Πανεπιστήμιο Πατρών, Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Ανάκτηση Πληροφορίας

Εργαστηριακή Άσκηση
Χειμερινό Εξάμηνο 2024

Βέργος Γεώργιος 1072604 up1072604@upnet.gr

Πάτρα Ιανουάριος 2024

Περιεχόμενα

Εισαγωγή

Ανεστραμμένο αρχείο(ευρετήριο)

Το ανεστραμμένο αρχείο – ευρετήριο(στη βιβλιογραφία αναφέρεται και ως αντεστραμμένος κατάλογος) αποτελεί την πιο διαδεδομένη μορφή καταλόγου για την αποθήκευση των όρων μιας συλλογής εγγράφων. Αυτό οφείλεται κυρίως σε δύο λόγους: **1)** Την απλότητα της δομής του και **2)** την καλή του απόδοση συγκριτικά με άλλες μορφές αποθήκευσης. Ένας αντεστραμμένος κατάλογος αποτελείται από δύο μέρη: **α)** Το λεξικό και **β)** τις λίστες εμφανίσεων. Το λεξικό αποτελεί το σύνολο των όρων των εγγράφων της συλλογής. Στην πιο απλή μορφή η λίστα εμφανίσεων περιλαμβάνει τους κωδικούς των εγγράφων που εμφανίζεται κάθε όρος. Αυτός λέγεται ανεστραμμένος κατάλογος επιπέδου εγγράφων. Σε πιο σύνθετες παραλλαγές του εκτός από τους κωδικούς των εγγράφων, καταχωρείται και η θέση εμφάνισης του όρου μέσα στο έγγραφο και το πλήθος των εμφανίσεων του μέσα στο έγγραφο.

Γενικά όσο αυξάνεται η λεπτομέρεια ενος ανεστραμμένου καταλόγου τόσο περισσότερο χώρο θέλει για την αποθήκευση του και τόσο λιγότερος χρόνος απαιτείται για την επεξεργασία των ερωτημάτων. Ένας ανεστραμμένος κατάλογος επιπέδου όρων, δηλαδή εκτός από τους κωδικούς των εγγράφων στα οποία ανήκει κάθε όρος αποθηκεύονται επίσης οι θέσεις των εμφανίσεων του όρου μέσα στο έγγραφο, απαιτεί περισσότερο αποθηκευτικό χώρο. Αυτό οφείλεται σε δύο λόγους: **1)** εκτός από τον κωδικό αριθμό του εγγράφου αποθηκεύεται και η θέση του όρου μέσα στο έγγραφο, και **2)** σε περίπτωση που ένας όρος εμφανίζεται περισσότερες φορές σε ένα έγγραφο καταγράφονται όλες οι εμφανίσεις του.

Σε ένα ιδανικό σενάριο τόσο το λεξικό όσο και οι λίστες εμφάνισης αποθηκεύονται στην κύρια μνήμη. Ωστόσο σε πραγματικές εφαρμογές, το πλήθος και το μέγεθος των κειμένων είναι τόσο μεγάλο που συνηθίζεται οι λίστες εμφάνισης να

αποθηκεύονται σε δευτερεύουσα μνήμη ενώ το λεξικό μόνο να αποθηκεύεται στην κύρια μνήμη.

Μοντέλο διανυσματικού χώρου(Vector Space Model)

Το διανυσματικό μοντέλο(vector space model) αποτελεί ένα από τα βασικά και γνωστότερα μοντέλα ανάκτησης πληροφορίας. Ανήκει στην κατηγορία των αλγεβρικών μοντέλων ανάκτησης. Σύμφωνα με αυτό κάθε έγγραφο dj της συλλογής αναπαρίσταται ως ένα διάνυσμα dj = $(w_{t1,dj}, w_{t2,dj}, ..., w_{tM,dj})$ όπου M το πλήθος των όρων που υπάρχουν στην συλλογή και $w_{ti,dj}$ το βάρος του όρου t_i στο έγγραφο d_j. Υπάρχουν πολλές επιλογές για τις τιμές που λαμβάνει το $w_{ti,dj}$. Στην πιο απλή μορφή παίρνει τις τιμές 1 ή 0(δυαδικό βάρος) που υποδηλώνουν την ύπαρξη ή μη του όρου μέσα στο κείμενο. Ωστόσο η χρήση δυαδικών βαρών δε λαμβάνει υπόψη ούτε τη συχνότητα εμφάνισης του όρου στο έγγραφο αλλά ούτε και τον αριθμό των εγγράφων στα οποία εμφανίζεται ο όρος αυτός. Δηλαδή εάν ένας όρος εμφανίζεται πολλές φορές μέσα σε ένα έγγραφο τότε η σημαντικότητα του για το έγγραφο θα πρέπει να είναι μεγαλύτερη από έναν όρο που εμφανίζεται μόνο μία φορά στο έγγραφο αυτό. Επιπλέον αν ένας όρος εμφανίζεται σε πολλά έγγραφα τότε δεν αποτελεί χαρακτηριστικό κάποιου συγκεκριμένου. Γι'αυτό τον λόγο προτάθηκε η χρήση του σχήματος tf - idf(term frequency - inverse document frequency) για τον υπολογισμό των βαρών των όρων στα κείμενα. Μία πρώτη προσέγγιση είναι να χρησιμοποιηθεί ως βάρος ενός όρου t_i στο έγγραφο d_j η απλή συχνότητα εμφάνισης του όρου στο έγγραφο(πόσες φορές εμφανίζεται στο έγγραφο ο συγκεκριμένο όρος) δηλαδή να έχουμε : $w_{ti,dj} = f_{ti,dj}$. Ωστόσο με τη χρήση της προηγούμενης σχέσης για τον υπολογισμό του βάρους, όροι που βρίσκονται σε μεγάλα έγγραφα πιθανώς θα έχουν και μεγαλύτερο βάρος γιατί αυξάνεται η πιθανότητα ύπαρξης τους μέσα στο έγγραφο. Για να μην υπάρχει τέτοια

διάκριση κανονικοποιείται ο όρος $f_{ti,dj}$ και έτσι έχουμε τον όρο $nf_{ti,dj} = f_{ti,dj} / \max_x\{f_{x,d}\}$.
Ωστόσο προκύπτει ένα ακόμη πρόβλημα: Όροι που εμφανίζονται πολλές φορές μέσα σε κάποια έγγραφα μπορεί να εμφανίζονται σε πολλά έγγραφα. Αυτό μειώνει τη σημαντικότητα τους. Άρα το συνολικό βάρος του όρου θα είναι μικρό αφού δε θα αποτελεί αντιπροσωπευτική λέξη για κανένα έγγραφο. Επομένως προτάθηκε και η χρήση της αντίστροφης συχνότητας εγγράφου idf_{ti} όπου $idf_{ti} = \ln(N/n_i)$, όπου N ο αριθμός των εγγράφων της συλλογής και n_i ο αριθμός των εγγράφων στα οποία εμφανίζεται ο όρος t_i.
Έτσι υπολογίζεται μέσω του σχήματος tf - idf το βάρος του όρου μέσα στο κείμενο : $W_{ti,dj} = nf_{ti,dj} \cdot idf_{ti} = f_{ti,dj} / \max_x \{f_{x,d}\} \cdot \ln(N/n_i)$. Ο παράγοντας $\ln(N/n_i)$ μπορεί να κανονικοποιηθεί αντιστοίχως. Γενικά υπάρχουν διάφοροι τρόποι για τον υπολογισμό του βάρους $w_{ti,dj}$, μερικοί από αυτούς φαίνονται στον παρακάτω πίνακα:

Τρόπος υπολογισμού	**$TF_{i,j}$**
Δυαδικό	$\{0, 1\}$
Απλή συχνότητα εμφάνισης	**$F_{i,j}$**
Απλή λογαριθμική κανονικοποίηση	**$1 + \log(F_{i,j})$**
Διπλή 0.5 κανονικοποίηση	**$0.5 + 0.5 \cdot F_{i,j} / \max_x\{F_{x,d}\}$**
Διπλή K κανονικοποίηση(γενίκευση του παραπάνω, για τιμή K=0.5 φέρνει τα καλύτερα αποτελέσματα.	**$K + (1-K) \cdot F_{i,j} / \max_x \{F_{x,d}\}$**

Και:

Τρόπος υπολογισμού	**IDF_i**
Μοναδιαίο	**1**
Απλή αντίστροφη συχνότητα εμφάνισης	**$\log(N/n_i)$**
Απλή λογαριθμική κανονικοποίηση	**$\log(1 + N/n_i)$**
Αντίστροφη κανονικοποίηση περισσότερων εμφανίσεων(εναλλακτικά πρώτος κανονικοποιημένος μετασχηματισμός)	**$\log(1 + \max_x\{n_x\} / n_i)$**

Υπάρχουν και άλλοι διάφοροι τρόποι υπολογισμού των βαρών που αναφέρονται στην βιβλιογραφία παρακάτω.

Ομοιότητα εγγράφων και ερωτημάτων στο μοντέλο διανυσματικού χώρου

Για να υπολογιστεί η ομοιότητα ενός ερωτήματος q και ενός εγγράφου d, $S_{vector}(q,d)$, έχουν προταθεί διάφοροι τρόποι. Αρχικά έχοντας ένα κείμενο d_j το οποίο αναπαρίσταται ως διάνυσμα $\vec{d} = (w_{t1,dj}, w_{t2,dj}, \ldots, w_{tM,dj})$ και ένα ερώτημα $\vec{q} = (w_{t1,q}, w_{t2,q}, \ldots, w_{tM,q})$ όπου $w_{ti,q} = tf_{ti,q} \cdot idf_{ti}$ επιχειρούμε να βρούμε την ομοιότητα των διανυσμάτων αυτών. Μια πρώτη προσέγγιση είναι να χρησιμοποιήσουμε την ευκλείδεια απόσταση μεταξύ q και d η οποία περιγράφεται από τη σχέση: $D_e(q,d) = \sqrt{\sum_{i=1}^{M} \left| w_{ti,q} - w_{ti,d} \right|^2}$.

Η χρήση της ευκλείδειας απόστασης ωστόσο παρουσιάζει ένα βασικό πρόβλημα. Συνήθως το έγγραφο του ερωτήματος είναι αρκετά μικρότερο από τα έγγραφα της συλλογής γεγονός που οδηγεί το διάνυσμα q να έχει πάρα πολλές μηδενικές συνιστώσες. Ακόμη όσο μεγαλύτερο είναι ένα έγγραφο τόσες περισσότερες μη-μηδενικές συνιστώσες έχει. Αυτό έχει ως αποτέλεσμα να τιμωρούνται τα μεγαλύτερα έγγραφα μιας και η ευκλείδεια απόσταση τους από το ερώτημα θα είναι μεγάλη ακόμη και αν σχετίζονται με το ερώτημα.

Μία δεύτερη προσέγγιση είναι να χρησιμοποιήσουμε ως μετρική ομοιότητας ερωτήματος - εγγράφου το εσωτερικό γινόμενο των διανυσμάτων τους, το οποίο περιγράφεται από τη σχέση: $S_{inner}(q,d) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{M} w_{ti,q} \bullet w_{ti,d}$.

Όσο πιο όμοια είναι τα διανύσματα τόσο μεγαλύτερες τιμές παίρνει το εσωτερικό γινόμενο και αντιστρόφως. Παρ'όλα αυτά η χρήση του εσωτερικού γινομένου έχει το μειονέκτημα ότι τιμωρούνται τα μικρότερα έγγραφα.

Λύση στα δύο παραπάνω είναι η χρήση της μετρικής του συνημιτόνου της γωνίας που σχηματίζουν τα διανύσματα q και d. Ως γνωστόν το εσωτερικό γινόμενο δύο διανυσμάτων q και d δίνεται από τη σχέση: $\vec{q} \bullet \vec{d} = \left|\vec{q}\right| \bullet \left|\vec{d}\right| \bullet cos(\theta)$, όπου θ η γωνία που σχηματίζουν μεταξύ τους τα διανύσματα $\vec{q}$ και $\vec{d}$. Όσο μεγαλύτερη τιμή παίρνει η γωνία θ τόσο μικρότερη τιμή παίρνει η τιμή cos(θ) και έτσι τόσο πιο ανόμοια είναι τα διανύσματα μεταξύ τους και αντιστρόφως. Έτσι έχουμε την περίπτωση που τα διανύσματα ταυτίζονται δηλαδή θ = 0 και cos(θ) = 1, με άλλα λόγια έχουμε πλήρη ομοιότητα ε γγράφου και ερωτήματος και την περίπτωση όπου θ = 90 και cos(θ) = 0 δηλαδή καμία ομοιότητα εγγράφου και ερωτήματος. Η μετρική του συνημιτόνου επομένως παίρνει τιμές στο διάστημα [0,1]. Τέλος ένα πλεονέκτημα της μετρικής συνημιτόνου είναι η ανεξαρτησία της από την επιλογή των βαρών $w_{ti,dj}$.

Πλεονεκτήματα και μειονεκτήματα

Το μοντέλο διανυσματικού χώρου διακρίνεται για την απλότητα της υλοποίησης του, αφού απαιτούνται απλές μαθηματικές πράξεις για τον υπολογισμό των βαρών. Ωστόσο οι πράξεις αυτές μπορούν να γίνουν κοστοβόρες εάν ο αριθμός των όρων και των εγγράφων είναι μεγάλος(συνήθως χιλιάδες). Το μοντέλο διανυσματικού χώρου επιτρέπει την ύπαρξη μερικής ομοιότητας μεταξύ ερωτήματος και κειμένου δηλαδή ένα έγγραφο ακόμη και να μην περιλαμβάνει όλους τους όρους του ερωτήματος θα λάβει μη μηδενικό βαθμό. Αυτό δεν είναι δυνατό σε άλλα μοντέλα όπως το Boolean μοντέλο.
Παρ'όλα αυτά το μοντέλο διανυσματικού χώρου παρουσιάζει ένα βασικό μειονέκτημα. Υποθέτει ότι οι όροι είναι μεταξύ τους ανεξάρτητοι. Αυτό δεν είναι πάντα αληθές σε πραγματικές εφαρμογές μιας και η εμφάνιση κάποιου όρου μπορεί να εξαρτάται από την εμφάνιση άλλων όρων. Ως εκ τούτου δεν μπορεί να αντιμετωπίσει δύο από τα μεγάλα προβλήματα στην ανάκτηση πληροφορίας την συνωνυμία και την πολυσημία οι οποίες με τη σειρά τους επηρεάζουν την ποιότητα της ανάκτησης(και μετρικές όπως ανάκληση και ακρίβεια). Τέλος σε αντίθεση με το Boolean μοντέλο ο υπολογισμός των βαρών είναι αυθαίρετος και όχι φορμαλιστικός.

