|  |
| --- |
|  |
| page1image29306176  Τμήμα Μηχανικών Η/Υ και Πληροφορικής  Πανεπιστήμιο Πατρών  Πολυτεχνική Σχολή  Τομέας Λογικού των Υπολογιστών  Διδάσκων: Χρήστος Μακρής  Ακαδημαϊκό Έτος: 2023 – 2024  Ημ / νία Παράδοσης: 28 / 01 / 2024 |

Ανάκτηση Πληροφορίας

Επιλεγόμενο Μάθημα – CEID\_NE5597

Εργαστηριακή Άσκηση

Χειμερινό Εξάμηνο 2023

Χρυσαυγή Πατέλη | 1084513 | up1084513@ac.upatras.gr

Μηλτιάδης Μαντές | 1084661 | up1084661@ac.upatras.gr

ΠΕΡΙΕΧΟΜΕΝΑ

**1 ΕΙΣΑΓΩΓΗ 1.1** Ερώτημα 1 …………………………………………………………………………………….. 3 **1.2** Ερώτημα 2 …………………………………………………………………………………….. 4 **1.3** Ερώτημα 3 …………………………………………………………………………………….. 7 **1.4** Ερώτημα 4 …………………………………………………………………………………….10 **1.5** Περιβάλλον Υλοποίησης και Βιβλιοθήκες ……………………………………………….12

**2 ΥΛΟΠΟΙΗΣΗ 2.1** Ερώτημα 1 …………………………………………………………………………………….14 **2.2** Ερώτημα 2 …………………………………………………………………………………….14  **2.3** Ερώτημα 3 …………………………………………………………………………………….14 **2.4** Ερώτημα 4 …………………………………………………………………………………….14

**3 ΑΠΟΤΕΛΕΣΜΑΤΑ – ΠΑΡΑΤΗΡΗΣΕΙΣ 3.1** Ερώτημα 1.…………………………………………………………………………………….14 **3.2** Ερώτημα 2 …………………………………………………………………………………….14  **3.3** Ερώτημα 3 …………………………………………………………………………………….14 **3.4** Ερώτημα 4 …………………………………………………………………………………….14

**4 ΑΝΑΦΟΡΕΣ**

**5 ΠΑΡΑΡΤΗΜΑ**

**1 ΕΙΣΑΓΩΓΗ**

* 1. **Ερώτημα 1**

Στο συγκεκριμένο ερώτημα μας ζητείται να υλοποιήσουμε την δομή του **Ανεστραμμένου Ευρετηρίου** (Ιnverted Ιndex) πάνω σε μια συλλογή 1209 κειμένων.

Το ανεστραμμένο ευρετήριο είναι μια δομή δεδομένων, η οποία χρησιμοποιείται σε συστήματα ανάκτησης πληροφοριών, μηχανές αναζήτησης και γενικά σε συστήματα που είναι απαραίτητη η αναζήτηση κειμένων. Στόχος του είναι η γρήγορη αναζήτηση και εύρεση όλων των εγγράφων που περιέχουν έναν συγκεκριμένο όρο ή φράση.

Το αναστραμμένο ευρετήριο αποτελείται από δυο μέρη, το λεξικό που περιέχει τους διακριτούς όρους όλης της συλλογής και για κάθε όρο του λεξικού τις λίστες εμφανίσεων, οι οποίες περιέχουν πληροφορίες για την εμφάνιση των λέξεων στα κείμενα. Οι λίστες αυτές ονομάζονται ανεστραμμένες λίστες. Η πιο απλή μορφή του για κάθε όρο περιέχει το πλήθος των εγγράφων που περιέχεται και τα id των εγγράφων. Υπάρχουν και άλλες εκδοχές, όπου στην ανεστραμμένη λίστα μαζί με το id των κειμένων που περιέχουν την λέξη υπάρχουν και άλλες πληροφορίες σχετικά με την λέξη, η εκδοχή που χρησιμοποιήσαμε εμείς περιέχει την συχνότητα της λέξης μέσα στο συγκεκριμένο κείμενο όπως φαίνεται και στην παρακάτω εικόνα.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, οθόνη, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

**Εικόνα 1**: Προσομοίωση Ανεστραμμένου Ευρετηρίου

Το πλεονέκτημα του **Ανεστραμμένου Ευρετηρίου** είναι ότι επιτρέπει γρήγορες, αποδοτικές αναζητήσεις ερωτημάτων και ανάλογα την μορφή μπορεί να απαντήσει σε διάφορα είδη ερωτημάτων (ερωτήματα πολλαπλών λέξεων, μονή λέξης, τομή λίστας, σύνθετα ερωτήματα, ερωτήματα φράσης και εγγύτητας, boolean). Ωστόσο έχουν και μειονεκτήματα, καθώς έχουν υψηλό κόστος συντήρησης, αποθήκευσης και απαιτούν πολύ χώρο στην μνήμη για την αποθήκευση και την ενημέρωσή τους. Γι’ αυτό και συχνά εφαρμόζονται τεχνικές συμπίεσης (κωδικοποίηση δέλτα, κωδικοποίηση γάμμα, η κωδικοποίηση μεταβλητών byte) πάνω στο **Aνεστραμμένο Eυρετήριο**, για περαιτέρω μείωση του απαιτούμενου χώρου.

**1.2 Ερώτημα 2**

Στο συγκεκριμένο ερώτημα μας ζητείται να υλοποιήσουμε το **Μοντέλο Διανυσματικού Χώρου** (Vector Space Model) πάνω στο ανεστραμμένο αρχείο που εξάγαμε στο προηγούμενο ερώτημα.

Το **Μοντέλο Διανυσματικού Χώρου** χρησιμοποιεί την έννοια των διανυσμάτων στον πολυδιάστατο χώρο προκειμένου να αναπαραστήσει αλγεβρικά την φυσική γλώσσα των κειμένων της συλλογής εγγράφων. Πιο συγκεκριμένα, το σκεπτικό πίσω από αυτό το σύστημα ανάκτησης πληροφορίας έγκειται στην ευκολία εύρεσης κοινών όρων (terms) ανάμεσα σε κείμενα και ερωτήματα χρήστη (queries) αν τα ζυγισμένα διανύσματα που τα αναπαριστούν τοποθετηθούν σε διανυσματικό χώρο *V*.

