεταβλητή Inside όταν είναι ίση με 1 συμβολίζει πως η λέξη δεν υπάρχει στο συγκεκριμένο έγγραφο, ενώ όταν είναι ίση με 0 υπάρχει. Άρα συνολικά με τον παραπάνω τρόπο γνωρίζουμε τις εμφανίσεις κάθε λέξης για κάθε έγγραφο. Τέλος, καλείται η κλάση InvertedIndex και για όλα τα αρχεία με τίτλο ως και 5 ψηφία εντός του range 1 μέχρι N που έχουμε ορίσει την αρχή του κώδικα υπολογίζεται το ανεστραμμένο αρχείο.

Vector Space

Για την δημιουργία του Vector Space μοντέλου χρησιμοποιείται ο κώδικας του 1ου ερωτήματος αρχικά. Συμπληρωματικά, δημιουργείται μια συνάρτηση vector space που ουσιαστικά δημιουργεί το μοντέλο. Ουσιαστικά, δημιουργεί έναν πίνακα στον οποίον θα περιέχονται τα βάρη κάθε λέξης από κάθε κείμενο και στην 1η στήλη του θα περιέχονται όλες οι λέξεις, αφού έχουν οριστεί πρώτα οι διαστάσεις του πίνακα σε γραμμές και στήλες. Ύστερα υπολογίζονται τα βάρη tf-idf για κάθε λέξη βάσει του μαθηματικού τύπου που περιέχει το βιβλίο βάσει των συχνοτήτων και τον αριθμό των εγγράφων (idf = $\log_2$(1+(συνολικός αριθμός εγγράφων/αριθμός σχετικών εγγράφων)) και tf=1+$\log_2$(συχνότητα εμφάνισης)). Στην συνέχεια η συνάρτηση get_vectors() δημιουργεί τα διανύσματα βάσει των βαρών που υπολογίζονται παραπάνω. Αφού δημιουργηθεί μια λίστα όπου θα αποθηκευτούν τα βάρη ενός κειμένου, αυτή η λίστα θα μετατραπεί σε διάνυσμα. Αφού έχουμε το διάνυσμα για κάθε αρχείο, δημιουργούμε μια συνάρτηση cosine_similarity() που παίρνει ως όρισμα 2 διανύσματα και υπολογίζει την μεταξύ τους ομοιότητα μέσω του παρακάτω τύπου:

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Εικόνα 7*: Τύπος για τον υπολογισμό της ομοιότητας ανάμεσα σε 2 διανύσματα.*

Όμως, επειδή θέλουμε το 1ο όρισμα να είναι ένα query και το 2ο όρισμα να είναι ένα αρχείο, πρέπει να δημιουργήσουμε και μια συνάρτηση που θα μετατρέπει σε διάνυσμα όλα τα queries. Η vector_for_query() το κάνει αυτό με τον ίδιο τρόπο. Να σημειωθεί πως καλείται η clean_text() για να καθαριστεί το κάθε query και ύστερα υπολογίζεται κάθε βάρος για τα queries σε σχέση με τα υπόλοιπα διανύσματα. Έπειτα, αφού φορτώσουμε τα queries από το txt αρχείο και έχουν γίνει όλα τα παραπάνω, δημιουργείται ένα ευρετήριο για κάθε doc από τα docs, δημιουργείται το διάνυσμα για κάθε doc και για κάθε query αντίστοιχα. Τέλος, υπολογίζεται η ομοιότητα μέσω του cosine similarity για κάθε query με κάθε doc και εκτυπώνεται σε μια λίστα το ποσοστό % της ομοιότητας. Αυτό γίνεται για όλα τα ερωτήματα και τα αρχεία, όμως εμφανίζονται οι κορυφαίες k τιμές ομοιότητες της λίστες με το αντίστοιχο doc_id, όπου το k είναι ένας ακέραιος που το πρόγραμμα ζητά από τον χρήστη να εισάγει.

ColBERT model

Αρχικά η υλοποίηση του μοντέλου ColBERT πραγματοποιήθηκε στο περιβάλλον Google colab καθώς είναι ένα περιβάλλον που προσφέρει δυνατότητες, βιβλιοθήκες που δεν βρίσκονται σε υλοποιήσεις της Python για Windows λειτουργικά παρά μόνο σε Linux/Unix λειτουργικά και είναι απαραίτητες για την λειτουργία του μοντέλου.
Σχετικά με την υλοποίηση χρησιμοποιήθηκε ως σημείο αναφοράς η υλοποίηση του Stanford και το παράδειγμα εκτέλεσης [5] του μοντέλου που βρίσκεται στο github της υλοποίησης [4] . Επιπλέον χρειάστηκε μετατροπή της υπάρχουσας συλλογής σε μορφή κατανοητή για το μοντέλο. Η αρχική μορφή της συλλογής ήταν με τα κείμενα σε .txt αρχεία με το όνομα του κάθε αρχείου να αποτελεί το document id καθενός ενώ εντός των αρχείων τα κείμενα είχαν τη μορφή μιας λέξης ανά γραμμή. Η μετατροπή που έγινε αφορούσε την δημιουργία ενός dataset το οποίο περιείχε δύο στήλες όπου στην μία αναγραφόταν το id κάθε κειμένου και στην άλλη το ίδιο το κείμενο με τη διαφορά πως το κείμενο πλέον βρισκόταν σε μια γραμμή και όχι σε πολλαπλές όπως φαίνεται και παρακάτω(**Εικόνα 10 και Εικόνα 11**).

Στην συνέχεια με παρόμοιο τρόπο δημιουργήθηκε ένα dataset για τα ερωτήματα μέσω του αρχείου Queries_20.txt. Μέσω των δύο datasets δημιουργήθηκαν δύο λίστες collection, queries που περιείχαν τα κείμενα και τα ερωτήματα αντίστοιχα. Η λίστα collection χρησιμοποιήθηκε για την δημιουργία του Index της συλλογής με αριθμό bits ίσο με 2 και με μέγιστο αριθμό token για κάθε κείμενο ίσο με 300. Τέλος βάσει του indexer δημιουργήθηκε και ένας searcher ο οποίος δοσμένου ενός ερωτήματος, ενός αριθμού k επιστρέφει μια λίστα με k αποτελέσματα με τα καλύτερα βάσει ποσοστού ομοιότητας κείμενα σχετικά με το ερώτημα. Παρακάτω φαίνεται ένα παράδειγμα εκτέλεσης του μοντέλου για k=5 και για όλα τα ερωτήματα όπου στην έξοδο εμφανίζεται σε φθίνουσα σειρά βάσει ομοιότητας το κάθε κείμενο, το id του, το ποσοστό ομοιότητας και η κατάταξη του στα αποτελέσματα(**Εικόνα 12**).

Μετρικές που υλοποιήθηκαν

Όσον αφορά την σύγκριση των μοντέλων χρησιμοποιήθηκαν οι μετρικές αξιολόγησης MAP, MRR και διαγράμματα ανάκλησης-ακρίβειας. Επίσης και στα δύο μοντέλα ο υπολογισμός των μετρικών έχει γίνει με ίδιο ακριβώς τρόπο και συνεπώς με σχεδόν τον ίδιο κώδικα με ελάχιστες διαφορές σχετικά με τα ορίσματα. Για τον υπολογισμό των μετρικών χρησιμοποιήθηκε το αρχείο Relevant_20.txt στο οποίο

βρίσκονται για κάθε ερώτημα τα id των σχετικών με το ερώτημα κειμένων. Έπειτα για τα αποτελέσματα που επιστρέφει το κάθε μοντέλο δοσμένου ενός ερωτήματος γίνεται σύγκριση κάθε id στα αποτελέσματα με τα id από το αρχείο relevant για το εκάστοτε ερώτημα. Εφόσον το id του αποτελέσματος είναι ίδιο με ένα id από το αρχείο υπολογιζόταν η μετρική precision για το συγκεκριμένο αποτέλεσμα, η μετρική RR εφόσον το αποτέλεσμα είναι το πρώτο σχετικό και το ποσοστό ανάκλησης εκείνη την στιγμή το οποίο στη συνέχεια χρησιμοποιείται για να υπολογιστεί η τιμή ακρίβειας σε κάθε επίπεδο ανάκλησης βάσει της συνάρτησης για παρεμβαλόμενη ακρίβεια που: **$P(r_j) = \max P(r)$** που αναγράφεται στο βιβλίο του **Baeza-Yates: Ανάκτηση πληροφορίας** στην σελίδα 140 [3]. Τέλος για κάθε ερώτημα υπολογίζεται η Μέση ακρίβεια και αφού ελεγχθούν όλα τα ερωτήματα υπολογίζεται το συνολικό MAP, MRR ενώ δημιουργείται και εκτυπώνεται το διάγραμμα ανάκλησης ακρίβειας. Ενδεικτικά η έξοδος του μοντέλου ColBERT φαίνεται στην **Εικόνα 13** ενώ η έξοδος για το Vector Space στην **Εικόνα 16**.

