Εργασία Ανάκτηση Πληροφορίας
ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2024

Χρήστος Καραμάνος, 1072518, up1072518@upnet.gr

Άγγελος Τσαμόπουλος, 1072591, up1072591@upnet.gr

Η ομάδα μας αποτελείται από τα δύο αναγραφόμενα μέλη και ο φόρτος εργασίας μοιράστηκε ισόποσα ανάμεσα μας.

Για την υλοποίηση όλων των ερωτημάτων χρησιμοποιήσαμε μόνο Python.

Ερώτημα 1

Στο συγκεκριμένο ερώτημα καλούμαστε να παράξουμε δύο ανεστραμμένα ευρετήρια, από τα οποία το ένα θα χρησιμοποιηθεί για την υλοποίηση του Boolean μοντέλου και το άλλο για το Vector Space μοντέλο.

Boolean Inverted Index

Πρόκειται για το πιο απλό ευρετήριο, το οποίο αποθηκεύει τα έγγραφα στα οποία εμφανίζεται ο κάθε όρος. Αυτό επιτρέπει την αποτελεσματική απάντηση σε queries τύπου Boolean, queries δηλαδή που περιλαμβάνουν τους λογικούς τελεστες “AND”, “OR”, “NOT” και οποιουσδήποτε όρους.

- Δομή

  Υλοποίησαμε το συγκεκριμένο ευρετήριο ως ένα **dictionary** στην Python, όπου τα κλειδιά είναι οι όροι που συναντάμε στα κείμενα και οι τιμές είναι **sets** με τα **document id** των κειμένων που εμφανίζεται ο κάθε όρος. Η χρήση dictionary της Python μας εξυπηρετεί, καθώς ο κάθε όρος αποθηκεύεται μία φόρα ως κλειδί και τα σχετικά document ids προστίθενται στα αντίστοιχα sets κάθε φορά που συναντάται σε αυτά ένας νέος ή υπάρχων όρος, οπότε αποφεύγονται τελείως τα διπλότυπα.

- Προεπεξεργασία

  Γεγονός είναι ότι, λόγω της μορφής στης οποίας μας δόθηκαν τα κείμενα, δε χρειάστηκε κάποια ιδιαίτερη προεπεξεργασία. Το tokenization έγινε ανά γραμμή και εξαιρέθηκαν τα stopwords από τη βιβλιοθήκη NLTK της Python, μαζί με όλες τις λέξεις που αποτελούνταν από λιγότερα από 3 γράμματα, καθώς παρατηρήσαμε ότι στα κείμενα υπήρχαν λέξεις 2 γραμμάτων οι οποίες δεν προσθέταν καμία σημασιολογική αξία στα ευρετήρια μας. Σαν αποτέλεσμα, το ευρετήριο έχει μικρότερο μήκος (μείωση χώρου ευρετηρίασης)

- Διαχείριση κειμένων που απουσίαζαν

  Δεν έγινε κάποια ιδιαίτερη διαχείριση σχετικά με τα κείμενα που απουσίαζαν. Το ευρετήριο δημιουργήθηκε με βάση τη συλλογή που δόθηκε, οπότε και περιέχει τα ids των κειμένων που υπάρχουν σε αυτή, καθώς και τους όρους που αυτά εμπεριέχουν.

Vector Space Inverted Index

Το ανεστραμμένο ευρετήριο διανυσματικού χώρου είναι μεταγενέστερο του Boolean και αποθηκεύει τον δείκτη TF-IDF κάθε όρου σε κάθε έγγραφο. Αυτό επιτρέπει τη σύγκριση μεταξύ του query και των κειμένων και την

κατάταξη των δεύτερων με βάση τις μεταξύ τους σημασιολογικές ομοιότητες στον διανυσματικό χώρο.