Σε μια πιο γενική περιγραφή του μοντέλου ισχύουν τα εξής:

1. Έστω {t1, t2, …, tn} οι όροι ενός κειμένου di, i = 1, 2, …, m από την συλλογή μας. Υποθέτοντας χωρίς βλάβη γενικότητας ότι για κάθε διακριτό όρο ti υπάρχει στο χώρο ένα μοναδιαίο ορθοκανονικό διάνυσμα **ti** στο οποίο αντιστοιχίζεται, τότε ο διανυσματικός χώρος *V* θα είναι το span{ti} και έτσι το διάνυσμα **di** του κειμένου μπορεί να δοθεί ως γραμμικός συνδυασμός των **ti**.

**di =**

1. Έστω {T1, T2, …, Tn} οι όροι ενός ερωτήματος, οι οποίοι σχετίζονται με το σύνολο {t1, t2, …, tn}. Για κάθε ερώτημα q όμοια προκύπτει ότι το αντίστοιχο του διάνυσμα **q** του διανυσματικού χώρου *V* θα είναι πάλι γραμμικός συνδυασμός των διανυσμάτων **Τi**.

Εικόνα που περιέχει γραμμή

Περιγραφή που δημιουργήθηκε αυτόματα**q =**

Κάθε διακριτός όρος σε κάθε περίπτωση αντιπροσωπεύει μια διάσταση του χώρου μας. Συνεπώς, στόχος μας είναι στο χώρο *V* για κάθε διάνυσμα **q** να υπολογίσουμε την απόστασή του από κάθε διάνυσμα **di**. Για να γίνει αυτό επιδιώκουμε να προσεγγίσουμε τη γωνία που σχηματίζεται ανάμεσά τους και πιο συγκεκριμένα το συνημίτονό της, cos(θ), το οποίο δίνεται από το εσωτερικό γινόμενο.

cos(θ) =

Στο τέλος, χρησιμοποιούμε ως κριτήριο ταξινόμησης τη τιμή του συνημίτονου που προκύπτει προκειμένου να διατάξουμε τα κείμενα μας με βάση τη σχετικότητα τους ως προς κάθε ερώτημα χρήστη.

Ειδικότερα τώρα, στη δικιά μας υλοποίηση επιλέγουμε για απλοποίηση τα διανύσματα **di**, **qj** να έχουν ως διάσταση το πλήθος των διακριτών όρων του κειμένου και του ερωτήματος αντίστοιχα και κάθε γραμμή του διανύσματος **di** και **qj** να περιέχει μια boolean τιμή, με το 0 να συμβολίζει την μη – ύπαρξη του συγκεκριμένου όρου στο διάνυσμα κειμένου και το 1 να συμβολίζει προφανώς την ύπαρξή του.

Εικόνα που περιέχει κείμενο, γραμμή, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματαΤώρα, για την καλύτερη εφαρμογή του **Vector Space Model** είναι απαραίτητη η αξιοποίηση της τεχνικής του *term – weighting*, καθώς οι διαφορετικοί όροι μπορεί να έχουν διαφορετική σημασία και άρα και επίδραση στα συμφραζόμενα του κειμένου. Αρχικά, όσον αφορά το local term – weighting, επιλέγουμε μια από τις ακόλουθες τεχνικές υπολογισμού του term frequency (TF), καθώς μας αφορά μόνο η συχνότητα με την οποία εμφανίζονται οι όροι σε κάθε κείμενο ανεξάρτητα, την οποία και έχουμε αποθηκευμένη στο ανεστραμμένο αρχείο.

**Εικόνα 3**: Τρόποι Υπολογισμού Term Frequency

Η πρώτη μας σκέψη είναι να επιλέξουμε την *Απλή Συχνότητα Εμφάνισης*, η οποία υπολογίζει πόσες φορές εμφανίζεται ένας όρος στο κείμενο. Άλλωστε, όσο περισσότερες φορές εμφανίζεται ένας όρος στο κείμενο, τόσο πιο πιθανό είναι το κείμενο αυτό να είναι σχετικό με το ερώτημά μας. Ωστόσο, το μειονέκτημα αυτού του σκεπτικού είναι κυρίως ότι ευνοεί υπερβολικά τους όρους που εμφανίζονται με αυξημένη συχνότητα, καθώς η μεγάλη συχνότητα εμφάνισης δεν μας εξασφαλίζει υποχρεωτικά ότι οι όροι αυτοί θα είναι και πιο σημαντικοί από κάποιους όρους που εμφανίζονται πιο σπάνια. Επίσης, η χρήση δυαδικών βαρών είναι και αυτή αρκετά περιοριστική. Γι’ αυτό καταλήγουμε στην *Απλή Λογαριθμική Κανονικοποίηση*, η οποία προσπαθεί να αντισταθμίσει την επίδραση της *Απλής Συχνότητας Εμφάνισης* μειώνοντας μέσω του λογαρίθμου την επίδραση πολύ μεγάλων διαφορών στις συχνότητες εμφάνισης, έτσι ώστε να έχουμε δικαιότερη απόδοση βαρών στο τελικό διάνυσμα **di**.

Εικόνα που περιέχει κείμενο, γραμμή, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματαΈπειτα, όσον αφορά το global term – weighting, θέλουμε να εξετάσουμε πώς η κατανομή και η συχνότητα εμφάνισης ενός όρου σε όλα τα κείμενα της συλλογής επηρεάζει το πόσο σημαντικός είναι. Για το λόγο αυτό επιλέγουμε την αποδοτικότερη από τις πιο κάτω τεχνικές για τον υπολογισμό του Inverted Document Frequency (IDF):

**Εικόνα 4**: Τρόποι Υπολογισμού Inverted Document Frequency

Εδώ επιλέγουμε την τεχνική της Απλής Ανάστροφης Συχνότητας Εμφάνισης, η οποία «λογαριθμίζει» την αναλογία των κειμένων μέσα στη συλλογή μας που περιέχουν ένα συγκεκριμένο όρο. Η τιμή της αυξάνεται όσο το πλήθος των κειμένων στο οποίο εντοπίζεται αυτός ο όρος μειώνεται και έτσι ψηλή τιμή σημαίνει συνήθως ότι ένας όρος εμφανίζεται στο συγκεκριμένο έγγραφο με μεγαλύτερη συχνότητα από το συνηθισμένο.