Αποτελέσματα - Παρατηρήσεις

Περιγραφή Συλλογής

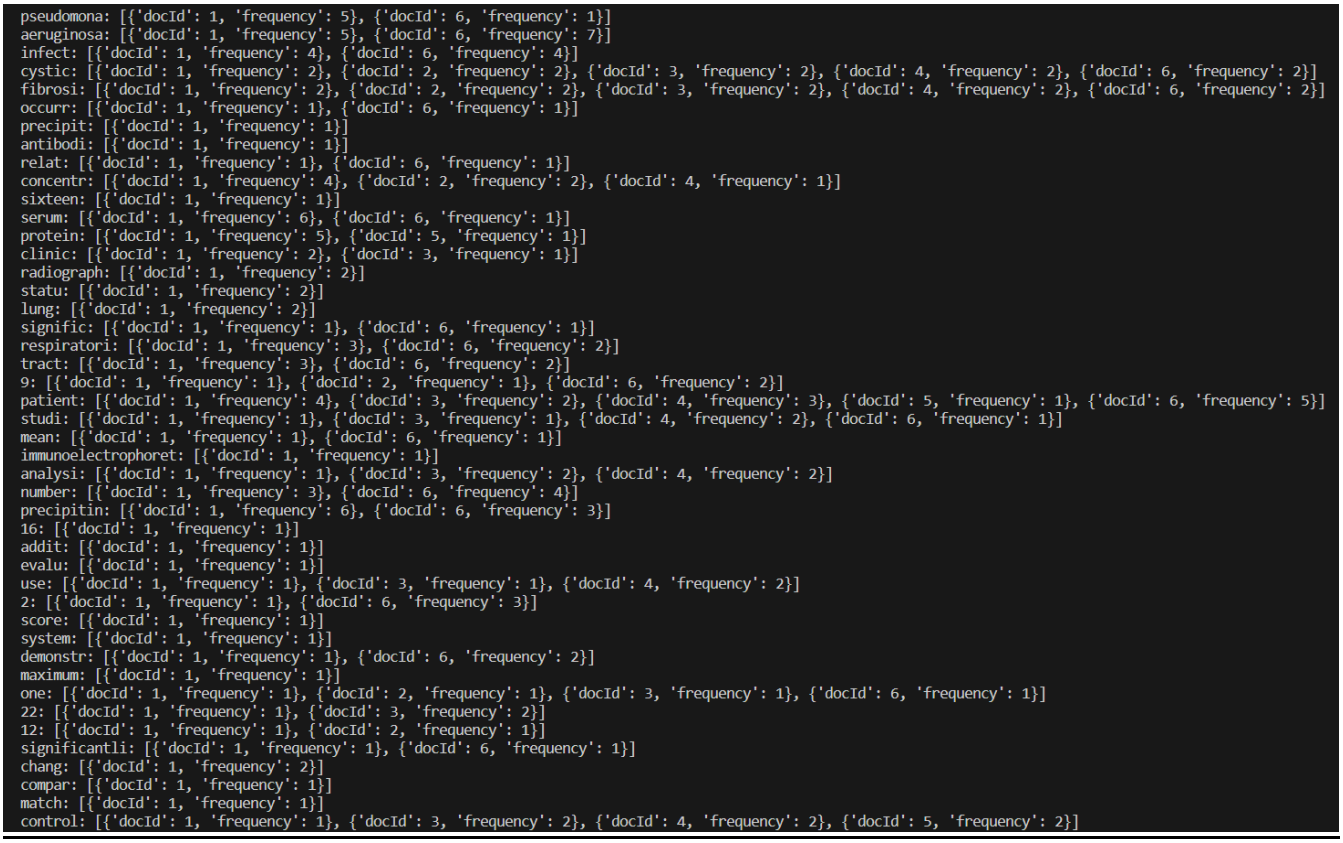
Η συλλογή είναι η Cystic Fibrosis η οποία περυλαμβάνει 1209 αρχεία/κείμενα και 20 ερωτήματα σε 3 αρχεία. Κάθε αρχείο κειμένου έχει ως όνομα το id του κειμένου και ως περιεχόμενο τον τίτλο και το κείμενο. Όσον αφορά για τα αρχεία ερωτημάτων το πρώτο αρχείο με τίτλο Queries_20.txt περιλαμβάνει τα ερωτήματα με κάθε ένα να βρίσκεται σε διαφορετική γραμμή εντός του αρχείου. Το δεύτερο αρχείο με τίτλο Relevant_20.txt περιλαμβάνει τα σχετικά κείμενα για ένα ερώτημα με την αντιστοίχιση με το πρώτο αρχείο να γίνεται ανά γραμμή (δηλαδή το ερώτημα που βρίσκεται στη πρώτη γραμμή έχει ως σχετικά τα κείμενα που αναγράφονται στην πρώτη γραμμή κ.ο.κ). Το τρίτο αρχείο με τίτλο cfquery_detailed.txt περιέχει τα ερωτήματα, το id τους, τον αριθμό σχετικών κειμένων και τα σχετικά κείμενα αναλυτικά. Τα αρχεία κειμένων και το πρώτο αρχείο ερωτημάτων χρησιμοποιούνται για να δημιουργήσουν το ευρετήριο, τα μοντέλα και για εύρεση ομοιότητας μεταξύ ερωτημάτων-κειμένων μέσω του κάθε μοντέλου. Τέλος τα υπόλοιπα αρχεία ερωτημάτων χρησιμοποιούνται στον υπολογισμό των μετρικών.

Πειραματικά αποτελέσματα και σύγκριση μοντέλων

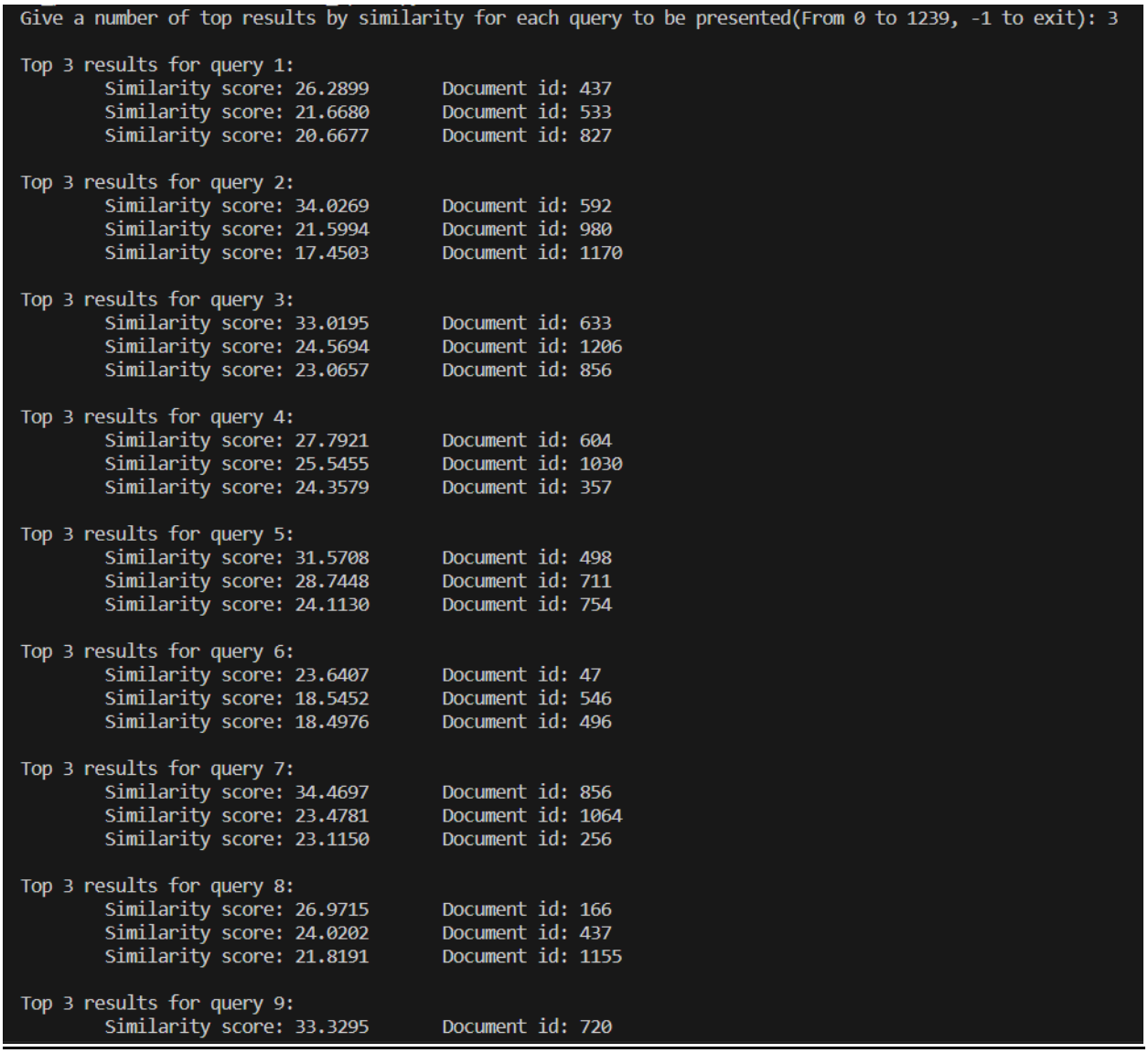
Αρχικά η σύγκριση των μοντέλων γίνεται βάσει των μετρικών MAP, MRR και των διαγραμμάτων ανάκλησης-ακρίβειας για κ=3,κ=5,κ=10 όπου κ ο αριθμός αποτελεσμάτων που επιστρέφει κάθε μοντέλο. Τα αποτελέσματα για τις μετρήσεις και τα διαγράμματα φαίνονται στον **Πίνακας 1**, στις **εικόνες Εικόνα 13Εικόνα 14Εικόνα 15**για το μοντέλο ColBERT και στις **εικόνες Εικόνα 16Εικόνα 17Εικόνα 18**για το μοντέλο Vector Space. Παρατηρώντας τα αποτελέσματα μπορούμε να δούμε πως οι τιμές των μετρικών είναι αρκετά κοντά και για τα δύο μοντέλα με κάθε μια μετρική να έχει μια απόκλιση περίπου 3-5% από μοντέλο σε μοντέλο. Επιπλέον φαίνεται πως η μεγαλύτερη διαφορά των μοντέλων υπάρχει στις μικρότερες τιμές του κ ενώ για μεγαλύτερες τιμές συμπεριφέρονται με παρόμοιο τρόπο. Παρόλα αυτά το ColBERT μοντέλο εξακολουθεί να έχει καλύτερη απόδοση σε όλες τις μετρικές ανεξάρτητα από την τιμή του κ. Η καλύτερη απόδοση του ColBERT φαίνεται και από τα διαγράμματα ανάκλησης-ακρίβειας τα οποία ενώ είναι αρκετά παρόμοια φαίνεται πως για μικρές τιμές στο κ το Vector Space έχει αρκετά απότομη κλίση στο διάγραμμα σε αντίθεση με το ColBERT το οποίο φαίνεται να έχει πιο σταθερή μεταβολή στις τιμές του διαγράμματος. Επομένως από τις μετρικές

προκύπτει το συμπέρασμα πως το μοντέλο ColBERT είναι καλύτερο και πιο αποδοτικό από το Vector Space για τη συγκεκριμένη συλλογή κειμένων και ερωτημάτων.
Επίσης πραγματοποιήθηκε μια σύγκριση αναφορικά με τους χρόνους εκτέλεσης για την δημιουργία ευρετήριού και της παραγωγής αποτελεσμάτων για το κάθε μοντέλο. Η σύγκριση αυτή ωστόσο δεν είναι δυνατό να δώσει πλήρη έγκυρα αποτελέσματα καθώς τα μοντέλα δεν εκτελούνται χρησιμοποιώντας τους ίδιους πόρους αλλά μπορεί να παρέχει μια γενική εικόνα για τα μοντέλα. Τα αποτελέσματα φαίνονται στον **Πίνακας 2**, στην **εικόνα 9** για την παραγωγή αποτελεσμάτων του ColBERT, στην **Εικόνα 19** για τους χρόνους indexer,searcher του μοντέλου ColBERT (Οι τιμές για το ColBERT προέρχονται από την μέτρηση εκτέλεσης του Google colab) και στην **Εικόνα 20** για το μοντέλο Vector Space. Παρατηρώντας τα αποτελέσματα βλέπουμε πως το μοντέλο ColBERT χρειάζεται περισσότερο χρόνο για την προετοιμασία του μοντέλου (4 λεπτά για τον Indexer, 5 δευτερόλεπτα για τον searcher) σε αντίθεση με το Vector Space που χρειάζεται μόλις 4.1 δευτερόλεπτα. Ωστόσο αυτός ο χρόνος που το ColBERT αναθέτει στην προετοιμασία της συλλογής δίνει το πλεονέκτημα στην παραγωγή αποτελεσμάτων καθώς χρειάζεται μόλις 1 δευτερόλεπτο ενώ το Vector Space χρειάζεται σχεδόν 2 λεπτά και 10 δευτερόλεπτα. Τέλος το συμπέρασμα που προκύπτει είναι πως το ColBERT κάνει μια πιο πολύπλοκη αλλά και πιο αποδοτική επεξεργασία του μοντέλου η οποία ειδικά για μεγαύτερες συλλογές είναι εξαιρετικά χρήσιμη, καθώς σε μεγαλύτερες συλλογές το Vector Space μοντέλο θα χρειαστεί αρκετό παραπάνω χρόνο.