- Δομή

  Η δομή του συγκεκριμένου ευρετηρίου είναι ένα εμφωλευμένο **dictionary** στη Python, δηλαδή ένα dictionary μέσα σε ένα άλλο dictionary. Το εξωτερικό dictionary έχει ως κλειδιά τους όρους που συναντώνται στα κείμενα και ως τιμές ολόκληρα dictionaries. Αυτά με τη σειρά τους έχουν ως κλειδιά τα **document id** των κειμένων στα οποία εμφανίζονται και ως τιμές τον δείκτη **TF-IDF** για τον συγκεκριμένο όρο στο συγκεκριμένο κείμενο. Η δομή αυτή επιλέγεται, καθώς για την υλοποίηση του vector space μοντέλου απαιτείται η αντιστοίχιση κάθε όρου με τα κείμενα στα οποία εμφανίζεται, αλλά και με τον δείκτη TF-IDF του όρου για αυτά τα κείμενα. Συνεπώς, το εμφωλευμένο dictionary προσφέρει αποδοτικότητα στη μνήμη και άμεση πρόσβαση στην τιμή TF-IDF για οποιοδήποτε όρο σε όλα τα κείμενα που εμφανίζεται που είναι το ζητούμενο όταν επεξεργαζόμαστε queries στο vector space.

- Προεπεξεργασία

  Η προεπεξεργασία που έγινε στα κείμενα είναι ίδια με αυτή για το Boolean ανεστραμμένο ευρετήριο. Εδώ θα εξηγήσουμε τα βήματα υπολογισμού του δείκτη TF-IDF.

  Πρώτα δημιουργήσαμε ένα εμφωλευμένο dictionary (**doc_term_frequencies**), όπου αποθηκεύσαμε πόσες φορές εμφανίζεται κάθε όρος σε κάθε κείμενο (Term Frequencies).

  ‘Επειτα, χρησιμοποιούμε ένα απλό dictionary (**term_doc_frequency**) για να αποθηκεύσουμε το πλήθος των κειμένων που εμφανίζεται κάθε όρος (Document Frequencies).

Στη συνέχεια, υπολογίζουμε το TF από τον εξής τύπο:

$TF(t,d) = \frac{Frequency\ of\ t\ in\ d}{Maximum\ frequency\ of\ terms\ in\ d}$, όπου ο αριθμητής είναι η τιμή

του εσωτερικού dictionary στο doc_term_frequencies και ο παρονομαστής προκύπτει ως το άθροισμα όλων των τιμών του ίδιου dictionary. Να τονίσουμε, πως το κλειδί του εξωτερικού dictionary είναι τα document ids, οπότε αυτές οι τιμές που αναφέραμε παραπάνω προκύπτουν για κάθε κείμενο ξεχωριστά.

Ακολουθεί ο υπολογισμός του IDF, μέσω του τύπου:

$$IDF(t) = \log\left(\frac{\text{Total Documents}}{1 + \text{DF(t)}}\right)$$

, όπου το DF(t) στον παρονομαστή είναι το πλήθος των κειμένων, στο οποίο εμφανίζεται ο όρος και έχει υπολογιστεί ως το κλειδί του term_doc_frequency dictionary.

Τέλος, ο τελικός υπολογισμός του δείκτη TF-IDF γίνεται ως εξής:

$$TF\text{-}IDF(t,d) = TF(t,d) \cdot IDF(t)$$

και αποθηκεύεται ως τιμή στο τελικό εμφωλευμένο dictionary, το οποίο έχει ως κλειδί του εξωτερικού dictionary τους όρους, κλειδί του εσωτερικού τα ids των κειμένων στα οποία εμφανίζεται ο κάθε όρος και για κάθε κείμενο που εμφανίζεται τον δείκτη TF-IDF ως τιμή του εεσωτερικού

- Διαχείριση κειμένων που απουσίαζαν

  Ισχύει το ίδιο με το Boolean ευρετήριο.

Ερώτημα 2

Σε αυτό το ερώτημα καλούμαστε να υλοποιήσουμε το μοντέλο ανάκτησης πληροφορίας Boolean χρησιμοποιώντας ως είσοδο το αντίστοιχο ευρετήριο που δημιουργήσαμε στο 1ο ερώτημα. Στόχος είναι το μοντέλο μας, αφού λάβει ως είσοδο ενα λογικό ερώτημα με τελεστές AND, OR ή NOT να επιστρέψει τα σωστά κείμενα από το ευρετήριο.