Το μοντέλο colBERT

Το **colBERT**(**contextual late interaction over BERT**) είναι ένα γρήγορο και ακριβές γλωσσικό μοντέλο το οποίο ισορροπεί μεταξύ απόδοσης και κατανόησης κειμένου από τα συμφραζόμενα. Επιτρέπει γρήγορη αναζήτηση σε τεράστιες συλλογές κειμένων. Προηγούμενες υλοποίησεις BERT μοντέλων ενώ ήταν αποδοτικές, είχαν τεράστιο υπολογιστικό κόστος στην εκπαίδευση τους, επομένως η ανάκτηση των κειμένων ήταν αργή. Από την άλλη μοντέλα που δεν χρησιμοποιούν το BERT όπως μοντέλα που βασίζονται στο tf-idf σχήμα δεν είχαν τα επιθυμητά

αποτελέσματα, παρέμεναν ωστόσο υπολογιστικά αποδοτικά. Στο colBERT το ερώτημα και τα κείμενα της συλλογής κωδικοποιούνται ξεχωριστά χρησιμοποιώντας δύο μοντέλα **BERT,** σε διανύσματα συμφραζόμενων . Τα διανύσματα συμφραζομένων είναι διανύσματα που παράγονται ως έξοδοι των δύο παραπάνω **BERT** μοντέλων. Τα δύο σύνολα των διανυσμάτων αυτών(το πρώτο σύνολο αφορά το ερώτημα και το άλλο τα κείμενα) ''επιβλέπουν'' το ένα το άλλο και έτσι υπολογίζουν ένα βαθμό σχετικότητας μεταξύ κειμένου ερωτήματος. Το κείμενο με τον υψηλότερο βαθμό παίρνει κατάταξη 1 κ.ο.κ. Ο υπολογισμός του βαθμού φαίνεται στην παρακάτω εικόνα:

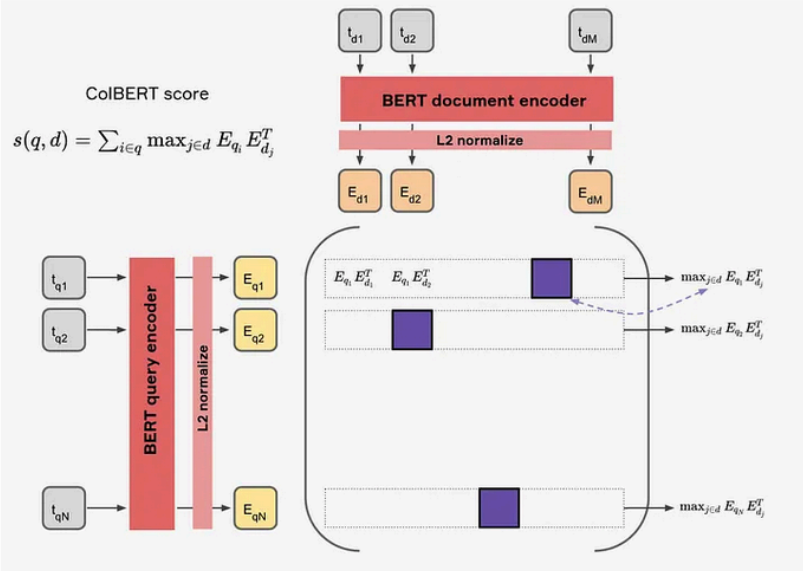
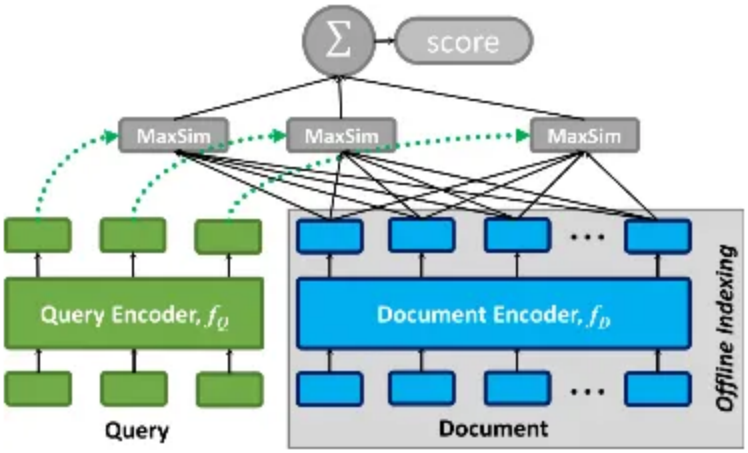


Figure 4: Late-interaction mechanism of ColBERT. Magenta coloured cells represents which document encoding has highest similarity with the corresponding query encoding.

Αρχιτεκτονική

Η αρχιτεκτονική του μοντέλου colBERT περιγράφεται από το παρακάτω σχήμα:



Ένα προ-εκπαιδευμένο μητρώο χρησιμοποιείται για να παράγει τα tokens των κειμένων και του ερωτήματος. Αυτά τα tokens μπαίνουν ως είσοδος σε δύο ξεχωριστά BERT μοντέλα. Οι έξοδοι των μοντέλων αυτών που είναι διανύσματα μπαίνουν ως είσοδος στο μητρώο παραπάνω όπου υπολογίζεται κάθε βαθμός ως εξής:

ColBERT score

$$s(q,d) = \sum_{i \in q} \max_{j \in d} E_{q_i} E_{d_j}^T$$

. Αναλυτικότερη περιγραφή υπάρχει στη βιβλιογραφία.

Μετρικές απόδοσης

Για να μετρήσουμε την απόδοση ενός συστήματος ανάκτησης πληροφορίας, χρησιμοποιούμε διάφορες μετρικές. Από αυτές οι δύο βασικότερες αποτελούν την **ανάκληση**(**Recall**) και την **ακρίβεια**(**Precision**). Αρχικά έστω ότι έχουμε μία πληροφοριακή ανάγκη **I** και το σύνολο της **R**, των σχετικών της εγγράφων. Έπειτα έστω ότι το σύστημα ανάκτησης πληροφορίας μέσω κάποιου

εξειδικευμένου αλγορίθμου επεξεργάζεται την πληροφοριακή ανάγκη μας και επιστρέφει ένα σύνολο απαντήσεων **Α. Ανάκληση** ορίζεται ως το ποσοστό των σχετικών εγγράφων το οποίο έχει ανακτηθεί δηλαδή: ανάκληση = r = $\left|R \cap A\right| \div |R|$. Αντίστοιχα η ακρίβεια ορίζεται ως το ποσοστό των ανακτημένων εγγράφων το οποίο είναι σχετικό δηλαδή: ακρίβεια = p = $\left|R \cap A\right| \div |A|$.

Αναφορές

Papadopoulos, Apostolos, Ioannis Manolopoulos, and Konstantinos Tsichlas. 2015. *Ανάκτηση πληροφορίας*.https://repository.kallipos.gr/handle/11419/4191.

Salton, Gerald, and Christopher Buckley. 1988. “Term-weighting approaches in automatic text retrieval.” *Information Processing & Management* 24 (5): 513-523. 0306-457.

Salton, Gerald, Andrew Wong, and Chungshu Yang. 1975. “A vector space model for automatic indexing.” *Communications of the ACM* 18, no. 11 (November): 613-620.https://doi.org/10.1145/361219.361220.

Yates, Ricardo B., and Berthier R. Neto. 2010. *Modern Information Retrieval*. N.p.: ACM Press Books.

colBERT explained

Υλοποίηση

Καταγραφή του περιβάλλοντος υλοποίησης και βιβλιοθήκες

Ως γλώσσα υλοποίησης χρησιμοποιώ την **Python,** έκδοση **3.10.** Για την ανάπτυξη της άσκησης χρησιμοποιήθηκε το εργαλείο(IDE) PyCharm της JetBrains. Χρησιμοποιήθηκαν οι ακόλουθες βιβλιοθήκες της Python:

1) **os** για το διάβασμα των αρχείων της συλλογής **Cystic Fibrosis.**
2) **pandas** για τη δημιουργία, επεξεργασία και αποθήκευση των μητρώων που περιέχουν την απαραίτητη πληροφορία για την υλοποίηση του μοντέλου διανυσματικού χώρου και των μετρικών απόδοσης.
3) **numpy** για μαθηματικούς υπολογισμούς όπως πράξεις πινάκων και μετασχηματισμοί αυτών.
4) **colbert** για την υλοποίηση του colBERT μοντέλου.
5) **matplotlib** για την απεικόνιση των μετρικών απόδοσης(λ.χ το διάγραμμα ανάκλησης – ακρίβειας).
6) **Scipy** για την απόδοση αριθμού κατάταξης στα κείμενα βάση της ομοιότητας τους με το ερώτημα.
7) **re** για την προ-επεξεργασία των κειμένων μέσω κανονικών εκφράσεων.