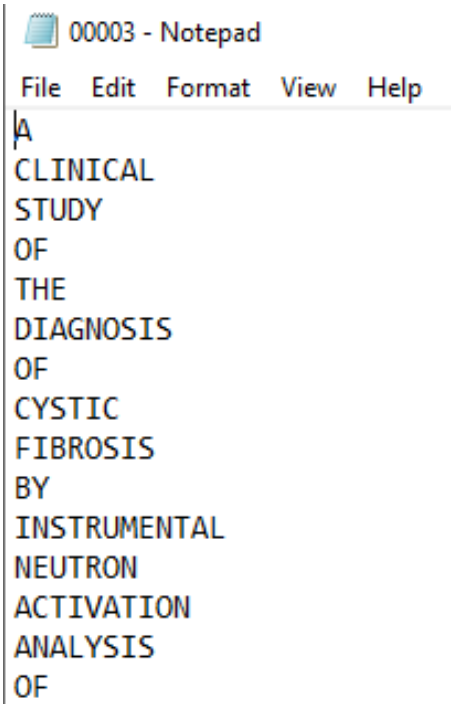
Εικόνες



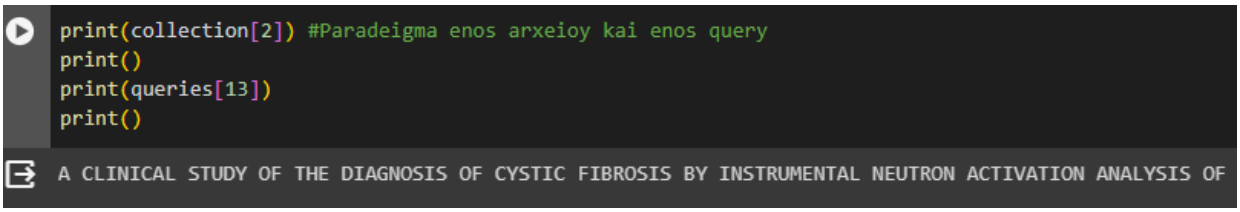
***Εικόνα 8**:Μέρος του ανεστραμμένου αρχείου, όπου εκτυπώνεται κάθε λέξη που έχει "καθαριστεί" ανά γραμμή, μαζί με το αρχείο στο οποίο έχει βρεθεί η λέξη μαζί με την συχνότητα εμφάνισης. Πρόκειται για την αρχή της εκτύπωσης, οπότε επειδή υπολογίζεται πρώτα για το do*



Εικόνα 9*:Μέρος του αποτελέσματος υπολογισμού των k-κορυφαίων ποσοστών ομοιότητας % για κάθε ερώτημα και αρχείο βάσει του Vector Space μοντέλου, όπου k ακέραιος που εισάγει ο χρήστης που αναπαριστά τον αριθμό των score από την κορυφή της λίστας με τα ποσοστά ομοι*



***Εικόνα 10**: Παράδειγμα από την αρχική μορφή του κειμένου με id = 3 στο αρχείο .txt*

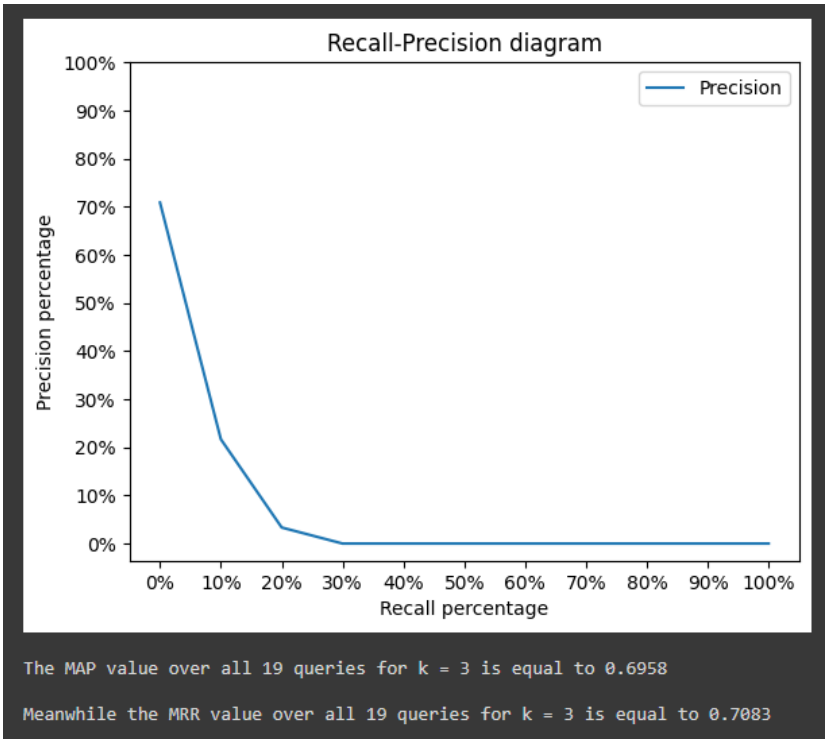


***Εικόνα 11**: Παράδειγμα από την τελική μορφή του κειμένου με id = 3 μέσω του google colab*

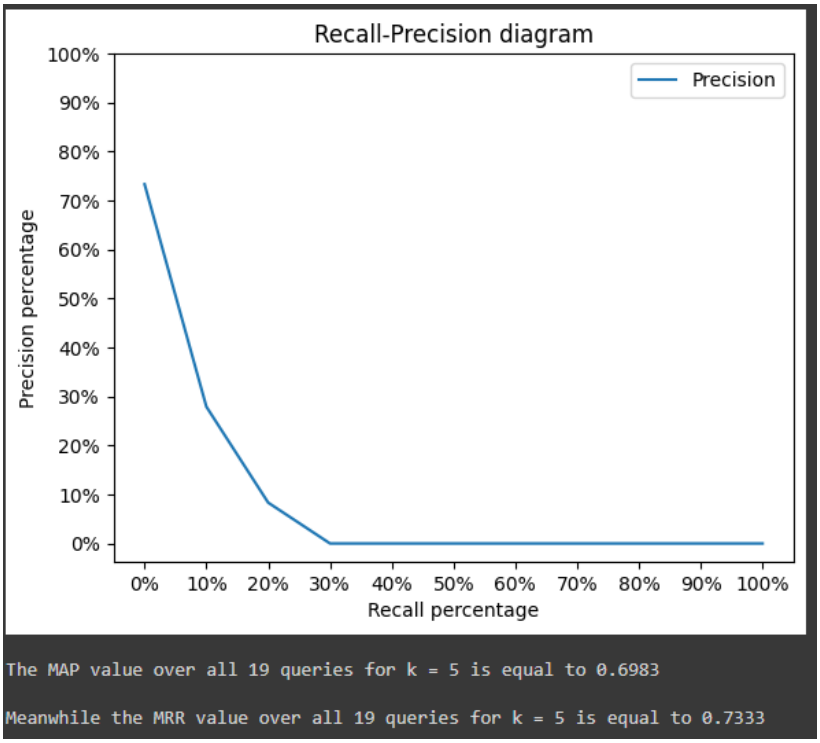
```
for query_id in range(19): #Gia ola ta queries entos toy range
  query = queries[query_id]
  print(f"#> {query}") #Emfanise to query

  results = searcher.search(query, k=5) #Vres ta k docs ws results gia to query mesw toy searcher sth syllogh

  for passage_id, passage_rank, passage_score in zip(*results): #Gia kathe doc sto result emfanise to
      id = doc_df.loc[passage_id]['doc_id']
      print(f"\t [{passage_rank}] \t\t {id} \t\t{passage_score:.1f} \t\t {searcher.collection[passage_id]}")
```

```
#> What are the effects of calcium on the physical properties of mucus from CF patients

        [1]             533             23.0            EFFECTS OF CALCIUM ON INTESTINAL MUCIN IMPLICATIONS FOR CYSTIC FIBROSIS A MAJOR FEATURE OF THE DISEASE CYSTIC FIBROS
        [2]             441             20.4            WATER AND ELECTROLYTES OF EXOCRINE SECRETIONS PP 17991 IT HAS BEEN RECOGNIZED THAT THE LEVELS OF WATER AND ELECTROLY
        [3]             957             19.4            THE BIOLOGIC ACTIVITIES OF CYSTIC FIBROSIS SERUM II ULTRASTRUCTURAL ASPECTS OF THE EFFECT OF CYSTIC FIBROSIS SERA AN
        [4]             484             19.2            CALCIUM FLUX AND CYSTIC FIBROSIS LETTER IN AN EFFORT TO DETERMINE WHETHER INCREASED PERMEABILITY TO CALCIUM IONS COU
        [5]             741             18.6            THE BIOLOGIC ACTIVITIES OF CYSTIC FIBROSIS SERUM I THE EFFECTS OF CYSTIC FIBROSIS SERA AND CALCIUM IONOPHORE A 23187
#> Can one distinguish between the effects of mucus hypersecretion and infection on the submucosal glands of the respiratory tract in CF

        [1]             437             17.7            PULMONARY ASPECTS OF CYSTIC FIBROSIS PP 324 DEFECTIVE MUCOCILIARY TRANSPORT HAS BEEN IMPLICATED IN THE PATHOGENESIS
        [2]             754             17.5            EFFECTS OF GRAVITY ON TRACHEAL MUCUS TRANSPORT RATES IN NORMAL SUBJECTS AND IN PATIENTS WITH CYSTIC FIBROSIS A NONIN
        [3]             589             17.0            PSEUDOMONAS AERUGINOSA INFECTION IN CYSTIC FIBROSIS BACTERICIDAL EFFECT OF SERUM FROM NORMAL INDIVIDUALS AND PATIENT
        [4]             592             17.0            TREATMENT OF MUCUS HYPERSECRETION IN HUMAN DISEASE TREATMENT OF MUCUS HYPERSECRETION IN HUMAN DISEASE SHOULD FIRST E
        [5]             151             16.7            THE ELECTRON MICROSCOPIC APPEARANCE OF PRESECRETED GASTRIC MUCUS IN CYSTIC FIBROSIS GASTRIC MUCOSAL BIOPSIES FROM EI
#> How are salivary glycoproteins from CF patients different from those of normal subjects

        [1]             633             23.7            A STUDY OF THE SALIVARY GLYCOPROTEIN IN CYSTIC FIBROSIS PATIENTS AND CONTROLS FUCOSE INCORPORATION AND PROTEIN PATTE
        [2]             1175            19.5            SULPHATED GLYCOPROTEINS IN THE PANCREAS SEVERAL REPORTS HAVE INDICATED THE PRESENCE OF ABNORMALLY HIGH AMOUNTS OF SU
        [3]             1206            19.4            SERUM SALIVARY AMYLASE IN CYSTIC FIBROSIS LETTER WE FOUND THAT CYSTIC FIBROSIS CF PATIENTS WITH PANCREATIC INSUFFICI
        [4]             55              18.7            SOME ASPECTS OF IMMUNITY IN PATIENTS WITH CYSTIC FIBROSIS VARIOUS ASPECTS OF THE IMMUNE STATUS WERE EXAMINED IN PATI
        [5]             139             18.5            PURIFICATION AND PROPERTIES OF THE CALCIUMPRECIPITABLE PROTEIN IN SUBMAXILLARY SALIVA OF NORMAL AND CYSTIC FIBROSIS
#> What is the lipid composition of CF respiratory secretions

        [1]             604             21.2            LIPID COMPOSITION OF AIRWAY SECRETIONS FROM PATIENTS WITH ASTHMA AND PATIENTS WITH CYSTIC FIBROSIS LIPIDS FROM THE F
        [2]             1039            19.4            THE ROLE OF NUTRITIONAL STATUS AIRWAY OBSTRUCTION HYPOXIA AND ABNORMALITIES IN SERUM LIPID COMPOSITION IN LIMITING E
        [3]             711             19.1            FATTYACID COMPOSITION OF LECITHIN FRACTION OF MUCUS IN CYSTIC FIBROSIS LETTER WE HAVE STUDIED SEVEN PATIENTS WITH CF
        [4]             374             18.7            HUMAN RESPIRATORY TRACT SECRETION MUCOUS GLYCOPROTEINS OF NONPURULENT TRACHEOBRONCHIAL SECRETIONS AND SPUTUM OF PATI
        [5]             23              18.4            HUMAN RESPIRATORY TRACT SECRETIONS 1 MUCOUS GLYCOPROTEINS SECRETED BY CULTURED NASAL POLYP EPITHELIUM FROM SUBJECTS
#> Is CF mucus abnormal

        [1]             776             25.4            CYSTIC FIBROSIS CYSTIC FIBROSIS CF IS A SERIOUS GENETIC DISORDER OCCURRING MAINLY AMONG THE WHITES AND STARTING IN E
        [2]             501             25.1            IMPAIRMENT OF MUCOCILIARY TRANSPORT IN CYSTIC FIBROSIS OVER THE PAST FEW YEARS STUDIES FROM VARIOUS LABORATORIES HAV
        [3]             499             25.0            PATHOPHYSIOLOGY OF MUCUS SECRETION IN CYSTIC FIBROSIS ABNORMALITY OF MUCUS SECRETION EITHER BIOCHEMICAL OR PHYSICAL
```