Έχοντας ολοκληρώσει αυτή τη διαδικασία, πολλαπλασιάζουμε κάθε συντεταγμένη των διανυσμάτων μας με τα τελικά βάρη και τα διανύσματα που προκύπτουν θα είναι:

**di** = [di1, di2, di3, …, dik] με dik να συμβολίζει το βάρος του k – όρου στο έγγραφο i **qj** =[qj1, qj2, qj3, …, qjk] με qjk να συμβολίζει το βάρος του k – όρου στο ερώτημα j

και συνεπώς το συνημίτονο της γωνίας που θα σχηματίζουν θα δίνεται από τη σχέση:

cos(**di**, **qj**) =

Σημειώνουμε εδώ ότι τα βάρη των όρων που δεν εντοπίζονται μέσα σε κάποιο κείμενο θα παραμείνουν 0 λόγω του πολλαπλασιασμού με τα βάρη tf-idf όπως προείπαμε.

Μια συνηθισμένη τεχνική αποτελεί επίσης η κανονικοποίηση του μήκους των διανυσμάτων **di**, **qi** (length – normalization) διαιρώντας κάθε συντεταγμένη τους με την L2 νόρμα τους, προκειμένου και να γίνουν και αυτά μοναδιαία όπως αυτά που ορίζουν τη βάση του χώρου *V* στον οποίον ανήκουν. Χάριν απλότητας επιλέγουμε να παραλείψουμε αυτό το βήμα, καθώς η χρησιμότητά του είναι κυρίως να κάνει τα βάρη των πολύ μικρών ή πολύ μεγάλων κειμένων ακόμα πιο συγκρίσιμα.

**1.3 Ερώτημα 3**

**1.4 Ερώτημα 4**

Στόχος μας είναι να συγκρίνουμε την απόδοση των δύο μοντέλων χρησιμοποιώντας ένα πλήθος διαφορετικών μετρικών, προκειμένου να έχουμε μεγαλύτερη αξιοπιστία στη τελική μας σύγκριση. Επιλέγουμε να χρησιμοποιήσουμε μετρικές αξιολόγησης σχετικότητας και πιο συγκεκριμένα τις πιο κάτω μετρικές:

1. **Precision – Recall Curve**

**1.1** Ηακρίβεια προσδιορίζει την ικανότητα του μοντέλου ανάκτησης πληροφορίας που διαθέτουμε να επιστρέφει τα κορυφαία σχετικά κείμενα με βάση το score τους κατά τον έλεγχο σχετικότητας. Στη προκειμένη περίπτωση, τα score της τελικής κατάταξης των ανακτημένων εγγράφων από τη συλλογή προκύπτουν από την εφαρμογή των **Vector Space Model** και **ColBERT**. Με άλλα λόγια, εκφράζει το ποσοστό των ανακτημένων κειμένων, τα οποία είναι όντως σχετικά με το ερώτημα του χρήστη. Το πλήθος και ο βαθμός σχετικότητας των σχετικών αυτών κειμένων καθορίζεται από κάποιους ειδικούς, οι οποίοι τα κατατάσσουν αυτά σε κλίμακα εύρους 0 – 2, με 0 να σημαίνει «καθόλου σχετικό» και 2 «αρκετά σχετικό».

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, κύκλος

Περιγραφή που δημιουργήθηκε αυτόματα**1.2** Η ανάκληση εκφράζει την ικανότητα του μοντέλου μας να επιστρέψει όλη τη συλλογή με τα σχετικά κείμενα, την οποία αναφέραμε ακριβώς πιο πάνω, δηλαδή το ποσοστό των σχετικών κειμένων τα οποία απαντάνε στα ερωτήματα του χρήστη.

**Εικόνα** : Σχέση ανάμεσα σε Ανάκληση και Ακρίβεια

1. **Mean Average Precision (MAP)**

Η Μέση Ακρίβεια (AP) δίνει έμφαση στην τοποθέτηση των πιο σχετικών εγγράφων στις πιο υψηλές θέσεις της κατάταξης. Πιο συγκεκριμένα, υπολογίζουμε το μέσο όρο των τιμών του precision από όλα τα σχετικά κείμενα της συλλογής για κάθε ερώτημα χρήστη. Συνεπώς, για τον υπολογισμό του MAP θα αξιοποιήσουμε την έννοια της Μέσης Ακρίβειας (AP), καθώς τώρα υπολογίζουμε τη μέση τιμή όλων των τιμών του AP από όλα τα ερωτήματα χρήστη. Δίνεται από τη σχέση:

**MAP =**

**Qj**: ο αριθμός των σχετικών κειμένων για το ερώτημα j

**N**: ο αριθμός των ερωτημάτων

**P(doci)**: η τιμή της ακρίβειας του σχετικού κειμένου i

1. **precision@k**

Υπάρχουν περιπτώσεις στις οποίες είναι απαραίτητο να συγκρίνουμε την απόδοση των μοντέλων ανάκτησης για ατομικές πληροφοριακές ανάγκες. Ο λόγος που μας οδηγεί σε αυτή την απόφαση είναι ότι η χρήση Mέσων Tιμών (MAP) που προκύπτουν από την εκτέλεση διαφόρων ερωτημάτων μπορεί να αποκρύπτει σημαντικές ανωμαλίες στο μοντέλο ανάκτησης που εξετάζεται, οπότε πρέπει για μεγαλύτερη ασφάλεια στην εκτίμηση απόδοσης να μελετήσουμε αν το ένα μοντέλο είναι καλύτερο από το άλλο για κάθε μία από τις πρότυπες πληροφοριακές ανάγκες. Στις περιπτώσεις αυτές, μόνο μία τιμή precision υπολογίζεται για κάθε ερώτημα και η τιμή αυτή θεωρούμε ότι λειτουργεί ως σύνοψη του συνολικού Διαγράμματος Ακρίβειας – Ανάκλησης. Συνήθως αυτή η τιμή είναι η ακρίβεια σε κάποιο προκαθορισμένο επίπεδο k στην ουρά ανάκλησης. Δίνεται από τη σχέση:

**precision@k =**

**rj** = 1 αν το κείμενο στη θέση j είναι σχετικό αλλιώς 0.

Οι μετρικές 2, 3 μας επιστρέφουν μια μοναδική τιμή, οπότε για να συγκρίνουμε την απόδοση των δύο μοντέλων αρκεί να συγκρίνουμε τις τιμές των **MAP** και **precision@k**. Όσο μεγαλύτερες είναι αυτές οι τιμές, τόσο πιο αποδοτικά ανακτά το μοντέλο μας τα σχετικά έγγραφα. Για τη 1η περίπτωση των γραφικών precision – recall για να συγκρίνουμε τις αποδόσεις, μια καλή τεχνική θα ήταν να υπολογίσουμε το εμβαδόν της περιοχής που περικλείεται ανάμεσα στις γραφικές και στον άξονα x. Όμοια με πριν, όσο μεγαλύτερη είναι η τιμή του εμβαδού που υπολογίζουμε, τόσο πιο αποδοτικό είναι το μοντέλο μας. Ωστόσο, επειδή υπάρχει περίπτωση για κάποια ερωτήματα να λειτουργεί καλύτερα το ένα μοντέλο (**VSM**) και για κάποια το άλλο (**ColBERT**), υπολογίζουμε τη μέση τιμή των εμβαδών για όλα τα ερωτήματα, προκειμένου να έχουμε μια συνολική εικόνα για το ποιο από τα δύο μοντέλα λειτουργεί καλύτερα σε γενικές γραμμές.