Παρακάτω επισυνάπτω τα στιγμιότυπα με τις βιβλιοθήκες κάθε ερωτήματος:

Ερωτήματα 1-2

```
import numpy as np
import pandas as pd
import os
import re
```

Ερώτημα 3

```
import colbert

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

Ερωτήμα 4

```
import pandas as pd
import numpy as np
from scipy.stats import rankdata
from matplotlib import pyplot as plt
```

Η εγκατάσταση των παραπάνω έγινε μέσω powershell(Περιβάλλον Windows 10) με την εκτέλεση της εντολής:
pip3 install <όνομα πακέτου>.

Αλγοριθμική λογική της υλοποίησης

Αρχικά μέσω της βιβλιοθήκης **os** διαβάζω τα αρχεία και αποθηκεύω το περιεχόμενο τους σε μία λίστα. Πριν την αποθήκευση τους στη λίστα εφαρμόζω μερικές τεχνικές προ-επεξεργασίας. Τα κείμενα μετατρέπονται εξ'ολοκλήρου σε **πεζά** γράμματα, αφαιρούνται ειδικοί χαρακτήρες από τα κείμενα(σημεία στίξης, παύλες, τελείες, κόμματα κ.ο.κ), πλεονάζοντα κενά συνενώνονται σε ένα. Κάθε λέξη χωρίζεται από

την προηγούμενη και επόμενη της με ένα μόνο κενό. Έπειτα με χρήση μίας **stop list** αφαιρούνται οι λέξεις με πολύ υψηλή συχνότητα εμφάνισης μέσα στα κείμενα, λέξεις τις οποίες δεν μας ενδιαφέρει να δεικτοδοτήσουμε. Έπειτα κατασκευάζω το ανεστραμμένο ευρετήριο **επιπέδου εγγράφων** ως εξής: Αν μία λέξη υπάρχει ήδη στο ευρετήριο απλά πρόσθεσε το id του κειμένου που ανήκει. Αλλιώς πρόσθεσε το id του κειμένου που ανήκει η λέξη ως το πρώτο κείμενο που ανήκει η λέξη αυτή.
Για την υλοποίηση του μοντέλου διανυσματικού χώρου έχοντας προ-επεξεργαστεί τα κείμενα όπως προηγουμένως και κατασκευάζοντας το ανεστραμμένο ευρετήριο του πρώτου ερωτήματος , μέσω της βιβλιοθήκης **pandas** κατασκευάζω και υπολογίζω τα εξής μητρώα: **Ni**(αριθμός των κειμένων που εμφανίζεται κάθε όρος) όπου εδώ χρησιμοποιώ το ανεστραμμένο ευρετήριο του πρώτου ερωτήματος, **idf**(διάνυσμα αντίστροφων συχνοτήτων των όρων), **tf**(μητρώο συχνοτήτων εμφάνισης των όρων σε κάθε κείμενο και οι παραλλαγές του), **weight**(μητρώο βαρών των όρων σε κάθε κείμενο σύμφωνα με το σχήμα **tf-idf**).
Τέλος κατασκευάζω ένα ερώτημα δοκιμής, με την ίδια διαδικασία υπολογίζω το διάνυσμα βαρών των όρων του ερωτήματος αφού το προ-επεξεργαστώ(μετατροπή σε πεζά,αφαίρεση ειδικών χαρακτήρων και κενών), υπολογίζω το εσωτερικό γινόμενο του ερωτήματος με όλα τα κείμενα της συλλογής(αφού πρώτα μετασχηματίσω το διάνυσμα της ερώτησης ώστε να έχει τις ίδιες διαστάσεις με αυτά των κειμένων) και το διαιρώ με το γινόμενο του μέτρου του διανύσματος του ερωτήματος και των μέτρων των κειμένων. Έτσι προκύπτει ένα διάνυσμα συνημιτόνων των γωνιών που σχηματιζει το διάνυσμα του ερωτήματος με το διάνυσμα κάθε κειμένου.
Για την υλοποίηση των δύο μετρικών(**ανάκληση** και **ακρίβεια**) αρχικά κατασκευάζω ένα array μέσω της βιβλιοθήκης numpy που περιέχει τα Ids των κειμένων που ανακτήθηκαν από το διάνυσμα **cosine_similarity**(στην ουσία περιέχει τους δείκτες του πίνακα **cosine_similarity**). Κατασκευάζονται δύο ακόμη arrays , το **score** που περιέχει τις τιμές του πίνακα **cosine_similarity** και το **relevant_docs** που περιέχει τα σχετικά ως προς το ερώτημα κείμενα(δίνονται στο αρχείο **cfquery_detailed**). Έπειτα μέσω των συναρτήσεων **flip()** και

argsort() της **numpy** παίρνω τους δείκτες(indices) του πίνακα score προκειμένου στη συνέχεια να διατάξω τα ids των κειμένων σε φθίνουσα σειρά βάσει τη βαθμολογίας τους. Στη συνέχεια μέσω της **rankdata()** της **scipy** αναθέτω στο υψηλότερο score τιμή rank 1, στο δεύτερο υψηλότερο την τιμή 2 κ.ο.κ. Έπειτα διατρέχω όλα τα κείμενα και αν κάποιο από αυτά ανήκει και στον πίνακα **relevant_docs**, άρα είναι σχετικό, το σημειώνω ως σχετικό(true) και αντίστοιχα false αν είναι μη σχετικό. Τέλος βάσει ορισμού σε κάθε θέση του πίνακα υπολογίζω τιμές της ανάκλησης και της ακρίβειας. Για την **ακρίβεια** για κάθε θέση του πίνακα, υπολογίζω το πλήθος των σχετικών κειμένων που έχουν βρεθεί μέχρι τη θέση αυτή μέσω της συνάρτησης **sum()** και διαιρώ το αποτέλεσμα με την τιμή της θέσης αυτής. Αντίστοιχα για την **ανάκληση** υπολογίζω το πλήθος των σχετικών κειμένων που έχουν ανακτηθεί μέχρι τη θέση αυτή μέσω της συνάρτησης **sum()** και διαιρώ το αποτέλεσμα με το μέγεθος του array **relevant_docs** δηλαδή των πλήθος των σχετικών κειμένων όπως έχουν οριστεί από τους ειδικούς. Ενώνω τα αποτελέσματα σε δύο Dataframes και μέσω της δικής μου συνάρτησης **Precision_Recall_Graph()** κατασκευάζω το διάγραμμα ανάκλησης-ακρίβειας.

Αποτελέσματα – Παρατηρήσεις

Αρχικά όσων αφορά το ερώτημα 1, δημιουργώ ένα ανεστραμμένο ευρετήριο επιπέδου εγγράφων δηλαδή μία λίστα με τους όρους και δίπλα τους τα id των κειμένων όπου εμφανίζονται. Ένα παράδειγμα εκτέλεσης φαίνεται στην παρακάτω εικόνα:

```
inverted_index = {}#anestrammeno evretirio
for i,doc in enumerate(doc_files): #gia kathe keimeno arxika pare to idio to keimeno kai th thesh tou sth lista keimenon
    for word in doc.lower().split():#gia kathe leksi sto ekastote keimeno
        if word in inverted_index:#
            inverted_index[word].add(i)#an einai hdh sth lista prosthese to id tou keimenou pou vrisketai h leksi
        else:
            inverted_index[word] = {i}#an den einai sth lista o oros tote prosthese thn emfanisi tou sto keimeno doc