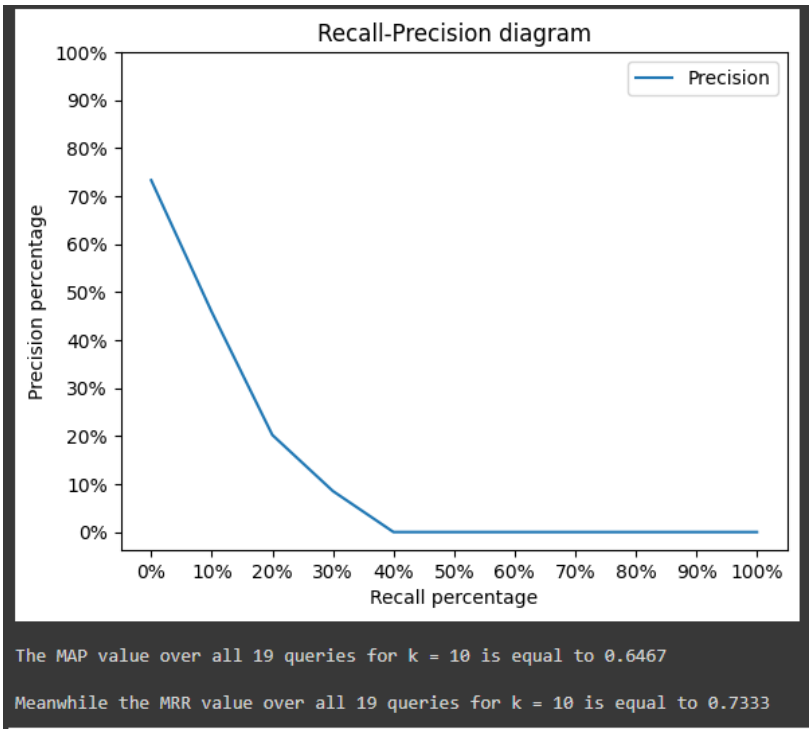
Εικόνα 12: *Αποτελέσματα εκτέλεσης του μοντέλου ColBERT με εκτύπωση των 5 αποτελεσμάτων με φθίνουσα σειρά αξιολόγησης*



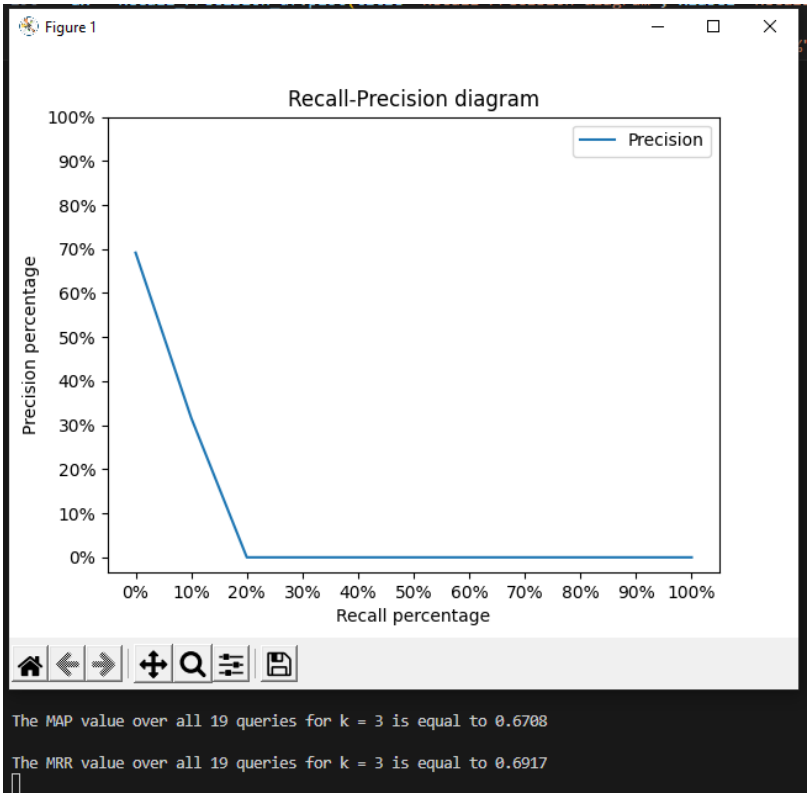
Εικόνα 13*: Αποτελέσματα μετρικών για το μοντέλο ColBERT με κ = 3*



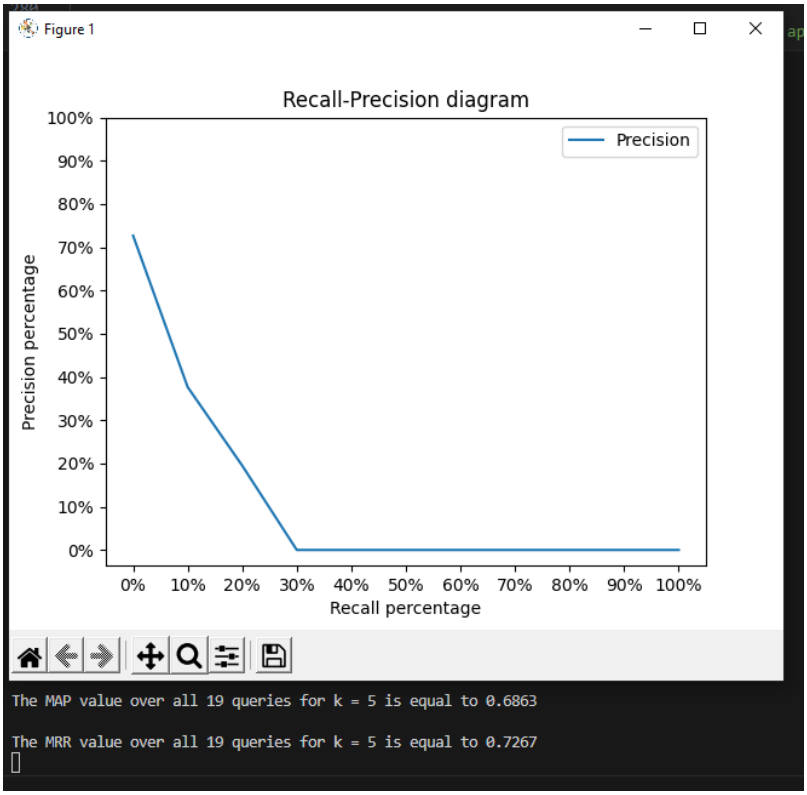
***Εικόνα 14**: Αποτελέσματα μετρικών για το μοντέλο ColBERT με κ = 5*



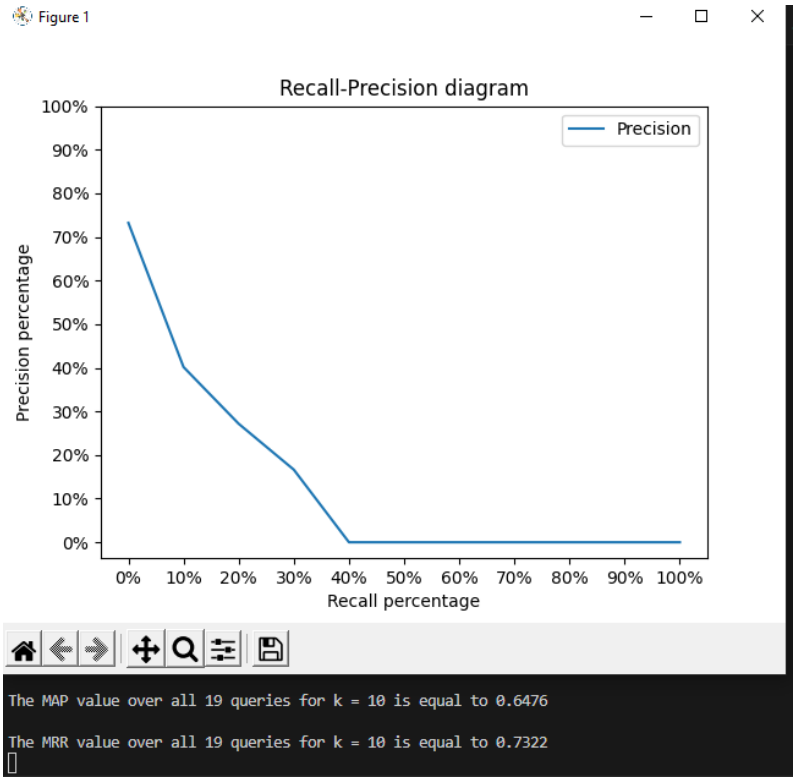
***Εικόνα 15**: Αποτελέσματα μετρικών για το μοντέλο ColBERT με κ = 10*