* 1. **Περιβάλλον Υλοποίησης και Βιβλιοθήκες**

Όλα τα παραπάνω τα οποία εν συντομία αναπτύχθηκαν θεωρητικά, καλούμαστε τώρα να τα υλοποιήσουμε και σε κώδικα. Επιλέγουμε να χρησιμοποιήσουμε ως περιβάλλον ανάπτυξης αυτό της *PyCharm*, το οποίο διαθέτουμε ήδη εγκατεστημένο από το *JetBrains*. Η έκδοση που χρησιμοποιούμε είναι η 2023.3.2 και επιπλέον οι βιβλιοθήκες που εισάγουμε κατά την εκτέλεση του προγράμματος είναι οι εξής:

1. **numpy**

Από αυτή τη βιβλιοθήκη αξιοποιούμε τις συναρτήσεις np.linalg.norm( ) και np.dot( ), προκειμένου να εφαρμόσουμε στο Ερώτημα 2 πάνω στα dataframes μας την πράξη της Ευκλείδιας νόρμας (L2) και του εσωτερικού γινομένου στοιχείο προς στοιχείο.

1. **pandas**

Από αυτή τη βιβλιοθήκη αξιοποιούμε την pd.Dataframe( ), προκειμένου να κατασκευάσουμε όλα τα dataframes στα οποία θα αποθηκεύουμε τις διάφορες τιμές που προκύπτουν κατά τη διαδικασία παραγωγής των τελικών scores του cosine similarity στο Ερώτημα 2.

1. **csv**

Από αυτή τη βιβλιοθήκη αξιοποιούμε τη συνάρτηση csv.reader(csvfile, dialect='excel', \*\*fmtparams) προκειμένου να διαβάσουμε στο Ερώτημα 4 τα ανακτηθέντα έγγραφα που μας επιστρέφει το μοντέλο **ColBERT**, τα οποία έχουμε μεταβιβάσει σε ένα αρχείο csv από το περιβάλλον του *Google Collab* σε αυτό της *PyCharm*.

1. **math**

Από αυτή τη βιβλιοθήκη αξιοποιούμε τη συνάρτηση math.log10(*x*) για τον υπολογισμό του δεκαδικού λογαρίθμου στις συχνότητες TF και IDF όταν καθορίζουμε τα βάρη των διανυσμάτων μας στο Ερώτημα 2.

1. **pyplot**

Από αυτή τη βιβλιοθήκη αξιοποιούμε όλες τις απαραίτητες συναρτήσεις για το Ερώτημα 4 προκειμένου να χαράξουμε σωστά τις γραφικές παραστάσεις των δύο μοντέλων στη μετρική Precision – Recall. Τέτοιες συναρτήσεις είναι οι: plt.figure( ), …

1. **collections**

Από αυτή τη βιβλιοθήκη εισάγουμε τη κλάση class collections.defaultdict(default\_factory=None, /[, ...]), η οποία είναι ιδιαίτερα χρήσιμη για την κατασκευή του **Ανεστραμμένου Ευρετηρίου** στο Ερώτημα 1, καθώς μας επιτρέπει να προσθέτουμε αντικείμενα (δηλαδή δυάδες της μορφής [ID κειμένου, πλήθος εμφανίσεων όρου]) στη λίστα που αντιστοιχεί στο εκάστοτε key (δηλαδή κάθε μοναδικός όρος) χωρίς να χρειάζεται να ελέγξουμε αν αυτό υπάρχει ήδη στο λεξικό.

1. **os**

Από αυτή τη βιβλιοθήκη αξιοποιούμε τη συνάρτηση os.chdir(path) προκειμένου να μεταβούμε στο Ερώτημα 1 στο φάκελο docs που περιέχει τα 1209 κείμενα της συλλογής και στη συνέχεια να ακολουθήσουμε τη διαδικασία που περιγράφεται στην υποενότητα **2.1** για την κατασκευή του **Ανεστραμμένου** **Ευρετηρίου**.

1. **ast**

Από αυτή τη βιβλιοθήκη αξιοποιούμε τη συνάρτηση ast.literal\_eval(node\_or\_string), καθώς σε κάθε ερώτημα στο αρχείο csv όπου έχουμε αποθηκευμένα τα αποτελέσματα του **ColBERT** αντιστοιχεί ένα αλφαριθμητικό το οποίο αντιπροσωπεύει τη λίστα με τα ταξινομημένα ανακτηθέντα κείμενα. Συνεπώς, είναι απαραίτητο κάθε ένα από τα αλφαριθμητικά αυτά να μετατραπούν σε πραγματικές λίστες με περιεχόμενο ακεραίους αριθμούς, δηλαδή τα IDs των κειμένων.

1. **re**

Από αυτή τη βιβλιοθήκη αξιοποιούμε τη συνάρτηση

**2 ΥΛΟΠΟΙΗΣΗ**

**2.1 Ερώτημα 1**

Αρχικά ορίζουμε-αρχικοποιούμε 4 μεταβλητές που θα χρησιμοποιηθούν στην συνέχεια.

1. **inverted\_index**: είναι ένα defaultdict, το οποίο αρχικοποιείται ως κενή λίστα. Το deafultdict είναι μια υποκλάση της κλάσης dictionary και επιστρέφει ένα αντικείμενο που μοιάζει με λεξικό. Χρησιμοποιείται ώστε να δημιουργήσει το ανεστραμμένο ευρετήριο, όπου κάθε λέξη θα συσχετίζεται με μια λίστα πλειάδων που θα περιέχουν τα id των κειμένων και τον αριθμό εμφανίσεων της λέξης στο αντίστοιχο κείμενο.
2. **document\_count:** είναι dictionary και θα χρησιμοποιηθεί για να αποθηκεύσουμε τα id των κειμένων που εμφανίζεται κάθε λέξη, για να υπολογίζεται το πλήθος των κειμένων σε επόμενα ερωτήματα.
3. **path:** μεταβλητή η οποία περιέχει την διαδρομή που βρίσκεται ο φάκελος με την συλλογή των κειμένων (docs).