#for i,doc in enumerate(doc_files):
    #print(i,doc)
print('Documents where cardiopulmonary exists:',inverted_index['cardiopulmonary'])
#print(doc_files[824])
```

Και αποτέλεσμα:

```
Number of terms :
 11367
Own stopwords list length: 68

With stopwords: 11367

Without stopwords: 11299
Documents where cardiopulmonary exists: {824, 1016, 986, 822}

Process finished with exit code 0
```

Δηλαδή ο όρος 'cardiopulmonary' εμφανίζεται σε 4 κείμενα με id: 824,1016,986,822. Το ευρετήριο αυτό είναι χρήσιμο για το ερώτημα 2(υλοποίηση μοντέλου διανυσματικού χώρου) για να εξάγουμε από αυτό, το πλήθος των κειμένων που βρίσκεται μία λέξη δηλαδή τον όρο n_i.

Όσων αφορά το ερώτημα 2 κατασκευάζω τους απαραίτητους πίνακες για να αναπαρασταθούν τα κείμενα και το ερώτημα δοκιμής ως διανύσματα. Ένα παράδειγμα εκτέλεσης που φαίνεται το μητρώο βαρών των όρων φαίνεται στις παρακάτω εικόνες:

```
Number of documents in our collection:
 1209
Waiting...

Number of terms :
 11367
WAIT
Number of documents where cardiopulmonary exists: 4.0
WAIT
```

Μερικές τιμές term-frequency, inverse document frequency του όρου 'cystic'.

```
Inverse document frequency of cystic term: 0.14420150645656704
Number of documents where cystic exists: 1094.0
Term frequency i.e number of occurences of cystic term in the last document: 4.0
409.0
Weight of term: cystic, in document 1208(the last document in the collection) 0.5768060258262682
```

Μετά τον καθαρισμό του ερωτήματος δοκιμής:

```
[5 rows x 11299 columns]
Cleaning query of extra spaces and special characters: what are the effects of calcium on the physical properties of mucus from cf patients
['what', 'are', 'the', 'effects', 'of', 'calcium', 'on', 'the', 'physical', 'properties', 'of', 'mucus', 'from', 'cf', 'patients']
STOP
```

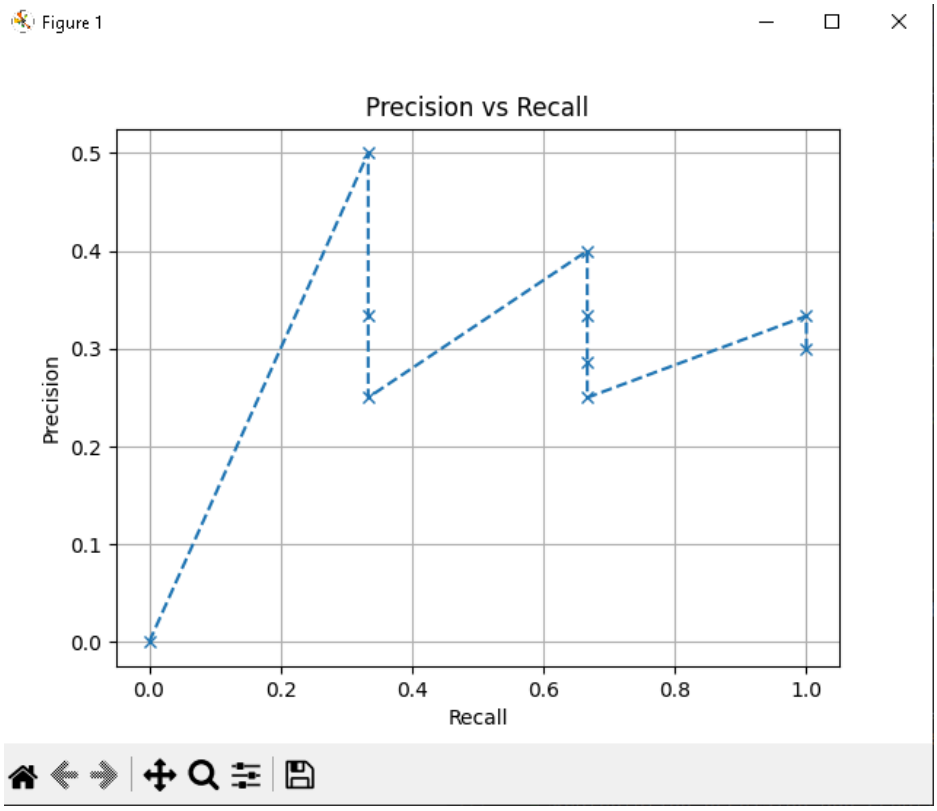
Μετά την εκτέλεση του εσωτερικού γινομένου και της γωνίας μεταξύ κειμένων και ερωτήματος παίρνω τα 10 πιο σχετικά κείμενα(τα id τους και την τιμή της ομοιότητας τους με το ερώτημα):

```
10 most matched documents:
  427    0.226632
473    0.211563
484    0.195829
952    0.172932
721    0.171652
516    0.156505
807    0.154499
734    0.149619
509    0.146387
294    0.142169
dtype: float64
```

Όσων αφορά το ερώτημα 4, για το ίδιο ερώτημα δοκιμής και τυχαίο σύνολο σχετικών κειμένων(έστω τα 721,509,473 είναι σχετικά) παίρνω τους εξής πίνακες με τιμές ανάκλησης-ακρίβειας για κάθε επίπεδο ανάκλησης. Ακόμη φαίνεται και το αντίστοιχο διάγραμμα ανάκλησης-ακρίβειας:

	Doc id	Score	Rank
0	427	0.226632	1
1	473	0.211563	2
2	484	0.195829	3
3	952	0.172932	4
4	721	0.171652	5
5	516	0.156505	6
6	807	0.154499	7
7	734	0.149619	8
8	509	0.146387	9
9	294	0.142169	10

	Rank	Doc id	Relevant	Precision	Recall
0	1	427	False	0.000000	0.000000
1	2	473	True	0.500000	0.333333
2	3	484	False	0.333333	0.333333
3	4	952	False	0.250000	0.333333
4	5	721	True	0.400000	0.666667
5	6	516	False	0.333333	0.666667
6	7	807	False	0.285714	0.666667
7	8	734	False	0.250000	0.666667
8	9	509	True	0.333333	1.000000
9	10	294	False	0.300000	1.000000



Ως τιμή για το $\mathbf{TF_{i,j}}$ χρησιμοποίησα την απλή συχνότητα εμφάνισης($\mathbf{F_{i,j}}$) ενώ ως τιμή για την $\mathbf{IDF_i}$ την τιμή $\mathbf{log_2(N \div n_i)}$, για τον καθορισμό των βαρών των όρων στα κείμενα. Για τον υπολογισμό των βαρών των όρων του ερωτήματος χρησιμοποίησα τον ακόλουθο τύπο που έχει γενικά καλή απόδοση στις περισσότερες συλλογές:

$$w_{t,q} = \left(0.5 \cdot \frac{f_{t,q}}{max_x\{f_{x,q}\}} + 0.5\right) \cdot \ln\left(\frac{N}{n_t}\right)$$

Η απόδοση των μοντέλων θα μπορούσε να αυξηθεί με περαιτέρω προ-επεξεργασία των κειμένων. Για παράδειγμα θα μπορούσα να έκανα χρήση κάποιου stemmer(όπως ο Porter Stemmer) για την αφαίρεση των καταλήξεων, λημματοποίηση για την αναγωγή των λέξεων σε κάποια βασική τους αρχική λέξη. Δυστυχώς υπάρχουν ορισμένα προβλήματα όπως η συνωνυμία που αναπόφευκτα επηρεάζει αρνητικά την απόδοση των μοντέλων(συγκεκριμένα την ανάκληση).

Αναφορές

References

Papadopoulos, Apostolos, Ioannis Manolopoulos, and Konstantinos Tsichlas. 2015. *Ανάκτηση πληροφορίας*.https://repository.kallipos.gr/handle/11419/4191.

Salton, Gerald, and Christopher Buckley. 1988. “Term-weighting approaches in automatic text retrieval.” *Information Processing & Management* 24 (5): 513-523. 0306-457.

Salton, Gerald, Andrew Wong, and Chungshu Yang. 1975. “A vector space model for automatic indexing.” *Communications of the ACM* 18, no. 11 (November): 613-620. https://doi.org/10.1145/361219.361220.

Yates, Ricardo B., and Berthier R. Neto. 2010. *Modern Information Retrieval*. N.p.: ACM Press Books.

Παράρτημα

Κώδικας

Ερώτημα 1