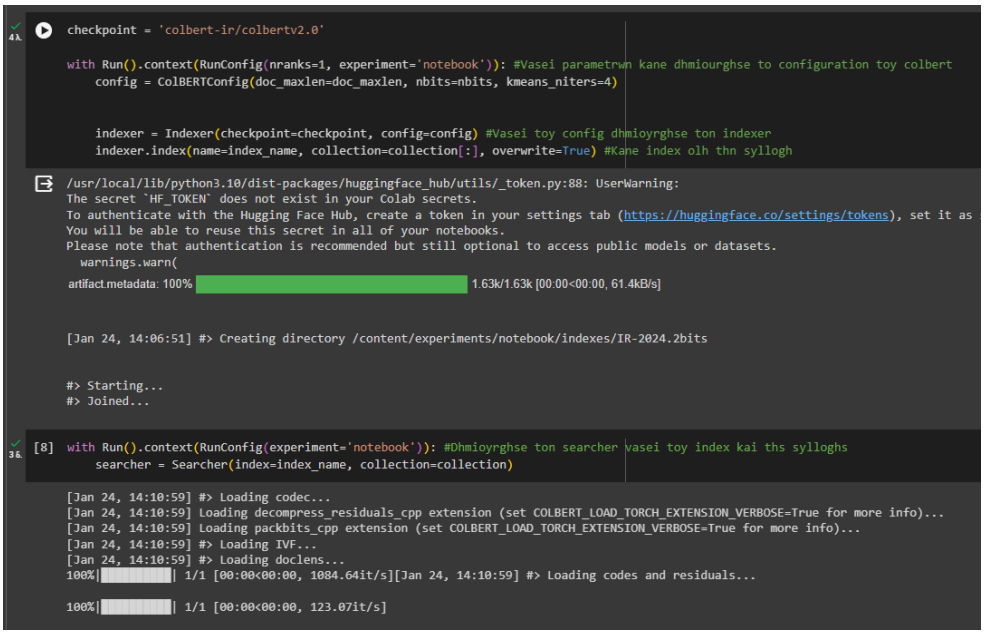
***Εικόνα 16**: Αποτελέσματα μετρικών για το μοντέλο Vector Space με κ = 3*



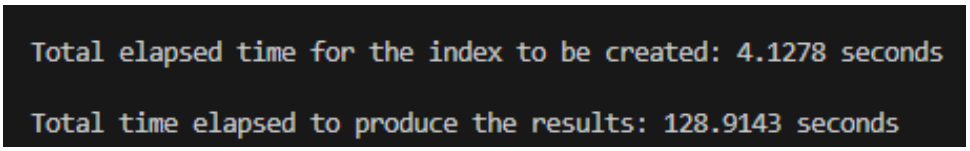
***Εικόνα 17**: Αποτελέσματα μετρικών για το μοντέλο Vector Space με κ = 5*



***Εικόνα 18**: Αποτελέσματα μετρικών για το μοντέλο Vector Space με κ = 10*



***Εικόνα 19**: Χρόνοι εκτέλεσης για indexer,searcher ColBERT μοντέλου.*



***Εικόνα 20**: Χρόνοι εκτέλεσης για το Vector Space μοντέλο*

Πίνακες

ΜΟΝΤΕΛΟ	ΤΙΜΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (ΑΡΙΘΜΟΣ κ)	ΤΙΜΗ ΜΕΤΡΙΚΩΝ (MAP, MRR)
ColBERT	3	MAP=0.6958, MRR=0.7083
ColBERT	5	MAP=0.6983, MRR=0.7333
ColBERT	10	MAP=0.6467, MRR=0.7333
Vector Space	3	MAP=0.6708, MRR=0.6917
Vector Space	5	MAP=0.6863, MRR=0.7267
Vector Space	10	MAP=0.6476, MRR=0.7322

***Πίνακας 1**: Τιμές μετρικών MAP,MRR για τα δύο μοντέλα*

ΜΟΝΤΕΛΟ	ΤΥΠΟΣ ΕΚΤΕΛΕΣΗΣ	ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ
ColBERT	INDEXER	4 ΛΕΠΤΑ
ColBERT	SEARCHER	5 ΔΕΥΤΕΡΟΛΕΠΤΑ
ColBERT	ΠΑΡΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ	1 ΔΕΥΤΕΡΟΛΕΠΤΟ
Vector Space	ΔΗΜΙΟΥΡΓΙΑ ΕΥΡΕΤΗΡΙΟΥ	4.1 ΔΕΥΤΕΡΟΛΕΠΤΑ
Vector Space	ΠΑΡΑΓΩΓΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ	2 ΛΕΠΤΑ ΚΑΙ 10 ΔΕΥΤΕΡΟΛΕΠΤΑ

***Πίνακας 2**: Χρόνοι εκτέλεσης δύο μοντέλων*

Αναφορές

[1]. Salton, G., Buckley, C., 1988. Term-weighting Approaches in Automatic Text Retrieval. Inf Process Manage 24, 513–523. https://doi.org/10.1016/0306-4573(88)90021-0

[2]. Salton, G., Wong, A., Yang, C.S., 1975. A Vector Space Model for Automatic Indexing. Commun ACM 18, 613–620. https://doi.org/10.1145/361219.361220

[3]. Ricardo Baeza - Yates, Berthier Ribeiro – Neto, Ανάκτηση πληροφορίας 2η έκδοση, 140-141.

[4]. https://github.com/stanford-futuredata/ColBERT

[5]. https://colab.research.google.com/github/stanfordfuturedata/ColBERT/blob/main/docs/intro2new.ipynb#scrollTo=CQFUHYTZs0aa

[6]. https://www.datacamp.com/tutorial/stemming-lemmatization-python

Παράρτημα

ΑΡΧΕΙΑ ΑΝΑ ΕΡΩΤΗΜΑ

ΕΡΩΤΗΜΑ 1: Indexing.py
ΕΡΩΤΗΜΑ 2: Vector_space.py
ΕΡΩΤΗΜΑ 3: Colbert.py
ΕΡΩΤΗΜΑ 4: Vector_space_Metrics.py, Colbert_Metrics.py

Κώδικας ερωτήματος 1

```
import collections
import numpy as np
import math
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

N = 7
#nltk.download('stopwords')
#nltk.download('wordnet')


class Appearance:                       #Domh dedomenwn gia tis emfaniseis se ena document
    def __init__(self, docId, frequency):
        self.docId = docId
        self.frequency = frequency

    def __repr__(self):
        return str(self.__dict__)

class InvertedIndex:                    #Klash gia to eurethrio
    def __init__(self):
        self.index = dict()
        self.stemmer = PorterStemmer()
        self.lemmatizer = WordNetLemmatizer()

    def __repr__(self):
        return str(self.index)

    def index_document(self,document, doc_id): #Sinartisi me thn opoia dhmiourgeite to eurethrio
            word = document.read()
            clean = self.clean_text(word)      #Katharizei tis lexeis diathrwntas tis simantikes
```

```
        terms = clean.split()
        counter = collections.Counter(terms) #Counter poy ypologizei ta frequency gia kathe lexh entos listas

        for word in terms:              #Gia kathe lexh sth lista an yparxei sto eyrethrio hdh elegxe to doc_id
alliws prosthese thn
            if word in self.index:
                ex = self.index[word]     #Ex exei th lista apo appearance ths kathe lexhs
                inside = 1
                for item in ex:          #Gia kathe stoixeio ths listas an yparxei eisagwgh ths lexhs me ayto to docId
sto eyrethrio kane inside = 0
                    if item.docId == doc_id:
                        inside = 0
                if inside:              #An den yparxei eisagwgh ths lexhs me ayto to doc_id prosthese thn
                    freq = counter[word]
                    app = Appearance(doc_id, freq)
                    self.index[word].append(app)
            else:
                freq = counter[word]
                app = Appearance(doc_id, freq)
                apps = []
                apps.append(app)
                self.index[word] = apps

    def clean_text(self, text):             #Sinartisi poy metatrepei tis lexeis se eniko me mikra grammata kai
afairei ta stopwords apo to keimeno
        stop_words = set(stopwords.words('english'))
        clean_text = re.sub(r'[^\w\s]', '', text)
        terms = clean_text.split()
        filtered = [self.lemmatizer.lemmatize(self.stemmer.stem(term.lower())) for term in terms if term.lower()
not in stop_words]
        return ' '.join(filtered)

index = InvertedIndex()

for doc_id in range(1, N):                  #Gia ola ta arxeia me id sto range twn dyo arithmwn (An range(1,3) tote
arxeia me id 1 kai 2)
    file_path = 'docs/'+f'{doc_id:05}'+'.txt' #Metetrepse to id se filepath (An id = 1 tote filepath 00001.txt)
    f = open(file_path, 'r')                #Anoixe to arxeio

    index.index_document(f, doc_id)

    f.close()                           #Kleise to arxeio
```