*# defaultdict to store the inverted index*inverted\_index = defaultdict(list)  
  
*# Dictionary to store the documents each word appears in*document\_count = {}  
  
path = (r"C:\Users\chryssa\_pat\PycharmProjects\pythonProject\docs")  
os.chdir(path)

Στην συνέχεια ξεκινάει η διαδικασία δημιουργίας του ανεστραμμένου ευρετηρίου. Η διαδικασία αρχίζει με μια επανάληψη που διατρέχει κάθε αρχείο μέσα στο φάκελο docs με την χρήση της εντολής **os.listdir(path)** και αποθηκεύουμε στην μεταβλητή file\_path την διαδρομή για το συγκεκριμένο αρχείο (**os.path.join(path, file)**). Ανοίγουμε, διαβάζουμε το αρχείο και στη μεταβλητή dictionary αποθηκεύονται οι λέξεις που περιέχει το συγκεκριμένο κείμενο. Επίσης, δημιουργούμε ένα ακόμη λεξικό (count) που θα χρησιμοποιηθεί για να αποθηκεύουμε την συχνότητα εμφάνισης κάθε λέξης στο τρέχων κείμενο. Ύστερα, διατρέχουμε κάθε λέξη του dictionary και αν βρεθεί ανανεώνουμε το count για την συγκεκριμένη λέξη. Επιπλέον, για κάθε λέξη που υπάρχει στο τρέχων κείμενο εισάγουμε στο inverted\_index το id(είναι το όνομα του αρχείου) και την συχνότητα εμφάνισης (count) αν η λέξη υπάρχει ήδη, αλλιώς εισάγεται η λέξη μαζί με τις αντίστοιχες πληροφορίες. Τέλος, στον ίδιο βρόγχο επανάληψης ανανεώνουμε το document\_count για κάθε λέξη του κειμένου αν υπάρχει στο dictionary προσθέτουμε το id του, αλλιώς προσθέτουμε όλη την λέξη μαζί με το id.

for file in os.listdir(path):  
 file\_path = os.path.join(path, file)  
 with open(file\_path, 'r') as folder:  
 text = folder.read()  
 dictionary = text.split()  
  
 *# Create a dictionary to store word counts for each document* count = {}  
  
 for word in dictionary:  
 count[word] = count.get(word, 0) + 1  
  
 *# Update the inverted index with word counts for the current document* for word, count in count.items():  
 inverted\_index[word].append((file, count))  
 *# Update the document count for the current word* if word in document\_count:  
 document\_count[word].add(file)  
 else:  
 document\_count[word] = {file}

Τέλος εκτυπώνουμε το τελικό inverted\_index και το αποθηκεύουμε σε ένα csv (inverted\_index.csv).

inverted\_index\_csv = r"C:\Users\chryssa\_pat\PycharmProjects\pythonProject\inverted\_index.csv"

*# Print the inverted index for each word without changing the overall format*for word, documents in inverted\_index.items():  
 print(f"{word}: {documents}")  
  
*# Save the inverted index into a CSV file*with open(inverted\_index\_csv, 'w', newline='', encoding='utf-8') as csvfile:  
 csv\_writer = csv.writer(csvfile)  
 csv\_writer.writerow(['Word', 'Documents'])  
  
 for word, documents in inverted\_index.items():  
 csv\_writer.writerow([word, documents])

**2.2 Ερώτημα 2**

Ξεκινάμε με τον υπολογισμό του παράγοντα IDF. Αρχικά, για κάθε λέξη word που έχουμε στο ευρετήριό μας διατρέχουμε το document\_count, το οποίο είναι λεξικό με κλειδί τη συγκεκριμένη λέξη και περιεχόμενο τη λίστα των κειμένων στο οποίο εμφανίζεται αυτή η λέξη. Αποθηκεύουμε μετά τη σάρωση στη λίστα documents τα κείμενα όπου εμφανίζεται η κάθε λέξη και υπολογίζουμε το idf ως τον δεκαδικό λογάριθμο του μήκους του φακέλου που έχουμε όλα τα έγγραφα της συλλογής διά το μήκος της λίστας documents. Έτσι καταφέρνουμε να βρούμε τον λόγο από όλα τα κείμενα της συλλογής προς τα κείμενα που περιλαμβάνουν τη κάθε λέξη. Το idf είναι επίσης λεξικό με κλειδί το συγκεκριμένο term και περιεχόμενο κλειδιού τη τιμή του IDF του κλειδιού. Αφού ολοκληρώσουμε την επεξεργασία και γεμίσουμε το λεξικό με τιμές, το μετατρέπουμε σε dataframe και το αποθηκεύουμε σε ένα αρχείο CSV για να μπορούμε να το επεξεργαστούμε πιο εύκολα.

Με όμοιο τρόπο ακριβώς κατασκευάζουμε και το dataframe tf με τη μόνη διαφορά ότι τώρα ορίζουμε ένα μετρητή count, ο οποίος μεταβαίνει στο ευρετήριο και εντοπίζει για κάθε κείμενο document το πλήθος των εμφανίσεων κάθε λέξης του word που είναι αποθηκευμένο στη λίστα occurrences. Έτσι, υπολογίζουμε το TF κάθε term κάθε κειμένου ξεχωριστά ως το δεκαδικό λογάριθμο της τιμής count συν 1. Στο τελικό dataframe που προκύπτει συμπληρώνουμε μηδενικά για τα terms τα οποία δεν υπάρχουν καθόλου μέσω της συνάρτησης fillna( ).

Για τον υπολογισμό του βάρους tf-idf πολλαπλασιάζουμε στη συνέχεια κάθε στήλη (term) του dataframe tf με την αντίστοιχη τιμή του dataframe idf και το τελικό αποτέλεσμα το αποθηκεύουμε στο dataframe tfidf, το οποίο επίσης αποθηκεύουμε σε ένα CSV αρχείο.

**2.4 Ερώτημα 4**

Για το συγκεκριμένο ερώτημα αρχικά θα ταξινομήσουμε σε φθίνουσα σειρά τα αποτελέσματα του vector space model με βάση το cosine similarity και τα αποθηκεύουμε στο dataframe sorted\_doc\_ids και τα εξάγουμε και σε csv cosine\_similarity.csv" . Στην συνέχεια, αντικαθιστούμε τις cosine similarity τιμές με το id κάθε κειμένου το οποίο το μετατρέπουμε και σε ακέραιο για κάθε query, ώστε τα πιο σχετικά id να βρίσκονται στην κορυφή της λίστα και τα λιγότερο σχετικά στο τέλος. Τα αποτελέσματα τα αποθηκεύουμε σε μια λίστα retrieved\_docs στην οποία κλειδί είναι το id του ερωτήματος και το περιεχόμενο είναι λίστα που περιέχει τα id των κειμένων.