```
import os #vivliothiki gia diavasma arxeion
doc_files=[] #ta keimena os stixia listas
```

```
#

doc_path = os.getcwd() + "\\Collection\\docs" #kataskeyi tou monopatiou pou
vriskontai ta keimena
#
print(doc_path)#ektyposi aytou tou monopatiou
for file in sorted(os.listdir(doc_path)):#gia kathe string(noumero) ths listas
tou path
   file_path = os.path.join(doc_path,file)#ousiastika .../docs/px-1239 kanei
join to px ../docs me to 1239 kai vgenei: ../docs/1239
   #print(file_path)
   #print(type(file)) #- einai strings
   #print(int(file))
   if os.path.isfile(file_path):#an to ../docs/1239 px einai arxeio tote:
      f = open(file_path,'r')#anikse to ekastote arxeio se reading mode
      doc_files.append(f.read())#diavase to kai to periexomeno kane to eisagogi
sth lista eggrafon

##
print('\nNumber of documents in our collection: \n',len(doc_files))#1209
documents
#print('\nPrinting the first document: \n',doc_files[0])
input('Waiting...')
tokens =[]#tokens as a list per document

for i in range(len(doc_files)):
   tokens.append(doc_files[i].lower().split())#apo ta keimena pare kathe
keimeno ksexorista kai spasto se lekseis. valta os i-osto stixio
   #th lista ayth ton lekseon sta tokens. diladi px. keimeno 5 5-th stixio ton
tokens einai h lista me tis lekseis tou keimenou 5

tokens_all = []#all tokens in the collection

for i in range(len(tokens)):
   tokens_all = tokens_all + tokens[i]#enono ola ta tokens se mia eniaia lista
#print(tokens)

#print('\nAll tokens in the collection: \n',tokens_all)
terms = list(set(tokens_all))#perno tis monadikes times ton lekseon apo ayth th
lista kai etsi exo tous orous to leksilogio mou
print('\nUnique terms gathered from the document collection\n',terms)
print('\nNumber of terms : \n',len(terms))

###---placeholder for preprocessing

#
words = []
stop_words = ["a", "an", "the", "and", "but", "or", "because", "as", "until",
"while", "of", "at", "by", "for", "with", "about", "against", "between",
```

```
"into", "through", "during", "before", "after", "above", "below", "to", "from",
"up", "down", "in", "out", "on", "off", "over", "under", "again", "further",
"then", "once", "here", "there", "when", "where", "why", "how", "all", "any",
"both", "each", "few", "more", "most", "other", "some", "such", "no", "nor",
"not", "onλy", "own", "same", "so", "than", "too", "very", "can", "will",
"just"]
for word in terms:
   if word not in stop_words:
      words.append(word)#apaloifh ton stop words
print('Own stopwords list length:',len(stop_words))
print('\nWith stopwords:',len(terms))
print('\nWithout stopwords:',len(words))


inverted_index = {}#anestrammeno eyretirio
for i,doc in enumerate(doc_files): #gia kathe keimeno arxika pare to idio to
keimeno kai th thesh tou sth lista keimenon
   for word in doc.lower().split():#gia kathe leksi sto ekastote keimeno
       if word in inverted_index:#
           inverted_index[word].add(i)#an einai hdh sth lista prosthese to id
tou keimenou pou vrisketai h leksi
       else:
           inverted_index[word] = {i}#an den einai sth lista o oros tote
prosthese thn emfanisi tou sto keimeno doc

#for i,doc in enumerate(doc_files):
   #print(i,doc)
print(inverted_index['cardiopulmonary'])
#print(doc_files[824])
```

Ερώτημα 2

```
import numpy as np
import pandas as pd
import os
import re
#vivliothikes
doc_files=[] #ta keimena os stixia listas
#

doc_path = os.getcwd() + "/Collection/docs" #kataskeyi tou monopatiou pou
vriskontai ta keimena

#
print(doc_path)#ektyposi aytou tou monopatiou
```

```
for file in sorted(os.listdir(doc_path)):#gia kathe string(noumero) ths listas
tou path
   file_path = os.path.join(doc_path,file)#ousiastika .../docs/px-1239 kanei
join to px ../docs me to 1239 kai vgenei: ../docs/1239
   #print(file_path)
   #print(type(file)) #- einai strings
   #print(int(file))
   if os.path.isfile(file_path):#an to ../docs/1239 px einai arxeio tote:
      f = open(file_path,'r')#anikse to ekastote arxeio se reading mode
      doc_files.append(f.read())#diavase to kai to periexomeno kane to eisagogi
sth lista eggrafon

##

print('\nNumber of documents in our collection: \n',len(doc_files))#1209
documents
#print('\nPrinting the first document: \n',doc_files[0])
input('Waiting...')
tokens =[]#tokens as a list per document

for i in range(len(doc_files)):
   tokens.append(doc_files[i].lower().split())#apo ta keimena pare kathe
keimeno ksexorista kai spasto se lekseis. valta os i-osto stixio
   #th lista ayth ton lekseon sta tokens. diladi px. keimeno 5 5-th stixio ton
tokens einai h lista me tis lekseis tou keimenou 5

tokens_all = []#all tokens in the collection

for i in range(len(tokens)):
   tokens_all = tokens_all + tokens[i]#enono ola ta tokens se mia eniaia lista
#print(tokens)

#print('\nAll tokens in the collection: \n',tokens_all)
terms = list(set(tokens_all))#perno tis monadikes times ton lekseon apo ayth th
lista kai etsi exo tous orous to leksilogio mou
#print('\nUnique terms gathered from the document collection\n',terms)
print('\nNumber of terms : \n',len(terms))

###

#
words = []
stop_words = ["a", "an", "the", "and", "but", "or", "because", "as", "until",
"while", "of", "at", "by", "for", "with", "about", "against", "between",
"into", "through", "during", "before", "after", "above", "below", "to", "from",
"up", "down", "in", "out", "on", "off", "over", "under", "again", "further",
"then", "once", "here", "there", "when", "where", "why", "how", "all", "any",
"both", "each", "few", "more", "most", "other", "some", "such", "no", "nor",
```

```
"not", "only", "own", "same", "so", "than", "too", "very", "can", "will",
"just"]
#
for word in terms:
   if word not in stop_words:
      words.append(word)
######-----INVERTED INDEX-------#####

inverted_index = {}
for i,doc in enumerate(doc_files): #gia kathe keimeno arxika pare to idio to
keimeno kai th thesh tou sth lista keimenon
   for word in doc.lower().split():#gia kathe leksi sto ekastote keimeno
       if word in inverted_index:#
           inverted_index[word].add(i)#an einai hdh sth lista prosthese to id
tou keimenou pou vrisketai h leksi
       else:
           inverted_index[word] = {i}#an den einai sth lista o oros tote
prosthese thn emfanisi tou sto keimeno doc
input('WAIT')