Κώδικας ερωτήματος 2

```
import collections
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

N = 1240
# nltk.download('stopwords')
# nltk.download('wordnet')

#Domh dedomenwn gia tis emfaniseis se ena document
class Appearance:
    def __init__(self, docId, frequency):
        self.docId = docId
        self.frequency = frequency

    def __repr__(self):
        return f"[docId: {self.docId}, frequency: {self.frequency}]"

#Klash gia to eurethrio
class InvertedIndex:
    def __init__(self):
        self.index = dict()
        self.stemmer = PorterStemmer()
        self.lemmatizer = WordNetLemmatizer()

    def __repr__(self):
        result = ""
        for word, appearances in self.index.items():
            result += f"{word}: "
            result += f"{appearances[0]}"
            for appearance in appearances[1:]:
                result += f", {appearance}"
            result += "\n"
        return result

    #Sinartisi me thn opoia dhmiourgeite to eurethrio
    def index_document(self, document, doc_id):
```

```
        clean = self.clean_text(document) #Katharizei tis lexeis diathrwntas tis simantikes
        terms = clean.split()
        counter = collections.Counter(terms) #Counter poy ypologizei ta frequency gia kathe lexh entos listas

        for word in terms: #Gia kathe lexh sth lista an yparxei sto eyrethrio hdh elegxe to doc_id alliws prosthese thn
            if word in self.index:
                ex = self.index[word] #Ex exei th lista apo appearance ths kathe lexhs
                inside = 1
                for item in ex: #Gia kathe stoixeio ths listas an yparxei eisagwgh ths lexhs me ayto to docId sto eyrethrio kane inside = 0
                    if item.docId == doc_id:
                        inside = 0
                if inside: #An den yparxei eisagwgh ths lexhs me ayto to doc_id prosthese thn
                    freq = counter[word]
                    app = Appearance(doc_id, freq)
                    self.index[word].append(app)
            else:
                freq = counter[word]
                app = Appearance(doc_id, freq)
                apps = []
                apps.append(app)
                self.index[word] = apps

    #Sinartisi poy metatrepei tis lexeis se eniko me mikra grammata kai afairei ta stopwords apo to keimeno
    def clean_text(self, text):
        stop_words = set(stopwords.words('english'))
        clean_text = re.sub(r'[^\w\s]', '', text)
        terms = clean_text.split()
        filtered = [self.lemmatizer.lemmatize(self.stemmer.stem(term.lower())) for term in terms if term.lower() not in stop_words]
        return ' '.join(filtered)

    #Sinartisi poy dhmiourgei to vector space montelo
    def vector_space(self, N):
        rows = len(self.index) + 1
        columns = N
        w_matrix = [[0 for i in range(columns)] for j in range(rows)] #Pinakas ston opoion tha vriskontai ta varh kathe lexhs ana keimeno
        w_matrix[0][0] = "Word\Doc id"

        for i in range(1, N): #Sthn grammh 0 toy pinaka yparxei index gia ta doc id
            w_matrix[0][i] = i

        i = 1
```

```
        for word in self.index: #Sthn sthlh 0 toy pinaka anagrafontai oi lexeis
            w_matrix[i][0] = word
            i += 1

        for i in range(1, rows): #Se kathe allo stoixeio toy pinaka ypologizontai ta katallhla tf-idf gia thn kathe lexh sto ekastote keimeno (Gia thn lexh sth thesh [1][0] h thesh [1][1] apothikeyei to varos ths sto doc 1)
            idf_list = self.index[w_matrix[i][0]]
            ni = len(idf_list)
            idf = math.log2(1 + ((N - 1) / ni))
            for item in idf_list:
                f_ij = item.frequency
                j = item.docId
                tf = 1 + math.log2(f_ij)
                result = round(tf * idf, 3)
                w_matrix[i][j] = result
        return w_matrix

    #Sinartisi poy epistrefei ta varh twn keimenwn synolika ws dianismata
    def get_vectors(self, matrix, N):
        d = []
        d.append("Vectors:")
        for j in range(1, N): #Dhmiourgeitai lista sthn opoia apothikeyontai ta varh enos keimenoy kai meta afth h lista metatrpetai se dianisma
            lst = []
            for i in range(1, len(matrix)):
                lst.append(matrix[i][j])
            vctr = np.array(lst)
            d.append(vctr)
        return d

    def cosine_similarity(self, a, b): #Sinartisi poy ypologizei omoiothta metaxi dyo dianismatwn gia thn eyresh ths omoiothtas gia dianysmata enos doc kai enos query
        dot_product = sum(a * b for a, b in zip(a, b))
        norm_a = sum(a * a for a in a) ** 0.5
        norm_b = sum(b * b for b in b) ** 0.5
        if norm_a == 0 or norm_b == 0:
            return 0
        return dot_product / (norm_a * norm_b)

    def vector_for_query(self, query): #Sinartisi poy dhmioyrgei to dianysma gia ena query ypologizontas opws kai sta docs ta tf-idf varh toy kathe oroy
        clean_query = self.clean_text(query)
        terms = clean_query.split()
        counter = collections.Counter(terms)
        query_vector = np.zeros(len(self.index) + 1)
```

```
        for i, word in enumerate(self.index.keys(), start=0): #keys() dinei ta words apo to query
            if word in terms:
                idf_list = self.index[word]
                ni = len(idf_list)
                idf = math.log2(1 + ((N - 1) / ni))
                f_ij = counter[word]
                tf = 1 + math.log2(f_ij)
                query_vector[i] = round(tf * idf, 3)

        return query_vector

#Anoigma arxeioy queries gia fortwsh twn erwthmatwn
file_name = 'Queries_20.txt'
queries = []
with open(file_name, 'r') as file:
    lines = file.readlines()

index = InvertedIndex()

#Index query kai apothikeysh se ena arxeio me thn morfh poy exoyn ta docs apo to opoio dhmiourgeite to
katharo query gia ta vectors
for line in lines:
    words = line.split()
    query_file_name = 'query.txt'
    with open(query_file_name, 'w') as query_file:
        for word in words:
            query_file.write(word + "\n")

    with open(query_file_name, 'r') as query_file:
        cleaned_query = index.clean_text(query_file.read())

    queries.append(cleaned_query)

#Dhmioyrgia eyrethrioy gia kathe doc
for doc_id in range(1, N):
    file_path = 'docs/' + f'{doc_id:05}' + '.txt'

    exists = os.path.exists(file_path)
    if exists:
        with open(file_path, 'r') as f:
            index.index_document(f.read(), doc_id)

#Dhmioyrgia vectors gia kathe doc
matrix = index.vector_space(N)
```

```
vector_list = index.get_vectors(matrix, N)

#Dhmioyrgia twn vectors gia kathe query
query_vectors = []
for query in queries:
    vector = index.vector_for_query(query)
    query_vectors.append(vector)

#Eisodos toy xrhsth gia posa apotelesmata thelei sto telos sxetika me thn omoiothta kai tis metrikes
k = -1
while k < 0 or k > 1210:
    k = int(input(f"Give a number of top results by similarity for each query to be presented(From 0 to {N-1}, -1 to exit): "))
    if k == -1:
        print("Goodbye!")
        exit()

#Ypologise to similarity gia kathe query me kathe doc kai emfanise ta kalytera k apotelesmata
for query_id,item in enumerate(query_vectors):
    sim = []
    for doc_id in range(1, N):
        document_vector = vector_list[doc_id]
        similarity = index.cosine_similarity(item, document_vector)*100 #Ypologismos similarity apo thn sinartisi cosine similarity kai metatroph se pososto
        sim.append([similarity,doc_id]) #Lista poy periexei th timh omoiothtas enos query me ena doc kai to id toy doc aytoy

    sort_sim = sorted(sim, reverse=True) #Sort th lista apo to megalytero similarity sto mikrotero

    print(f"\nTop {k} results for query {query_id+1}:") #Emfanise ta k apotelesmata gia kathe query
    for i in range(k):
        print(f"\tSimilarity score: {sort_sim[i][0]:.4f} \tDocument id: {sort_sim[i][1]}")
```

Κώδικας ερωτήματος 3

```
# -*- coding: utf-8 -*-
"""Untitled0.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1HzNAL24E393zJnZdCqfZFGp44ZHH_9TP
"""

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

try: # When on google Colab, let's install all dependencies with pip.
    import google.colab
    !pip install -U pip
    !pip install -e ColBERT/['faiss-gpu','torch']
except Exception:
  import sys; sys.path.insert(0, 'ColBERT/')
  try:
    from colbert import Indexer, Searcher
  except Exception:
    print("If you're running outside Colab, please make sure you install ColBERT in conda following the instructions in our README. You can also install (as above) with pip but it may install slower or less stable faiss or torch dependencies. Conda is recommended.")
    assert False

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

import pandas as pd
import os

text_list = [] #Listes gia th dhmiourgia twn katalhllwn dataframes
doc_list = []
query_id = []
query_text = []

for doc_id in range(1,1240): #Gia kathe doc_id se ayto to range anoixe ta arxeia me tetoio doc id efoson yparxoyn
  file_path = f'{doc_id:05}'+'.txt'
```

```
  exists = os.path.exists(file_path) #True an to arxeio yparxei
  if exists:
    f = open(file_path,'r')

    text = f.read() #Diavase to arxeio
    result = " ".join(line.strip() for line in text.splitlines()) #Enwse tis polles grammes toy arxeioy se mia
    doc_list.append(doc_id) #Prosthese to doc_id sth lista twn id
    text_list.append(result) #Prosthese to keimeno toy arxeioy sth lista arxeiwn
    f.close()

doc_data = { #Ftiaxe ena dict to opoio exei doc_id ta stoixeia ths listas id kai text ta stoixeia keimenoy gia
kathe doc_id
  "doc_id": doc_list,
  "text": text_list
}

doc_df = pd.DataFrame(doc_data) #Dhmioyrghse to dataframe

collection = [doc_df.loc[x]['text'] for x in range(len(doc_df))] #Ftiaxe to collection gia to Colbert mesw toy
dataframe

f = open('Queries_20.txt', 'r') #Idia diadikasia me th dhmiourgia toy collection gia ta queries

text = f.readlines()

count = 1
for line in text:
  query_id.append(count)
  count += 1
  query_text.append(line)

f.close()

q_data = {
  "query_id": query_id,
  "text": query_text
}

q_df = pd.DataFrame(q_data)

queries = [q_df.loc[x]['text'] for x in range(len(q_df))]

f'Loaded {len(queries)} queries and {len(collection):,} passages' #Emfanise posa queries kai keimena
fortothikan
```

```
print(collection[2]) #Paradeigma enos arxeioy kai enos query
print()
print(queries[13])
print()

nbits = 2   #Dhmiourgia index name gia ton index ths sylloghs, arithmo bit gia thn kwdikopoihsh kai maximum arithmos tokens gia kathe arxeio
doc_maxlen = 300

index_name = f'IR-2024.{nbits}bits'

checkpoint = 'colbert-ir/colbertv2.0'

with Run().context(RunConfig(nranks=1, experiment='notebook')): #Vasei parametrwn kane dhmiourghse to configuration toy colbert
    config = ColBERTConfig(doc_maxlen=doc_maxlen, nbits=nbits, kmeans_niters=4)


    indexer = Indexer(checkpoint=checkpoint, config=config) #Vasei toy config dhmioyrghse ton indexer
    indexer.index(name=index_name, collection=collection[:], overwrite=True) #Kane index olh thn syllogh

with Run().context(RunConfig(experiment='notebook')): #Dhmioyrghse ton searcher vasei toy index kai ths sylloghs
    searcher = Searcher(index=index_name, collection=collection)


for query_id in range(19): #Gia ola ta queries entos toy range
  query = queries[query_id]
  print(f"#> {query}") #Emfanise to query

  results = searcher.search(query, k=5) #Vres ta k docs ws results gia to query mesw toy searcher sth syllogh

  for passage_id, passage_rank, passage_score in zip(*results): #Gia kathe doc sto result emfanise to
      id = doc_df.loc[passage_id]['doc_id']
      print(f"\t [{passage_rank}] \t\t {id} \t\t{passage_score:.1f} \t\t {searcher.collection[passage_id]}")
```