*#cosine similarity  
# Sort each column and replace values with doc IDs*sorted\_doc\_ids = cosine\_similarity.apply(lambda col: col.sort\_values(ascending=False).index)  
csv\_file\_path = r"C:\Users\chryssa\_pat\PycharmProjects\pythonProject\cosine\_similarity.csv"  
*# Save the DataFrame to a CSV file*sorted\_doc\_ids.to\_csv(csv\_file\_path, index=False)  
*# Convert document IDs to integers and get the retrieved docs*retrieved\_docs = {query\_id: [int(doc\_id) for doc\_id in doc\_ids] for query\_id, doc\_ids in sorted\_doc\_ids.items()}

Σχετικά με τα αποτελέσματα του colbert τα έχουμε αποθηκεύσει σε csv, η πρώτη στήλη περιέχει τα queris και η δεύτερη τα id των κειμένων που είναι ήδη τοποθετημένα σε αύξουσα σειρά. Οπότε διατρέχουμε κάθε γραμμή του csv και ελέγχουμε αν στη δεύτερη στήλη τα id των κειμένων είναι ακέραιοι και σε λίστα. Αν είναι τότε τα αποθηκεύουμε στο λεξικό query\_docs όπου το κλειδί είναι το id των ερωτημάτων και τα περιεχόμενα είναι λίστα με όλα τα κείμενα.

*#colbert*colbert\_path = r"C:\Users\chryssa\_pat\PycharmProjects\pythonProject\colbert\_result.csv"  
query\_docs = {}  
*# Read the ColBERT results into a DataFrame*with open(colbert\_path, 'r') as csvfile:  
 csvreader = csv.reader(csvfile)  
 next(csvreader) *# Skip the header row* for query\_id, row in enumerate(csvreader):  
 *# Parse the string representation of the list into an actual list* doc\_ids = ast.literal\_eval(row[1])  
 *# Ensure doc\_ids is a list and contains integers* if isinstance(doc\_ids, list) and all(isinstance(id, int) for id in doc\_ids):  
 query\_docs[query\_id] = doc\_ids

Αναφορικά με το αρχείο csv relevant\_20 που κάθε γραμμή του περιέχει τα σχετικά κείμενα για κάθε query, διατρέχουμε το αρχείο, μετατρέπουμε τα id των κειμένων σε ακεραίους και τα αποθηκεύουμε στην λίστα relevant\_docs, όπου το κλειδί είναι το id των ερωτημάτων και τα περιεχόμενα είναι λίστα με όλα τα κείμενα.

*# Get the relevant docs*file\_path = r"C:\Users\chryssa\_pat\PycharmProjects\pythonProject\Relevant\_20"  
relevant\_docs = {}  
with open(file\_path, 'r') as file:  
 for query\_id, line in enumerate(file):  
 *# Convert all space-separated numbers in the line to integers and store them as a set* doc\_ids = set(map(int, line.split()))  
 *# Assign this set to the corresponding query ID* relevant\_docs[query\_id] = doc\_ids

Όπως αναφέρετε και στο ερώτημα 1.4 θα υλοποιήσουμε τις μετρικές precision-recall, preciosion@k και mean average length. Η precision-recall και η MAP θα υπολογιστούν στην ίδια συνάρτηση calculate\_metrics() ενώ η p@k υπολογίζεται από την συνάρτης precision\_at\_k().

Όσον αφορά την συνάρτηση calculate\_metrics(), δέχεται σαν όρισμα την λίστα με τα σχετικά κείμενα και την λίστα με τα ανακτημένα. Αρχικά ορίζονται δυο μεταβλητές που θα χρησιμοποιηθούν σαν μετρητές η ret\_count που είναι μετρητής για τα ανακτημένα κείμενα και η rel\_count, η οποία είναι μετρητής για τα σχετικά. Επίσης ορίζονται οι λίστες precisions, recalls. Ύστερα, διατρέχουμε με μια επανάληψη όλα τα κείμενα που βρίσκονται μέσα στη λίστα retrieved\_docs και κάθε φορά που πηγαίνουμε σε καινούριο κείμενο αυξάνεται ο μετρητή ret\_count κατά 1. Ελέγχουμε αν το κείμενο που βρισκόμαστε βρίσκεται και στην λίστα relevant\_docs αν βρίσκεται, τότε σημαίνει ότι είναι σχετικό κείμενο οπότε αυξάνουμε τον μετρητή rel\_count κατά 1. Υπολογίζουμε το precision και το recall για το συγκεκριμένο κείμενο εφαρμόζοντας τους τύπους που έχουν αναφερθεί στο ερώτημα 1.4 και τα αποθηκεύουμε στις λίστες precisios, recalls αντίστοιχα, οι οποίες επιστρέφονται όταν καλούμε την συνάρτηση. Για τον υπολογισμό της MAP μέσα στην συνάρτηση αλλά έξω από την επανάληψη γιατί θέλουμε όλη την λίστα precisions για κάθε ερώτημα, θα υπολογίσουμε τον μέσο όρο των precisions (avg\_precision) και το αποτέλεσμα θα επιστρέφεται.

def calculate\_metrics(relevant\_docs, retrieved\_docs):  
 ret\_count = 0  
 rel\_count = 0  
 precisions = []  
 recalls = []  
  
 *# Iterate through each relevant document* for doc in retrieved\_docs:  
 ret\_count += 1  
 *# Check if the relevant document is in the retrieved documents* if doc in relevant\_docs:  
 rel\_count += 1  
 precision = rel\_count / ret\_count  
 precisions.append(precision)  
 recall = rel\_count / len(relevant\_docs)  
 recalls.append(recall)  
 avg\_precision = sum(precisions) / len(relevant\_docs)  
  
 return precisions, recalls, avg\_precision

Για να μπορούμε να συγκρίνουμε την μετρική precision-recall για κάθε μοντέλο(vsm, colbert) θα δημιουργήσουμε μια συνάρτηση calculate\_area(), η οποία θα υπολογίζει το εμβαδόν κάτω από κάθε γραφική για όλα τα ερωτήματα και για τα δύο μοντέλα. Δέχεται σαν όρισμα την λίστα recalls, precision που θα έχει δημιουργηθεί από την προηγούμενη συνάρτηση και διατρέχει

def calculate\_area(recalls, precisions):  
 area = 0.0  
 for i in range(1, len(recalls)):  
 *# Calculate the area of the trapezoid* width = recalls[i] - recalls[i-1]  
 height = (precisions[i] + precisions[i-1]) / 2  
 area += width \* height  
 return area