Αλγοριθμική Περιγραφή

Η υλοποίηση για το συγκεκριμένο μοντέλο είναι αρκετά απλή. Το πρώτο βήμα ήταν να διαχειριστούμε το **query** που θα είχαμε ως είσοδο και αυτό θα έπρεπε να μετατραπεί από **infix** μορφή σε **postfix**, ώστε να είναι διαχειρίσιμο από το μοντέλο μας που χρησιμοποιεί **stack** για τη λειτουργία. Η μετατροπή πραγματοποιήθηκε με τη συνάρτηση **infix_to_postfix(query)** και ένα ενδεικτικό παράδειγμα εφαρμογής της είναι το εξής:

```
Query: nephrotoxic AND (reversibly OR vestibular)
Postfix Query: ['NEPHROTOXIC', 'REVERSIBLY', 'VESTIBULAR', 'OR', 'AND']
```

Το Postfix Query λοιπόν θα μπει ως είσοδος στη συνάρτηση υλοποίησης του μοντέλου (**boolean_retrieval_model(query, inverted_index, all_docs)**) μαζί με το αντίστοιχο ανεστραμμένο ευρετήριο και το directory με όλα τα κείμενα. Όπως προαναφέραμε, η υλοποίηση του μοντέλου γίνεται με τη βοήθεια ενός **stack** και η λειτουργία του είναι η εξής:

- Αν το token είναι ένα **term**, τότε στην κορυφή του stack γίνεται push το set με όλα τα document id που περιέχουν τον όρο αυτό από το ευρετήριο.
- Αν το token είναι ο τελεστής **OR** (ή αντίστοιχα **AND**), εκτελούνται 2 pop για τις 2 κορυφαίες θέσεις του stack και έπειτα ανάμεσα σ'αυτά τα 2 set εκτελείται η λογική πράξη OR (AND). Το αποτέλεσμα γίνεται push στο stack.
- Αν το token είναι ο τελεστής **NOT**, τότε εκτελείται μία φορα pop και με το set που προκύπτει εκτελείται η λογική πράξη *all docs NOT set*. Το

αποτέλεσμα, που είναι όλα τα κείμενα που ΔΕΝ περιέχουν τον όρο που ακολουθεί το NOT, γίνεται push στο stack.

Ας δούμε πως λειτουργεί το μοντέλο μας στο προηγούμενο παράδειγμα. Αρχικά, διαβάζονται οι τρείς όροι και στην κορυφή της στοίβας έχουμε:
setV(vestibular)
setR(reversibly)
setN(nephrotoxic)

Έπειτα, διαβάζουμε OR οπότε και εκτελείται **setV OR setR** και το αποτέλεσμα **setVR** γίνεται push στην κορυφή της λίστας.

Τέλος, διαβάζουμε AND και εκτελείται **setVR AND setR**. Το αποτέλεσμα είναι το ζητούμενο και τοποθετείται στην κορυφή του stack, απ'όπου με ένα pop μπορούμε να το εμφανίσουμε.

```
Query: nephrotoxic AND (reversibly OR vestibular)
Postfix Query: ['NEPHROTOXIC', 'REVERSIBLY', 'VESTIBULAR', 'OR', 'AND']
Result: {'01078', '01071'}
```

Προβλήματα που αντιμετωπίσαμε

Το κύριο πρόβλημα που αντιμετωπίσαμε ήταν, αφού καταλήξαμε ότι το μοντέλο θα πραγματοποιηθεί με τη χρήση stack, η συνειδητοποίηση ότι το query μας χρειάζεται μετατροπή από infix σε postfix μορφή και η διαδικασία που ακολουθήσαμε για την υλοποίηση της.

Ο αλγόριθμος στον οποίο βασιστήκαμε ήταν ο **Shunting-Yard**. Πρώτα ορίσαμε τις προτεραιότητες των τελεστών ως εξής:

- **NOT**: υψηλότερη προτεραιότητα
- **AND**: δεύτερη υψηλότερη προτεραιότητα
- **OR**: τρίτη υψηλότερη προτεραιότητα

Τα βήματα του αλγορίθμου είναι τα παρακάτω (από το geekforgeeks):

1. Scan the infix expression **from left to right**.
2. If the scanned character is an operand, put it in the postfix expression.
3. Otherwise, do the following
   - If the precedence of the current scanned operator is higher than the precedence of the operator on top of the stack, or if the stack is empty, or if the stack contains a '(', then push the current operator onto the stack.
   - Else, pop all operators from the stack that have precedence higher than or equal to that of the current operator. After that push the current operator onto the stack.
4. If the scanned character is a '**(**', push it to the stack.
5. If the scanned character is a '**)**', pop the stack and output it until a '**(**' is encountered, and discard both the parenthesis.
6. Repeat steps **2-5** until the infix expression is scanned.
7. Once the scanning is over, Pop the stack and add the operators in the postfix expression until it is not empty.
8. Finally, print the postfix expression.