Κώδικας ερωτήματος 4

COLBERT ΜΕΤΡΙΚΕΣ

```
# -*- coding: utf-8 -*-
"""Untitled0.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1HzNAL24E393zJnZdCqfZFGp44ZHH_9TP
"""

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

try: # When on google Colab, let's install all dependencies with pip.
    import google.colab
    !pip install -U pip
    !pip install -e ColBERT/['faiss-gpu','torch']
except Exception:
  import sys; sys.path.insert(0, 'ColBERT/')
  try:
    from colbert import Indexer, Searcher
  except Exception:
    print("If you're running outside Colab, please make sure you install ColBERT in conda following the instructions in our README. You can also install (as above) with pip but it may install slower or less stable faiss or torch dependencies. Conda is recommended.")
    assert False

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

import pandas as pd
import os
import matplotlib.pyplot as plt

text_list = [] #Listes gia th dhmiourgia twn katalhllwn dataframes
doc_list = []
query_id = []
query_text = []

for doc_id in range(1,1240): #Gia kathe doc_id se ayto to range anoixe ta arxeia me tetoio doc id efoson yparxoyn
```

```
  file_path = f'{doc_id:05}'+'.txt'

  exists = os.path.exists(file_path) #True an to arxeio yparxei
  if exists:
    f = open(file_path,'r')

    text = f.read() #Diavase to arxeio
    result = " ".join(line.strip() for line in text.splitlines()) #Enwse tis polles grammes toy arxeioy se mia
    doc_list.append(doc_id) #Prosthese to doc_id sth lista twn id
    text_list.append(result) #Prosthese to keimeno toy arxeioy sth lista arxeiwn
    f.close()

doc_data = { #Ftiaxe ena dict to opoio exei doc_id ta stoixeia ths listas id kai text ta stoixeia keimenoy gia
kathe doc_id
  "doc_id": doc_list,
  "text": text_list
}

doc_df = pd.DataFrame(doc_data) #Dhmioyrghse to dataframe

collection = [doc_df.loc[x]['text'] for x in range(len(doc_df))] #Ftiaxe to collection gia to Colbert mesw toy
dataframe

f = open('Queries_20.txt', 'r') #Idia diadikasia me th dhmiourgia toy collection gia ta queries

text = f.readlines()

count = 1
for line in text:
  query_id.append(count)
  count += 1
  query_text.append(line)

f.close()

q_data = {
  "query_id": query_id,
  "text": query_text
}

q_df = pd.DataFrame(q_data)

queries = [q_df.loc[x]['text'] for x in range(len(q_df))]
```

```
f'Loaded {len(queries)} queries and {len(collection):,} passages' #Emfanise posa queries kai keimena fortothikan

print(collection[2]) #Paradeigma enos arxeioy kai enos query
print()
print(queries[13])
print()

nbits = 2   #Dhmiourgia index name gia ton index ths sylloghs, arithmo bit gia thn kwdikopoihsh kai maximum arithmos tokens gia kathe arxeio
doc_maxlen = 300

index_name = f'IR-2024.{nbits}bits'

checkpoint = 'colbert-ir/colbertv2.0'

with Run().context(RunConfig(nranks=1, experiment='notebook')): #Vasei parametrwn kane dhmiourghse to configuration toy colbert
    config = ColBERTConfig(doc_maxlen=doc_maxlen, nbits=nbits, kmeans_niters=4)


    indexer = Indexer(checkpoint=checkpoint, config=config) #Vasei toy config dhmioyrghse ton indexer
    indexer.index(name=index_name, collection=collection[:], overwrite=True) #Kane index olh thn syllogh

with Run().context(RunConfig(experiment='notebook')): #Dhmioyrghse ton searcher vasei toy index kai ths sylloghs
    searcher = Searcher(index=index_name, collection=collection)

relevant_id = [] #Paromoia diadikasia gia to arxeio relevant opws me ta arxeia docs, queries gia th dhmioyrgia enos dataframe poy tha xrhsimeysei ston ypologismo metrikwn
relevant_text = []
f = open('Relevant_20.txt', 'r')

text = f.readlines()

count = 1
for line in text:
  relevant_id.append(count)
  count += 1
  relevant_text.append(line.split()) #Split gia na lavei kathe stoixeio(dhladh kathe id) poy yparxei se kathe grammh toy relevant ws stoixeia listas kai oxi ws ena synoliko string

f.close()

relevant_data = {
```

```
  "relevant_id": relevant_id,
  "text": relevant_text
}

relevant_df = pd.DataFrame(relevant_data)

Average_Precision = [] #Listes gia tis metrikes
Reciprocal_Rank = []
k = 5
data =  { #Ftiaxe ena dataframe gia to recall-precision diagramma opoy values oi times precision kai index oi
times sta pososta gia to recall gia ola ta queries synolika
    "Precision": [0,0,0,0,0,0,0,0,0,0,0]
  }
Recall_Precision_df = pd.DataFrame(data, index =
["0%","10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"])

num_queries = len(queries)
for query_id in range(num_queries): #Gia ola ta queries entos toy range
  rdoc_list = [] #Lista gia ta id tis apanthseis gia ena query
  query = queries[query_id]
  print(f"#> {query}") #Emfanise to query

  query_rp_df = pd.DataFrame(data, index =
["0%","10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"]) #Dataframe gia to ekastote
query ta apotelesmata toy opoioy tha prostethoyn sto geniko recall-precision dataframe
  num_rel = len(relevant_df.loc[query_id]['text']) #Orise th metavlhth num_rel ws to plithos twn sxetikwn
keimenwn gia to ekastote query

  results = searcher.search(query, k) #Vres ta k docs ws results gia to query mesw toy searcher sth syllogh

  for passage_id, passage_rank, passage_score in zip(*results): #Gia kathe doc sto result vale to id sth lista me
tis apanthseis kai emfanise to
      id = doc_df.loc[passage_id]['doc_id']
      rdoc_list.append(id)
      print(f"\t [{passage_rank}] \t\t {id} \t\t{passage_score:.1f} \t\t {searcher.collection[passage_id]}")

  count = 0 #Arithmos sxetikwn poy vriskontai stis apanthseis
  precision = [] #Lista gia to precision se kathe thesh poy yparxei sxetiko doc
  first = True #Metavlhth gia to prwto sxetiko doc gia ton ypologismo toy MRR
  for id in rdoc_list: #Gia kathe doc sth lista apanthsewn elegxe to me kathe doc sth lista sxetikwn
    for relevant in relevant_df.loc[query_id]['text']:
      if int(id) == int(relevant): #An to doc einai sxetiko me to query ayxhse to count
        count += 1
        position = rdoc_list.index(id) + 1 #Vres to position toy doc stis apanthseis (+1 kathos python xekina apo
to 0)
```

```
        p = count/position #Ypologise to precision sth thesh ayth vasei twn sxetikwn (p@k)
        precision.append(p) #Prosthese to sth lista precision

        if first: #An to first true tote to id einai to prwto sxetiko stis apanthseis ypologise to rr kai prosthese to sth lista enw kane to first false
            rr = 1/position
            Reciprocal_Rank.append(rr)
            first = False

        recall = (count/num_rel)*100 #Vres to pososto anaklhshs analoga me to posa sxetika exoyn vrethei kathe fora

        if (0 <= recall) & (float(query_rp_df.loc["0%"]) == 0): #Gia kathe pososto mikrotero toy recall poy den exei oristei hdh orise thn timh toy ws to precision epi %
            query_rp_df.loc["0%"] = p*100
        if (10 <= recall) & (float(query_rp_df.loc["10%"]) == 0):
            query_rp_df.loc["10%"] = p*100
        if (20 <= recall) & (float(query_rp_df.loc["20%"]) == 0):
            query_rp_df.loc["20%"] = p*100
        if (30 <= recall) & (float(query_rp_df.loc["30%"]) == 0):
            query_rp_df.loc["30%"] = p*100
        if (40 <= recall) & (float(query_rp_df.loc["40%"]) == 0):
            query_rp_df.loc["40%"] = p*100
        if (50 <= recall) & (float(query_rp_df.loc["50%"]) == 0):
            query_rp_df.loc["50%"] = p*100
        if (60 <= recall) & (float(query_rp_df.loc["60%"]) == 0):
            query_rp_df.loc["60%"] = p*100
        if (70 <= recall) & (float(query_rp_df.loc["70%"]) == 0):
            query_rp_df.loc["70%"] = p*100
        if (80 <= recall) & (float(query_rp_df.loc["80%"]) == 0):
            query_rp_df.loc["80%"] = p*100
        if (90 <= recall) & (float(query_rp_df.loc["90%"]) == 0):
            query_rp_df.loc["90%"] = p*100
        if (100 == recall) & (float(query_rp_df.loc["100%"]) == 0):
            query_rp_df.loc["100%"] = p*100

    Recall_Precision_df = Recall_Precision_df.add(query_rp_df) #Prosthese ta apotelesmata gia recall-precision toy query sto synoliko dataframe

    if first: #Se periptwsh poy stis apanthseis den yparxei sxetiko keimeno gia kapoio erwthma tote to RR einai 0
        Reciprocal_Rank.append(0)

    total = sum(precision) #Synolo twn epimeroys precision (An to precision einai keno tote total 0)
```

```
 if count != 0: #An yphrxan sxetika keimena stis apanthseis tote vres total einai to Average_Precision gia to query alliws einai 0
    total /= count
  Average_Precision.append(total)

map = sum(Average_Precision)/len(Average_Precision) #Ypologismos MAP kai MRR
mrr = sum(Reciprocal_Rank)/len(Reciprocal_Rank)

Recall_Precision_df = (Recall_Precision_df/num_queries).round(4) #Sto dataframe me to synoliko recall-precision gia ola ta queries diairese me ton arithmo twn queries gia na vreis tis meses times
ax = Recall_Precision_df.plot(title="Recall-Precision diagram", xlabel="Recall percentage", ylabel="Precision percentage", yticks = [0,10,20,30,40,50,60,70,80,90,100]) #Ftiakse to diagramma recall-precision me ta stoixeia toy dataframe
ax.set_xticks(range(len(Recall_Precision_df)))
ax.set_xticklabels(["0%","10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"])
ax.set_yticklabels(["0%","10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"])

print()
plt.show() #Emfanise to diagramma recall-precision

print(f"\nThe MAP value over all 19 queries for k = {k} is equal to {map:.4f}") #Emfanish apotelesmatwn metrikwn MAP kai MRR
print(f"\nThe MRR value over all 19 queries for k = {k} is equal to {mrr:.4f}")
```