#-------------------######---------

#pinakas pou krata ton arithmo ton keimenon pou vrisketai mia lexh
Ni = pd.Series(np.zeros(len(words)),index=words)
for word in words:
   Ni[word] = len(inverted_index[word])


idf = pd.Series(np.zeros(len(words)),index=words)#ypologismos anastrofis
syxnotitas emfanisis
for word in words:
   idf[word] = np.log2(len(doc_files)/(Ni[word]))
   #idf[word] = 1 #monadiaio
   #idf[word] = np.log2(1+len(doc_files)/(Ni[word])) -->aplh logarithmiki
kanonikopoihsh
   #idf[word] = np.log2(1+(Ni.max())/(Ni[word]))
#print(Ni.head()) #-ok
###------implementation with pandas to test speed------######
tf = pd.DataFrame(np.zeros((len(doc_files), len(words))), columns=words)
#mitroo me tis sixnotites emfanisis ton oron ana keimeno
#print(df.head()) #-ok
#print(df.shape) #-ok
#print(df.tail()) #-ok
print(Ni['cardiopulmonary'])

word_split = []
#print(tf['cystic'][1208])
input('WAIT')
for i in range(len(doc_files)):
```

```
   word_split = doc_files[i].lower().split()
   for w in list(set(word_split) - set(stop_words)):#ousiastika me ayto gia to
sygkekrimeno keimeno asxoloumai mono me tis lekseis tou
   #afou oi ypoloipes lekseis pou den yparxoun sto keimeno exoun fij=0 kai den
tha katso na ypologiso - epanalipseis gia mhdenika. glitono
   #ypologistiki isxy-xrono klp
      #tf[w][i] = 1#afou asxoliomaste mono me tis lexeis tou keimenou tha einai
anagkastika 1 eno oi ypoloipes 0
      #gegonos pou exoume frontisei arxikopoiontas olo to mitroo se 0
      tf[w][i] = word_split.count(w)#aplh syxnothta emfanisis
      #tf[w][i] = 1 + np.log2(word_split.count(w))#aplh logarithmiki
kanonikopoihsh
      #tf[w][i] = 0.5 + 0.5*word_split.count(w)
print(idf['cystic'])
print(Ni['cystic'])
print(tf['cystic'][1208])
######-----calculating tf-idf-------####

weight = pd.DataFrame(np.zeros((len(doc_files), len(words))),
columns=words)#mitroo me ta varh ton oron ana keimeno
for i in range(len(doc_files)):
   word_split = doc_files[i].lower().split()
   for word in list(set(word_split) - set(stop_words)):
      weight[word][i] = tf[word][i] * idf[word]

print(Ni['cf'])
print(weight['cystic'][1208])

#####--------------calculating similarity with test queries-----##
query = "What are the effects of calcium on the physical properties of mucus
from CF patients?"
clean_query = re.sub('[^A-Za-z0-9]+', ' ',query)#afairo to erotimatiko kai
antikathisto me keno
clean_query = clean_query.lower().strip()#oti keno emeine eksaleifetai
#
print(clean_query)
#
clean_query_split = clean_query.split()
print(clean_query_split)
#
query_weight =
pd.Series(np.zeros(len(list(set(clean_query_split)))),index=list(set(clean_quer
y_split)))
#print(query_weight.shape)
frequencies=[]
#vrisko ton oro me th megaliteri syxnothta emfanisis sto erotima
for word in clean_query_split:
   frequencies.append(clean_query_split.count(word))
#print(max(frequencies))
```

```
max_freq = max(frequencies)
for word in list(set(clean_query_split)):#calculating wt,q
   if word in words:
      query_weight[word] = (0.5 +
0.5*clean_query_split.count(word)/max_freq)*idf[word]



input('STOP')
query_norm = np.linalg.norm(query_weight)#ypologizo thn l2 norma h alios to
metro tou dianysmatos tou erotimatos
doc_norms = np.linalg.norm(weight.values,axis=1)#afou kathe grammh einai ena
keimeno ypologizo th norma ton gramon tou
aligned_weight, aligned_query = weight.align(query_weight,
axis=1,fill_value=0,broadcast_axis=0)
inner_product = aligned_weight.dot(aligned_query)

#mitroo me ta esoterika ginomena kathe grammhs(keimenou) me to dianysma varon
tou query
cosine_similarity = (inner_product)/(query_norm * doc_norms)
#mitroou varon

print('\n10 most matched documents:\n ',cosine_similarity.nlargest(10))
print('\n10 most matched documents:\n ',cosine_similarity.nlargest(10).index)
```

Ερώτημα 3

Το notebook με τον κώδικα υλοποίησης του **colBERT** με τις αλλαγές που πρέπει να γίνουν στη συλλογή cystic fibrosis βρίσκεται στον σύνδεσμο: colBERT .
Δεν πρόλαβα να το υλοποιήσω εις πέρας αλλά φαίνεται μία αρχική προσπάθεια υλοποίησης στο παραπάνω σύνδεσμο.

Ερώτημα 4

```
import numpy as np
import pandas as pd
import os
import re
from scipy.stats import rankdata
from matplotlib import pyplot as plt
#vivliothikes
```

```
doc_files=[] #ta keimena os stixia listas
#

doc_path = os.getcwd() + "/Collection/docs" #kataskeyi tou monopatiou pou
vriskontai ta keimena

#
print(doc_path)#ektyposi aytou tou monopatiou
for file in sorted(os.listdir(doc_path)):#gia kathe string(noumero) ths listas
tou path
   file_path = os.path.join(doc_path,file)#ousiastika .../docs/px-1239 kanei
join to px ../docs me to 1239 kai vgenei: ../docs/1239
   #print(file_path)
   #print(type(file)) #- einai strings
   #print(int(file))
   if os.path.isfile(file_path):#an to ../docs/1239 px einai arxeio tote:
      f = open(file_path,'r')#anikse to ekastote arxeio se reading mode
      doc_files.append(f.read())#diavase to kai to periexomeno kane to eisagogi
sth lista eggrafon

##

print('\nNumber of documents in our collection: \n',len(doc_files))#1209
documents
#print('\nPrinting the first document: \n',doc_files[0])
input('Waiting...')
tokens =[]#tokens as a list per document

for i in range(len(doc_files)):
   tokens.append(doc_files[i].lower().split())#apo ta keimena pare kathe
keimeno ksexorista kai spasto se lekseis. valta os i-osto stixio
   #th lista ayth ton lekseon sta tokens. diladi px. keimeno 5 5-th stixio ton
tokens einai h lista me tis lekseis tou keimenou 5

tokens_all = []#all tokens in the collection

for i in range(len(tokens)):
   tokens_all = tokens_all + tokens[i]#enono ola ta tokens se mia eniaia lista
#print(tokens)

#print('\nAll tokens in the collection: \n',tokens_all)
terms = list(set(tokens_all))#perno tis monadikes times ton lekseon apo ayth th
lista kai etsi exo tous orous to leksilogio mou
#print('\nUnique terms gathered from the document collection\n',terms)
print('\nNumber of terms : \n',len(terms))

###

#
```

```
words = []
stop_words = ["a", "an", "the", "and", "but", "or", "because", "as", "until",
"while", "of", "at", "by", "for", "with", "about", "against", "between",
"into", "through", "during", "before", "after", "above", "below", "to", "from",
"up", "down", "in", "out", "on", "off", "over", "under", "again", "further",
"then", "once", "here", "there", "when", "where", "why", "how", "all", "any",
"both", "each", "few", "more", "most", "other", "some", "such", "no", "nor",
"not", "only", "own", "same", "so", "than", "too", "very", "can", "will",
"just"]
#
for word in terms:
   if word not in stop_words:
      words.append(word)
######-----INVERTED INDEX-------#####

inverted_index = {}
for i,doc in enumerate(doc_files): #gia kathe keimeno arxika pare to idio to
keimeno kai th thesh tou sth lista keimenon
   for word in doc.lower().split():#gia kathe leksi sto ekastote keimeno
       if word in inverted_index:#
           inverted_index[word].add(i)#an einai hdh sth lista prosthese to id
tou keimenou pou vrisketai h leksi
       else:
           inverted_index[word] = {i}#an den einai sth lista o oros tote
prosthese thn emfanisi tou sto keimeno doc
input('WAIT')