Προγραμματιστικά έγινε χρήση stack και σε αυτή την περίπτωση.

Ερώτημα 3

Το ερώτημα 3 αφορά την υλοποίηση του **Vector Space Model**.

Αρχικά, το μοντέλο δέχεται σαν είσοδο τα ερωτήματα που υπάρχουν στο αρχείο **Queries.txt** με τα οποία ελέγχουμε αν δουλεύει σωστά το μοντέλο μας. Συνεπώς, έχουμε την συνάρτηση **load_queries** η οποία παίρνει σαν είσοδο το path του Queries.txt και χρησιμοποιούμε ένα λεξικό (queries) στο οποίο θα αποθηκεύσουμε τα ίδια τα queries ως εξής:

τα ids των queries θα είναι τα keys και το περιεχόμενο αυτών θα είναι οι τιμές όπως φαίνεται παρακάτω για παράδειγμα για τα 3 πρώτα

{ "1": " How effective are inhalations of mucolytic agents in the treatment of CF patients ",

"2": " What is the role of aerosols in the treatment of lung disease in CF patients ",

"3": " What is the role of bacterial phagocytosis by alveolar macrophages or polymorphonuclear leukocytes in lung disease in CF patients

" }

Έπειτα, έχουμε την συνάρτηση **process_query** η οποία παίρνει σαν είσοδο ένα query, το ανεστραμμένο αρχείο (για το διανυσματικό μοντέλο) και το πλήθος των κειμένων. Σκοπός εδώ είναι να βρούμε το πλήθος εμφανίσεων κάθε λέξης (term frequency) και το idf για μια συγκεκριμένη ερώτηση και η συνάρτηση να επιστρέψει το κατάλληλο **query vector** για κάθε λέξη.

Κάθε φορά, τεμαχίζουμε κάθε ερώτημα σε λέξεις και τις αποθηκεύουμε σε μια λίστα με όνομα **tokens**. Έπειτα, βρίσκουμε το πλήθος εμφανίσεων της κάθε λέξης στο query με την συνάρτηση **Counter** και παίρνουμε την μέγιστη συχνότητα αυτών και την βάζουμε στην μεταβλητή **max_freq**.

Δηλώνουμε επίσης ένα λεξικό με όνομα **query_vector** όπου θα αποθηκεύει τις tf-idf τιμές για κάθε λέξη του ερωτήματος (στην ουσία αναπαριστά το ερώτημα ως διάνυσμα). Υπολογίζουμε την tf value με τον παραπάνω τύπο που δείξαμε και το idf αντίστοιχα. Όποτε, το βάρος ενός όρου μιας ερώτησης υπολογίζεται ως εξής (πρώτος όρος το tf και δεύτερος το idf):

$$w(i,q) = \frac{freq(i,q)}{\max(freq(l,q))} \cdot \log\left(\frac{N}{1+ni}\right),$$

όπου N το σύνολο όλων των κειμένων και ni το πλήθος των κειμένων που εμφανίζεται η λέξη (ο υπολογισμός του τελευταίου γίνεται με την βοήθεια του ανεστραμμένου αρχείου)

Όταν τελειώσει η συγκεκριμένη διαδικασία για κάθε λέξη του ερωτήματος, επιστρέφεται το **query vector.**

Τέλος, η πιο σημαντική συνάρτηση είναι η **rank_documents** όπου παίρνει σαν είσοδο το query vector και το ανεστραμμένο αρχείο και προσπαθεί να κάνει κατάταξη των κειμένων, που ανακτήθηκαν από κάποιο ερώτημα, με βάση το συνημίτονο που προέκυψε από τα διανύσματα των κειμένων και του ερωτήματος.

Βλέπει κάθε λέξη του ερωτήματος και εφόσον παρατηρεί ότι υπάρχει στο ανεστραμμένο αρχείο, ανακτά τα κείμενα στα οποία περιέχεται η λέξη και την αντίστοιχη tf-idf τιμή και τα αποθηκεύει στο λεξικό **document_vectors**. Μετά καλεί την συνάρτηση **cosine_similarity** και υπολογίζει το συνημίτονο

με το κλασσικό τύπο. Την τιμή που βρίσκει την αποθηκεύει στην μεταβλητή (λεξικό) **document_scores** σαν value και keys τα ids των κειμένων. Γίνεται μια φθίνουσα ταξινόμηση των τιμών και των κειμένων και επιστρέφονται τα πρώτα δέκα σαν απάντηση.