VECTOR SPACE ΜΕΤΡΙΚΕΣ

```
import collections
import time
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

N = 1240
# nltk.download('stopwords')
# nltk.download('wordnet')

#Domh dedomenwn gia tis emfaniseis se ena document
class Appearance:
    def __init__(self, docId, frequency):
```

```
        self.docId = docId
        self.frequency = frequency

    def __repr__(self):
        return f"[docId: {self.docId}, frequency: {self.frequency}]"

#Klash gia to eurethrio
class InvertedIndex:
    def __init__(self):
        self.index = dict()
        self.stemmer = PorterStemmer()
        self.lemmatizer = WordNetLemmatizer()

    def __repr__(self):
        result = ""
        for word, appearances in self.index.items():
            result += f"{word}: "
            result += f"{appearances[0]}"
            for appearance in appearances[1:]:
                result += f", {appearance}"
            result += "\n"
        return result

    #Sinartisi me thn opoia dhmiourgeite to eurethrio
    def index_document(self, document, doc_id):
        clean = self.clean_text(document) #Katharizei tis lexeis diathrwntas tis simantikes
        terms = clean.split()
        counter = collections.Counter(terms) #Counter poy ypologizei ta frequency gia kathe lexh entos listas

        for word in terms: #Gia kathe lexh sth lista an yparxei sto eyrethrio hdh elegxe to doc_id alliws prosthese
thn
            if word in self.index:
                ex = self.index[word] #Ex exei th lista apo appearance ths kathe lexhs
                inside = 1
                for item in ex: #Gia kathe stoixeio ths listas an yparxei eisagwgh ths lexhs me ayto to docId sto
eyrethrio kane inside = 0
                    if item.docId == doc_id:
                        inside = 0
                if inside: #An den yparxei eisagwgh ths lexhs me ayto to doc_id prosthese thn
                    freq = counter[word]
                    app = Appearance(doc_id, freq)
                    self.index[word].append(app)
            else:
                freq = counter[word]
                app = Appearance(doc_id, freq)
```

```
            apps = []
            apps.append(app)
            self.index[word] = apps

    #Sinartisi poy metatrepei tis lexeis se eniko me mikra grammata kai afairei ta stopwords apo to keimeno
    def clean_text(self, text):
        stop_words = set(stopwords.words('english'))
        clean_text = re.sub(r'[^\w\s]', '', text)
        terms = clean_text.split()
        filtered = [self.lemmatizer.lemmatize(self.stemmer.stem(term.lower())) for term in terms if term.lower()
not in stop_words]
        return ' '.join(filtered)

    #Sinartisi poy dhmiourgei to vector space montelo
    def vector_space(self, N):
        rows = len(self.index) + 1
        columns = N
        w_matrix = [[0 for i in range(columns)] for j in range(rows)] #Pinakas ston opoion tha vriskontai ta varh
kathe lexhs ana keimeno
        w_matrix[0][0] = "Word\Doc id"

        for i in range(1, N): #Sthn grammh 0 toy pinaka yparxei index gia ta doc id
            w_matrix[0][i] = i

        i = 1
        for word in self.index: #Sthn sthlh 0 toy pinaka anagrafontai oi lexeis
            w_matrix[i][0] = word
            i += 1

        for i in range(1, rows): #Se kathe allo stoixeio toy pinaka ypologizontai ta katallhla tf-idf gia thn kathe
lexh sto ekastote keimeno (Gia thn lexh sth thesh [1][0] h thesh [1][1] apothikeyei to varos ths sto doc 1)
            idf_list = self.index[w_matrix[i][0]]
            ni = len(idf_list)
            idf = math.log2(1 + ((N - 1) / ni))
            for item in idf_list:
                f_ij = item.frequency
                j = item.docId
                tf = 1 + math.log2(f_ij)
                result = round(tf * idf, 3)
                w_matrix[i][j] = result
        return w_matrix

    #Sinartisi poy epistrefei ta varh twn keimenwn synolika ws dianismata
    def get_vectors(self, matrix, N):
        d = []
```

```
        d.append("Vectors:")
        for j in range(1, N): #Dhmiourgeitai lista sthn opoia apothikeyontai ta varh enos keimenoy kai meta afth h lista metatrpetai se dianisma
            lst = []
            for i in range(1, len(matrix)):
                lst.append(matrix[i][j])
            vctr = np.array(lst)
            d.append(vctr)
        return d

    def cosine_similarity(self, a, b): #Sinartisi poy ypologizei omoiothta metaxi dyo dianismatwn gia thn eyresh ths omoiothtas gia dianysmata enos doc kai enos query
        dot_product = sum(a * b for a, b in zip(a, b))
        norm_a = sum(a * a for a in a) ** 0.5
        norm_b = sum(b * b for b in b) ** 0.5
        if norm_a == 0 or norm_b == 0:
            return 0
        return dot_product / (norm_a * norm_b)

    def vector_for_query(self, query): #Sinartisi poy dhmioyrgei to dianysma gia ena query ypologizontas opws kai sta docs ta tf-idf varh toy kathe oroy
        clean_query = self.clean_text(query)
        terms = clean_query.split()
        counter = collections.Counter(terms)
        query_vector = np.zeros(len(self.index) + 1)

        for i, word in enumerate(self.index.keys(), start=0): #keys() dinei ta words apo to query
            if word in terms:
                idf_list = self.index[word]
                ni = len(idf_list)
                idf = math.log2(1 + ((N - 1) / ni))
                f_ij = counter[word]
                tf = 1 + math.log2(f_ij)
                query_vector[i] = round(tf * idf, 3)

        return query_vector

#Anoigma arxeioy queries gia fortwsh twn erwthmatwn
file_name = 'Queries_20.txt'
queries = []
with open(file_name, 'r') as file:
    lines = file.readlines()

index = InvertedIndex()
```

```
#Index query kai apothikeysh se ena arxeio me thn morfh poy exoyn ta docs apo to opoio dhmiourgeite to katharo query gia ta vectors
for line in lines:
    words = line.split()
    query_file_name = 'query.txt'
    with open(query_file_name, 'w') as query_file:
        for word in words:
            query_file.write(word + "\n")

    with open(query_file_name, 'r') as query_file:
        cleaned_query = index.clean_text(query_file.read())

    queries.append(cleaned_query)

start1 = time.time()
#Dhmioyrgia eyrethrioy gia kathe doc
for doc_id in range(1, N):
    file_path = 'docs/' + f'{doc_id:05}' + '.txt'

    exists = os.path.exists(file_path)
    if exists:
        with open(file_path, 'r') as f:
            index.index_document(f.read(), doc_id)
end1 = time.time()

#Dhmioyrgia vectors gia kathe doc
matrix = index.vector_space(N)
vector_list = index.get_vectors(matrix, N)

#Dhmioyrgia twn vectors gia kathe query
query_vectors = []
for query in queries:
    vector = index.vector_for_query(query)
    query_vectors.append(vector)

#Dhmioyrgia enos dataframe gia ton elegxo kai ypologismo twn metrikwn
relevant_id = []
relevant_text = []
f = open('Relevant_20.txt', 'r')

text = f.readlines()

count = 1
for line in text:
  relevant_id.append(count)
```

```
  count += 1
  relevant_text.append(line.split()) #Split gia na lavei kathe stoixeio(dhladh kathe id) poy yparxei se kathe grammh toy relevant ws stoixeia listas kai oxi ws ena synoliko string

f.close()

relevant_data = {
  "relevant_id": relevant_id,
  "text": relevant_text
}

relevant_df = pd.DataFrame(relevant_data)

#Listes kai metavlhtes gia ypologismo metrikwn
Average_Precision = []
Reciprocal_Rank = []
num_queries = len(queries)
data =  { #Ftiaxe ena dataframe gia to recall-precision diagramma opoy values oi times precision kai index oi times sta pososta gia to recall gia ola ta queries synolika
    "Precision": [0,0,0,0,0,0,0,0,0,0,0]
  }
Recall_Precision_df = pd.DataFrame(data, index = ["0%","10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"])

#Eisodos toy xrhsth gia posa apotelesmata thelei sto telos sxetika me thn omoiothta kai tis metrikes
k = -1
while k < 0 or k > 1210:
    k = int(input(f"Give a number of top results by similarity for each query to be presented(From 0 to {N-1}, -1 to exit): "))
    if k == -1:
        print("Goodbye!")
        exit()

start2 = time.time()
#Ypologise to similarity gia kathe query me kathe doc kai emfanise ta kalytera k apotelesmata
for query_id,item in enumerate(query_vectors):
    sim = []
    rdoc_list = []
    for doc_id in range(1, N):
        document_vector = vector_list[doc_id]
        similarity = index.cosine_similarity(item, document_vector)*100 #Ypologismos similarity apo thn sinartisi cosine similarity kai metatroph se pososto
        sim.append([similarity,doc_id]) #Lista poy periexei th timh omoiothtas enos query me ena doc kai to id toy doc aytoy
```

```
    sort_sim = sorted(sim, reverse=True) #Sort th lista apo to megalytero similarity sto mikrotero

    print(f"\nTop {k} results for query {query_id+1}:") #Emfanise ta k apotelesmata gia kathe query
    for i in range(k):
        rdoc_list.append(sort_sim[i][1]) #Sth lista me ta sxetika keimena vale ta kalytera k apotelesmata
        print(f"\tSimilarity score: {sort_sim[i][0]:.4f} \tDocument id: {sort_sim[i][1]}")

    num_rel = len(relevant_df.loc[query_id]['text']) #Ypologismos twn relevant stoixeiwn gia ena query apo to
dataframe me ta relevant keimena

    query_rp_df = pd.DataFrame(data, index =
["0%","10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"]) #Dataframe gia to ekastote
query ta apotelesmata toy opoioy tha prostethoyn sto geniko recall-precision dataframe

    count = 0 #Arithmos sxetikwn poy vriskontai stis apanthseis

    precision = [] #Lista gia to precision se kathe thesh poy yparxei sxetiko doc

    first = True #Metavlhth gia to prwto sxetiko doc gia ton ypologismo toy MRR

    for id in rdoc_list: #Gia kathe doc sth lista apanthsewn elegxe to me kathe doc sth lista sxetikwn
        for relevant in relevant_df.loc[query_id]['text']:
            if int(id) == int(relevant): #An to doc einai sxetiko me to query ayxhse to count
                count += 1
                position = rdoc_list.index(id) + 1 #Vres to position toy doc stis apanthseis (+1 kathos python xekina
apo to 0)
                p = count/position #Ypologise to precision sth thesh ayth vasei twn sxetikwn (p@k)
                precision.append(p) #Prosthese to sth lista precision

                if first: #An to first true tote to id einai to prwto sxetiko stis apanthseis ypologise to rr kai prosthese
to sth lista enw kane to first false
                    rr = 1/position
                    Reciprocal_Rank.append(rr)
                    first = False

                recall = (count/num_rel)*100 #Vres to pososto anaklhshs analoga me to posa sxetika exoyn vrethei
kathe fora

                if (0 <= recall) & (float(query_rp_df.loc["0%"]) == 0): #Gia kathe pososto mikrotero toy recall poy den
exei oristei hdh orise thn timh toy ws to precision epi %
                    query_rp_df.loc["0%"] = p*100
                if (10 <= recall) & (float(query_rp_df.loc["10%"]) == 0):
                    query_rp_df.loc["10%"] = p*100
                if (20 <= recall) & (float(query_rp_df.loc["20%"]) == 0):
                    query_rp_df.loc["20%"] = p*100
```

```
            if (30 <= recall) & (float(query_rp_df.loc["30%"]) == 0):
                query_rp_df.loc["30%"] = p*100
            if (40 <= recall) & (float(query_rp_df.loc["40%"]) == 0):
                query_rp_df.loc["40%"] = p*100
            if (50 <= recall) & (float(query_rp_df.loc["50%"]) == 0):
                query_rp_df.loc["50%"] = p*100
            if (60 <= recall) & (float(query_rp_df.loc["60%"]) == 0):
                query_rp_df.loc["60%"] = p*100
            if (70 <= recall) & (float(query_rp_df.loc["70%"]) == 0):
                query_rp_df.loc["70%"] = p*100
            if (80 <= recall) & (float(query_rp_df.loc["80%"]) == 0):
                query_rp_df.loc["80%"] = p*100
            if (90 <= recall) & (float(query_rp_df.loc["90%"]) == 0):
                query_rp_df.loc["90%"] = p*100
            if (100 == recall) & (float(query_rp_df.loc["100%"]) == 0):
                query_rp_df.loc["100%"] = p*100

    Recall_Precision_df = Recall_Precision_df.add(query_rp_df) #Prosthese ta apotelesmata gia recall-precision
toy query sto synoliko dataframe

    if first: #Se periptwsh poy stis apanthseis den yparxei sxetiko keimeno gia kapoio erwthma tote to RR einai
0
        Reciprocal_Rank.append(0)

    total = sum(precision) #Synolo twn epimeroys precision (An to precision einai keno tote total 0)

    if count != 0: #An yphrxan sxetika keimena stis apanthseis tote vres total einai to Average_Precision gia to
query alliws einai 0
        total /= count
    Average_Precision.append(total)

map = sum(Average_Precision)/len(Average_Precision) #Ypologismos MAP kai MRR
mrr = sum(Reciprocal_Rank)/len(Reciprocal_Rank)

Recall_Precision_df = (Recall_Precision_df/num_queries).round(4) #Sto dataframe me to synoliko recall-
precision gia ola ta queries diairese me ton arithmo twn queries gia na vreis tis meses times
ax = Recall_Precision_df.plot(title="Recall-Precision diagram", xlabel="Recall percentage", ylabel="Precision
percentage", yticks = [0,10,20,30,40,50,60,70,80,90,100]) #Ftiakse to diagramma recall-precision me ta
stoixeia toy dataframe
ax.set_xticks(range(len(Recall_Precision_df)))
ax.set_xticklabels(["0%","10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"])
ax.set_yticklabels(["0%","10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"])

print(f"\nThe MAP value over all 19 queries for k = {k} is equal to {map:.4f}") #Emfanish apotelesmatwn
metrikwn MAP kai MRR
```

```
print(f"\nThe MRR value over all 19 queries for k = {k} is equal to {mrr:.4f}")
end2 = time.time()
plt.show() #Emfanise to diagramma recall-precision

total1 = end1 - start1 #Ypologismos xronoy eyrethriashs
total2 = end2 - start2 #Ypologismos xronoy paragwghs apotelesmatwn

print(f"\nTotal elapsed time for the index to be created: {total1:.4f} seconds")
print(f"\nTotal time elapsed to produce the results: {total2:.4f} seconds")
```