Για την μετρική preciosion@k, όπως έχουμε αναφέρει ήδη δημιουργούμε την συνάρτηση precision\_at\_k(), οποία παίρνει σαν ορίσματα την λίστα με τα σχετικά κείμενα, την λίστα με τα ανακτημένα και το όριο k. Ελέγχει άμα η λίστα των ανακτηθέντων κειμένων είναι μεγαλύτερη του ορίου k και άμα είναι τότε η λίστα κόβεται και πλέον περιέχει μόνο τα πρώτα k. Στην συνέχεια, διατρέχουμε την καινούρια λίστα και για κάθε κείμενο ελέγχουμε αν είναι σχετικό και επιστρέφουμε 1 αν είναι. Υπολογίζουμε το άθροισμα των σχετικών κειμένων και η συνάρτηση επιστρέφει το αποτέλεσμα της διαίρεσης του αθροίσματος προς το όριο k.

def precision\_at\_k(relevant\_docs, retrieved\_docs, k):  
 if len(retrieved\_docs) > k:  
 retrieved\_docs = retrieved\_docs[:k]  
 relevant\_count = sum([1 for doc in retrieved\_docs if doc in relevant\_docs])  
 return relevant\_count / k

Αφού τελειώσαμε με την ανάλυση των συναρτήσεων τώρα θα εξηγήσουμε τον τρόπο εφαρμογής τους. Αρχικά, ορίζουμε 4 μεταβλητές που μας είναι απαραίτητες:

* **queries** = 20 : το πλήθος των ερωτημάτων.
* **top\_k** = 400 : το όριο k που χρειάζεται για την μετρική precision@k.
* **avg\_precisions** = []: λίστα που θα περιέχει τους μέσους όρους των precisions για κάθε ερώτημα για το μοντέλο vsm.
* avg\_precisions\_colbert = []: λίστα που θα περιέχει τους μέσους όρους των precisions για κάθε ερώτημα για το μοντέλο colbert
* **areas\_cosine\_similarity** = []: λίστα που θα περιέχει τa εμβαδά για κάθε ερώτημα για το μοντέλο vsm.
* **areas\_colbert** = []: λίστα που θα περιέχει τa εμβαδά για κάθε ερώτημα για το μοντέλο colbert.

queries = 20  
top\_k = 400  
avg\_precisions = []  
avg\_precisions\_colbert = []  
areas\_cosine\_similarity = []  
areas\_colbert = []

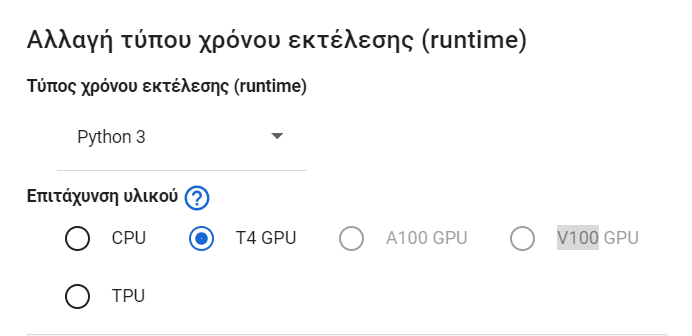
Στην συνέχεια, διατρέχουμε κάθε γραμμή της λίστας relevant\_docs και η λίστα relevant\_docs\_set θα περιέχει τα σχετικά κείμενα. Επίσης, με βάση το id των ερωτημάτων βρίσκουμε και αποθηκεύουμε σε λίστες τα ανακτηθέντα κείμενα για το vsm (retrieved\_docs\_list) και για το clobert (retrieved\_docs\_list\_colbert). Για κάθε μοντέλο ξεχωριστά καλούμε την συνάρτηση calculate\_metrics και την τρίτη παράμετρο που μας επιστρέφει, που είναι ο μέσος όρος των preciions την προσθέτουμε στην λίστα avg\_precisions, avg\_precisions\_colbert ανάλογα με το μοντέλο. Επιπλέον, υπολογίζουμε την MAP που θα είναι το άθροισμα των λιστών προς το πλήθος των ερωτημάτων. Ύστερα, καλούμε και την συνάρτηση Precision\_at\_k και για τα δύο μοντέλα και εκτυπώνουμε τα αποτελέσματά της για κάθε ερώτημα. Τα δυο ορίσματα που μας επιστρέφει η calculate\_metrics που είναι το recall και το precision θα τα αποτυπώσουμε σε ένα γράφημα, όπου ο άξονας χ θα είναι το recall και ο y το precision. Τέλος, υπολογίζουμε τα εμβαδά για κάθε γραφική καλώντας την συνάρτηση calculate\_area, τα αποθηκεύουμε σε λίστες και εκτυπώνουμε το αποτέλεσμα.

for query\_id, relevant\_docs\_set in relevant\_docs.items():  
 retrieved\_docs\_list = retrieved\_docs.get(query\_id, [])  
 retrieved\_docs\_list\_colbert = query\_docs.get(query\_id, [])  
 precisions, recalls, avg\_precision = calculate\_metrics(relevant\_docs\_set, retrieved\_docs\_list)  
 avg\_precisions.append(avg\_precision)  
 map\_values = sum(avg\_precisions) / queries  
  
 precisions\_colbert, recalls\_colbert, avg\_precision\_colbert = calculate\_metrics(relevant\_docs\_set, retrieved\_docs\_list\_colbert)  
 avg\_precisions\_colbert.append(avg\_precision\_colbert)  
 map\_values\_colbert = sum(avg\_precisions\_colbert) / queries  
  
 *# Calculate precision@k for Cosine Similarity and ColBERT* precision\_at\_k\_cosine = precision\_at\_k(relevant\_docs\_set, retrieved\_docs\_list, top\_k)  
 precision\_at\_k\_colbert = precision\_at\_k(relevant\_docs\_set, retrieved\_docs\_list\_colbert, top\_k)  
  