Από κάτω, δείχνουμε το αποτέλεσμα εκτέλεσης του συγκεκριμένου αρχείου όπου για κάθε ερώτημα φέρνουμε τα πρώτα 10 αποτελέσματα:

```
Processing Query 1: How effective are inhalations of mucolytic agents in the treatment of CF patients
Top Ranked Documents for Query 1:
  Document 00321: 0.7761
  Document 00546: 0.7086
  Document 00592: 0.6893
  Document 00213: 0.6300
  Document 00542: 0.6268
  Document 01092: 0.6245
  Document 00945: 0.5822
  Document 01029: 0.5697
  Document 00193: 0.4996
  Document 00067: 0.4615

Processing Query 2: What is the role of aerosols in the treatment of lung disease in CF patients
Top Ranked Documents for Query 2:
  Document 00321: 0.8299
  Document 00090: 0.8197
  Document 00025: 0.8062
  Document 00093: 0.7953
  Document 00327: 0.7653
  Document 00432: 0.5578
  Document 00501: 0.5528
  Document 00498: 0.5479
  Document 00986: 0.5403
  Document 00945: 0.5296

Processing Query 3: What is the role of bacterial phagocytosis by alveolar macrophages or polymorphonuclear leukocytes in lung disease in CF patients
Top Ranked Documents for Query 3:
  Document 00018: 0.6764
  Document 01203: 0.6628
  Document 00669: 0.6257
  Document 00572: 0.5546
  Document 00445: 0.5531
  Document 00306: 0.5530
  Document 00432: 0.5348
  Document 00397: 0.5046
  Document 00270: 0.4402
  Document 00192: 0.4268

Processing Query 4: What is the relationship between Haemophilus influenzae and Pseudomonas aeruginosa in CF patients
Top Ranked Documents for Query 4:
  Document 00160: 0.9177
```

Ερώτημα 4

Στο συγκεκριμένο ερώτημα, σκοπός είναι η αξιολόγηση και η σύγκριση των δυο μοντέλων χρησιμοποιώντας τα αρχεία **Queries.txt** και **Relevant.txt** που υπάρχουν στον φάκελο **collection** του project.

Υπάρχουν 4 αρχεία που αφορούν το ερώτημα. Το πρώτο είναι το **generate_boolean_queries.py** το οποίο μετατρέπει τα ερωτήματα του Queries.txt (που είναι σε φυσική γλώσσα) σε boolean ερωτήματα για να μπορέσουμε να εκτιμήσουμε το boolean μοντέλο. Αφού κάναμε tokenization τα queries (βγάλαμε τα stop words και την λέξη patients επειδή ήταν πολύ συχνή) συνδέσαμε τα terms που ποέκυπταν με κάθε query με OR εκτός από το query 12 που συνδέσαμε τα terms του με AND. Στη συνέχεια, τα παραγόμενα queries τα αποθηκεύουμε σε ένα αρχείο με όνομα **Boolean_Queries.txt** που βρίσκεται στο φάκελο collection.

Το δεύτερο αρχείο είναι το **boolean_metrics.py** με το οποίο υπολογίζεται η ακρίβεια και η ανάκληση για το boolean μοντέλο. Εδώ πέρα, φορτώνουμε τα αρχεία Boolean_Queries.txt και Relevant.txt και αφού γίνει αυτό, καλείται η συνάρτηση **compute_precison_and_recall**. Έχοντας ήδη κάνει import τις συναρτήσεις infix_to_postfix και boolean_retrieval_model από το αρχείο boolean_model.py, τις καλούμε για να ανακτήσουμε τα αρχεία που φέρνει το μοντέλο και τα αναθέτουμε στην μεταβλητή retrieved_docs. ‘Επειτα, επειδή όλα τα documents είναι της μορφής 00xxx και στο αρχείο Relevant.txt είναι της μορφής xxx αναιρούμε τα μπροστινά μηδενικά από τα ανακτηθέντα κείμενα (δηλαδή το 00023 γίνεται 23). Τέλος, χρησιμοποιούμε 4 μεταβλητές, τις relevant, relevant_retrieved, precision και recall με τις οποίες παίρνουμε τα σχετικά κείμενα, την τομή των ανακτηθέντων και σχετικών και υπολογίζουμε τα 2 τελευταία αντίστοιχα με τους 2 παρακάτω τύπους:

$$\Pr e\,cision = \frac{Ra}{R} \qquad Recall = \frac{Ra}{A},$$

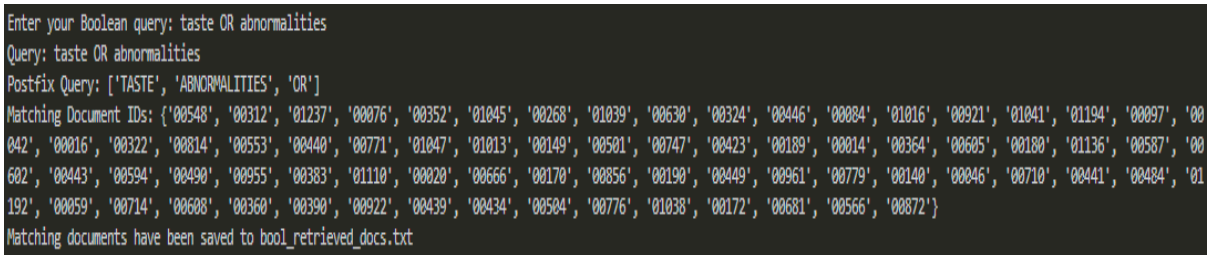
όπου Ra είναι η μεταβλητή relevant_retrieved, R η μεταβλητή retrieved_docs και A η μεταβλητή relevant.

Επιστρέφεται μετά από αυτούς τους υπολογισμούς το λεξικό scores όπου περιέχει τις τιμές της ανάκλησης και ακρίβειας για κάθε boolean query.

Ας δούμε ένα παράδειγμα ότι δουλεύει σωστά.

Έστω ότι εξετάζουμε το query 19 το οποίο μετά την επεξεργασία, έχει προκύψει το boolean query: **normal OR abnormalities**

Όταν τρέχουμε αυτό το boolean query, ανακτάμε κάποια κείμενα όπως βλέπουμε και στο παρακάτω screenshot:

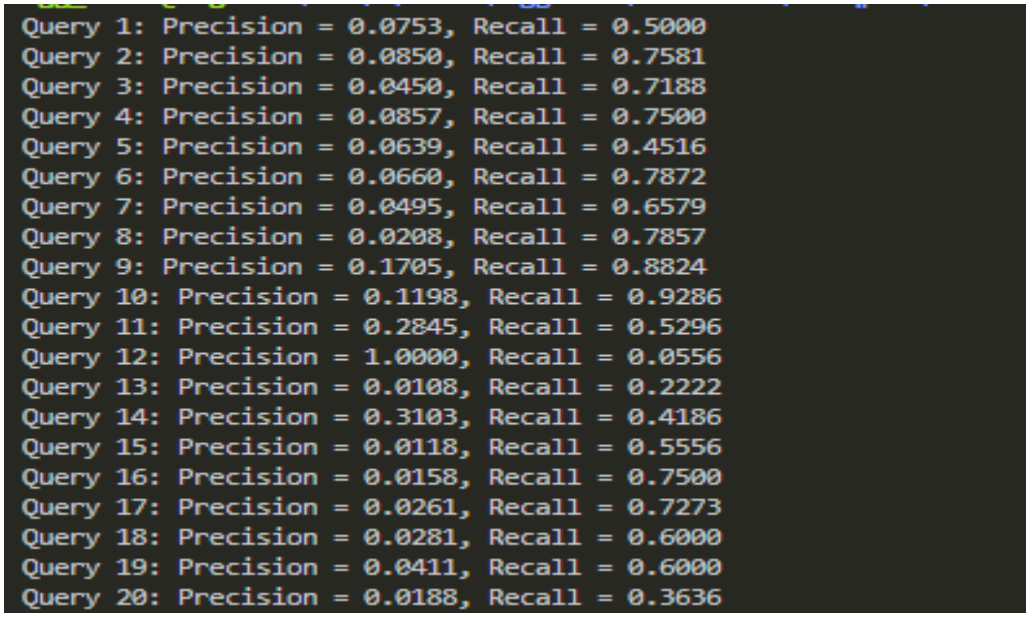


Στο αρχείο Relevant.txt, τα κείμενα που είναι σχετικά με το query 19 είναι τα **268, 324, 449, 992, 1191**.