#--------------------######---------

#pinakas pou krata ton arithmo ton keimenon pou vrisketai mia lexh
Ni = pd.Series(np.zeros(len(words)),index=words)
for word in words:
   Ni[word] = len(inverted_index[word])


idf = pd.Series(np.zeros(len(words)),index=words)#ypologismos anastrofis
syxnotitas emfanisis
for word in words:
   idf[word] = np.log2(len(doc_files)/(Ni[word]))
   #idf[word] = 1 #monadiaio
   #idf[word] = np.log2(1+len(doc_files)/(Ni[word])) -->aplh logarithmiki
kanonikopoihsh
   #idf[word] = np.log2(1+(Ni.max())/(Ni[word]))
#print(Ni.head()) #-ok
###------implementation with pandas to test speed------######
tf = pd.DataFrame(np.zeros((len(doc_files), len(words))), columns=words)
#mitroo me tis sixnotites emfanisis ton oron ana keimeno
#print(df.head()) #-ok
#print(df.shape) #-ok
```

```
#print(df.tail()) #-ok
print(Ni['cardiopulmonary'])

word_split = []
#print(tf['cystic'][1208])
input('WAIT')
for i in range(len(doc_files)):
   word_split = doc_files[i].lower().split()
   for w in list(set(word_split) - set(stop_words)):#ousiastika me ayto gia to
sygkekrimeno keimeno asxoloumai mono me tis lekseis tou
   #afou oi ypoloipes lekseis pou den yparxoun sto keimeno exoun fij=0 kai den
tha katso na ypologiso - epanalipseis gia mhdenika. glitono
   #ypologistiki isxy-xrono klp
      #tf[w][i] = 1#afou asxoliomaste mono me tis lexeis tou keimenou tha einai
anagkastika 1 eno oi ypoloipes 0
      #gegonos pou exoume frontisei arxikopoiontas olo to mitroo se 0
      tf[w][i] = word_split.count(w)#aplh syxnothta emfanisis
      #tf[w][i] = 1 + np.log2(word_split.count(w))#aplh logarithmiki
kanonikopoihsh
      #tf[w][i] = 0.5 + 0.5*word_split.count(w)
print(idf['cystic'])
print(Ni['cystic'])
print(tf['cystic'][1208])
######-----calculating tf-idf-------####

weight = pd.DataFrame(np.zeros((len(doc_files), len(words))),
columns=words)#mitroo me ta varh ton oron ana keimeno
for i in range(len(doc_files)):
   word_split = doc_files[i].lower().split()
   for word in list(set(word_split) - set(stop_words)):
      weight[word][i] = tf[word][i] * idf[word]

print(Ni['cf'])
print(weight['cystic'][1208])

#####--------------calculating similarity with test queries-----##
query = "What are the effects of calcium on the physical properties of mucus
from CF patients?"
clean_query = re.sub('[^A-Za-z0-9]+', ' ',query)#afairo to erotimatiko kai
antikathisto me keno
clean_query = clean_query.lower().strip()#oti keno emeine eksaleifetai
#
print(clean_query)
#
clean_query_split = clean_query.split()
print(clean_query_split)
#
```

```
query_weight =
pd.Series(np.zeros(len(list(set(clean_query_split)))),index=list(set(clean_quer
y_split)))
#print(query_weight.shape)
frequencies=[]
#vrisko ton oro me th megaliteri syxnothta emfanisis sto erotima
for word in clean_query_split:
   frequencies.append(clean_query_split.count(word))
#print(max(frequencies))
max_freq = max(frequencies)
for word in list(set(clean_query_split)):#calculating wt,q
   if word in words:
      query_weight[word] = (0.5 +
0.5*clean_query_split.count(word)/max_freq)*idf[word]



input('STOP')
query_norm = np.linalg.norm(query_weight)#ypologizo thn l2 norma h alios to
metro tou dianysmatos tou erotimatos
doc_norms = np.linalg.norm(weight.values,axis=1)#afou kathe grammh einai ena
keimeno ypologizo th norma ton gramon tou
aligned_weight, aligned_query = weight.align(query_weight,
axis=1,fill_value=0,broadcast_axis=0)
inner_product = aligned_weight.dot(aligned_query)

#mitroo me ta esoterika ginomena kathe grammhs(keimenou) me to dianysma varon
tou query
cosine_similarity = (inner_product)/(query_norm * doc_norms)
#mitroou varon

print('\n10 most matched documents:\n ',cosine_similarity.nlargest(10))
print('\n10 most matched documents:\n ',cosine_similarity.nlargest(10).index)
Ids = cosine_similarity.nlargest(10).index
Ids = np.array(Ids)
print(Ids)
#####

relevant_docs = [721,509,473]#synolo sxetikon me to erotima keimenon
relevant_docs = np.array(relevant_docs)
R = len(relevant_docs) #plithos sxetikon me to query keimenon-synolo R
score = cosine_similarity.nlargest(10).values#pinakas me ta score ton keimenon
sto query tha borouse na einai to cosine simularity
score = np.array(score)
sorted_indexes = np.flip(np.argsort(score))#me thn flip pernoume se fthinousa
seira ta indexes tou score: max score-...min score
###
print(sorted_indexes)
#
```

```
Ids_sorted = Ids[sorted_indexes]#taxinomimena keimena vash tou score
#sprint(Ids_sorted)
rank = rankdata(score,'max')
ranks = len(rank) - rank + 1
#tha paei me vash to score
print(ranks)

###----pinakas pou deixnei an to i osto keimeno anikei sta sxetika---###
relevant_or_not = np.zeros(len(Ids),dtype=bool)
#
for i in range(len(Ids)):
   if Ids_sorted[i] in relevant_docs:
       relevant_or_not[i] = True
print(relevant_or_not)
#calculating precision and recall#

#
sorted_rank = np.arange(1,len(Ids)+1)#h katataxh kathe keimena
#
Precision = np.zeros(len(Ids))#akriveia gia to query
#
Recall = np.zeros(len(Ids))#anaklisi gia to query
#
for i in range(len(Ids)):
   r = sorted_rank[i]#ousiastika posa keimena exo anaktisei mexri stigmhs
   Precision[i] = sum(relevant_or_not[:r])/r #plithos sxetikon keimenon pou
exoun anaktithei eos tora/posa exo anaktisei
   #synolika mexri tora
   Recall[i] = sum(relevant_or_not[:r])/R #plithos sxetikon keimenon pou exo
anaktithei mexri stigmis/plithossxetikon me
# to query

#
##------ftiaxno tous synolikous pinakes-#######

table_of_results = pd.DataFrame({'Doc
id':Ids,'Score':score,'Rank':ranks})#pinakas me ta apotelesmata tou query to
table_of_metrics = pd.DataFrame({'Rank':sorted_rank,'Doc
id':Ids_sorted,'Relevant':relevant_or_not,'Precision':Precision,'Recall':Recall
})#

#pinakas ton keimenon taxinomimena vash tou rank tous(vash tou score) mazi me
tis times ton precision-recall
print(table_of_results)
print(table_of_metrics)
##----PLOTTING-----##
def Precision_Recall_Graph(precision,recall):
   plt.plot(recall,precision,marker='x',linestyle='--')
   plt.xlabel('Recall')
```

```
   plt.ylabel('Precision')
   plt.title('Precision vs Recall')
   plt.grid()
   plt.show()
Precision_Recall_Graph(Precision,Recall)
```