 *# Print or store these values for comparison* print(f"Precision@{top\_k} for Cosine Similarity (Query {query\_id + 1}): {precision\_at\_k\_cosine}")  
 print(f"Precision@{top\_k} for ColBERT (Query {query\_id + 1}): {precision\_at\_k\_colbert}")  
  
 plt.figure()  
 plt.plot(recalls, precisions, marker='o', label = 'Cosine Similarity')  
 plt.plot(recalls\_colbert, precisions\_colbert, marker='o', label = 'ColBERT')  
 plt.title(f"Precision - Recall for Query {query\_id + 1}")  
 plt.xlabel("Recall")  
 plt.ylabel("Precision")  
 plt.grid(True)  
 plt.legend()  
 plt.show()  
  
 *# After calculating precisions and recalls for each query* area\_cosine\_similarity = calculate\_area(recalls, precisions)  
 areas\_cosine\_similarity.append(area\_cosine\_similarity)  
 area\_colbert = calculate\_area(recalls\_colbert, precisions\_colbert)  
 areas\_colbert.append(area\_colbert)  
  
 *# Now you can print or plot the AUC values* print(f"Area for Cosine Similarity (Query {query\_id + 1}): {area\_cosine\_similarity}")  
 print(f"Area for ColBERT (Query {query\_id + 1}): {area\_colbert}")

Μετά το τέλος του βρόγχου επανάληψης και αφού έχουμε επεξεργαστεί όλα τα ερωτήματα τυπώνουμε το τελικό αποτέλεσμα της MAP και υπολογίζουμε το μέσο όρο των εμβαδών και εκτυπώνουμε και αυτό το αποτέλεσμα.

print(f"MAP Metric for Cosine Similarity: {map\_values}")  
print(f"MAP Metric for ColBERT: {map\_values\_colbert}")  
  
*# Calculate the mean area to compare the general efficiency of both models*mean\_area\_cosine\_similarity = sum(areas\_cosine\_similarity) / queries  
mean\_area\_colbert = sum(areas\_colbert) / queries  
  
print(f"Mean Area under Precision-Recall Curve for Cosine Similarity: {mean\_area\_cosine\_similarity}")  
print(f"Mean Area under Precision-Recall Curve for ColBERT: {mean\_area\_colbert}")

Η υλοποίηση του colbert έγινε στο περιβάλλον google colab με τις παρακάτω ρυθμίσεις:



Αρχικά, εγκαθιστούμε virtual environment, το οποίο είναι απαραίτητο για την δημιουργία ενός εικονικού περιβάλλοντος, δημιουργούμε ένα καινούριο εικονικό περιβάλλον που το ονομάζουμε myenv και το ενεργοποιούμε ώστε να απομονώσει το περιβάλλον της python για να εμποδίσει τυχών επικαλύψεις με άλλα πακέτα.

Οι βιβλιοθήκες που χρησιμοποιούμε είναι:

**torch**: είναι συντομογραφία για την ανοικτού κώδικα βιβλιοθήκη μηχανικής μάθησης Pytorch η οποία έχει σχεδιαστεί για την εκτέλεση deep learning και μηχανικής μάθησης μέσω νευρωνικών δικτύων συνεπώς είναι απαραίτητη για την υλοποίηση του ColBert.

**Sys**: χρησιμοποείται η εντολή **sys.path.insert(0, 'ColBERT/')** για να προσθέσει το μονοπάτι του ColBert στο περιβάλλον της python που χρησιμοποιούμε.

**faiss**: είναι ανοικτού κώδικα βιβλιοθήκη και έχει σχεδιαστεί για αποτελεσματική αναζήτηση ομοιότητας και ομαδοποίηση συνόλων δεδομένων σε χώρους υψηλών διαστάσεων που και αυτό είναι απαραίτητο για το ColBert.

**transformers**: είναι ανοικτού κώδικα βιβλιοθήκη, η οποία έχει έτοιμα προεκπαιδευμένα μοντέκα και εργαλεία τα οποία τα χειαζόμαστε στην επεξεργασία φυσικής γλώσσα (NLP), η οποία πραγματοποιείται στο ColBert.

**cuda**: χρησιμοποιείται για την επιτάχυνση της GPU.

**wget**: χρησιμοποιείται για να κατεβάσουμε αρχεία από το διαδίκτυο.

**colbert**: χρησιμοποιείται ώστε να πάρουμε όλες τις απαραίτητες βιβλιοθήκες για την σωστή εκτέλεση του μοντέλου. Θα χρησιμοποιήσουμε την Indexer,Searcher, Run, RunConfig, ColBERTConfig, Queries, Collection

Επίσης, συνδέουμε το google colab με το drive, ώστε να έχει πρόσβαση στα απαραίτητα αρχεία.

ΥΛΟΠΟΙΗΣΗ

Αρχικά, όλα τα κείμενα και τα απαραίτητα αρχεία είναι αποθηκευμένα στο φάκελο docs μέσα στο drive και είναι σε συμπιεσμένα σε .zip μορφή, οπότε τον αποσυμπιέζουμε και τα περνάμε στο /content/docs.

Δημιουργείται η συνάρτηση **read\_documents\_from\_directory(),** η οποία διαβάζει κάθε αρχείο που βρίσκεται μέσα στο φάκελο docs με τον ίδιο τρόπο όπως και στο VSM μοντέλο και αναπαριστά κάθε αρχείο ως λίστα με id το όνομα του αρχείου και το περιεχόμενο του εγγράφου.

Η συνάρτηση **read\_queries\_from\_file(),** διαβάζει το αρχείο που περιέχει τα ερωτήματα και τα αποθηκεύει σε λίστα.

Στην συνέχεια ορίζουμε τα configurations του ColBeart.

* **nbits**: κωδικοποιεί κάθε διάσταση με 2 bit.
* **max\_id** : Ορίζουμε το μέγιστο id των αρχείων που είναι 1209 και συμβολίζει ότι στα 1209 id θα χρησιμοποιηθεί ο indexer.
* **index\_name** : το όνομα του δείκτη.
* **checkpoint**: περιέχει το μονοπάτι που βρίσκεται το checkpoint.

Όσον αφορά το indexing, γίνεται με βάση το προεκπαιδευμένο μοντέλο ColBert με βάση τα configurations που ορίσαμε παραπάνω. Δημιουργεί έναν δείκτη και στην συνέχεια δεικτοδοτεί τα 1209 έγγραφα.

Στην συνέχεια δημιουργούμε τοv searcher και για κάθε query εκτυπώνουμε τα ανακτημένα κείμενα αντιστοιχίζοντας το passage\_id με το σωστό document\_id. Επίσης αποθηκεύομε σε ένα αρχείο csv τα query και για κάθε query τα ανακτημένα κείμενα, ώστε να μπορούμε να τα επεξεργαστούμε στο ερώτημα 4.