Μπορούμε να παρατηρήσουμε ότι στα ανακτηθέντα κείμενα περιέχονται τα κείμενα 268, 324 και 449.

Συνεπώς η ανάκληση θα είναι 3/5=0.6 και η ακρίβεια θα είναι 3/73=0.411

Παρακάτω βλέπουμε τις 2 αυτές μετρικές για κάθε ερώτημα (παρατηρούμε ότι όντως για το ερώτημα 19 είναι σωστά):



Παρατηρούμε ότι η ακρίβεια είναι πολύ πιο μικρή από την ανάκληση λόγω το ότι ανακτάμε πολλά κείμενα τα οποία δεν είναι σχετικά (λόγω του OR operator). Στο query 12, που έχουμε βάλει AND operator, η ακρίβεια είναι 1 και η ανάκληση 0.055 λόγω του ότι ανακτάμε πολύ λίγα κείμενα. Με λίγα λόγια, το Boolean μοντέλο συχνά επιστρέφει είτε πολύ λίγα είτε υπερβολικά πολλά κείμενα ως απάντηση σε ερώτημα του χρήστη (ανάλογα αν έχουμε OR ή AND)

Προφανώς δεν έχει νόημα να υπολογίζουμε την ακρίβεια και ανάκληση για για τα πρώτα κέιμενα καθώς κάθε φορά επιστρέφονται τα κείμενα με διαφορετική σειρά λόγω του ότι δεν γίνεται κατάταξη των κειμένων όπως γίνεται στο vector space μοντέλο.

Το τρίτο αρχείο είναι το **vect_metrics.py** το οποίο με τη σειρά του υπολογίζει ακρίβεια και ανάκληση για το vector space μοντέλο. ‘Εχουμε βάλει να επιστρέφονται τα πρώτα δέκα κείμενα για κάθε ερώτημα και κάθε φορά που ανακτάμε από ένα κείμενο, τότε υπολογίζεται οι δυο μετρικές. Αφού φορτώσουμε τα αρχεία Queries.txt και Relevant.txt και κάνουμε import τις συναρτήσεις process_query και rank_documents από το αρχείο vector_space_model, καλείται η συνάρτηση compute_incremental_metrics.vsm η οποία καλεί με την σειρά της τις δυο πάνω συναρτήσεις για να ανακτήσει. Στη συνέχεια, δηλώνουμε 4 μεταβλητές:

1. **retrieved_docs** η οποία είναι μια λίστα που αποθηκεύει τα ανακτηθέντα κείμενα
2. **relevant** που παίρνει τα σχετικά κείμενα
3. **precision_at_k** που είναι μια λίστα όπου αποθηκεύομε την ακρίβεια για τα 10 πρώτα κείμενα (ακρίβεια για 1 κείμενο, ακρίβεια για 2 κείμενα κτλ)
4. **recall_at_k** που είναι μια λίστα όπου αποθηκεύουμε την ανάκληση για τα 10 πρώτα κείμενα

Κάνουμε ότι κάναμε και στο αρχείο **boolean_metrics.py** (υπολογίζουμε τις 2 μετρικές) μόνο που κάθε φορά που υπολογίζουμε ακρίβεια και ανάκληση για κάθε κείμενο, τις τιμές τους τις ενσωματώνουμε στις 2 λίστες precision_at_k και recall_at_k. Έτσι ανακτάμε τις τιμές τους για τα πρώτα κείμενα της επιστρεφόμενης λίστας, συνολικά για κάθε ερώτημα και για όλα τα ερωτήματα της συλλογής.

Ας δούμε ένα παράδειγμα.

Εξετάζουμε το query 15. Το query 15 έχει σχετικά κείμενα σύμφωνα με το αρχείο Relevant.txt τα **12, 30, 43, 192, 257, 322, 603, 606** και **1097** (9 κείμενα). Τα ανακτηθέντα κείμενα που έφερε το μοντέλο και είναι ταξινομημένα κατά φθίνουσα σειρά ως προς την ‘βαθμολογία τους’ φαίνονται παρακάτω (βλέπουμε για κάθε query):

```
Query 1: Retrieved documents = ['321', '546', '592', '213', '542', '1092', '945', '1029', '193', '67']
Query 2: Retrieved documents = ['321', '90', '25', '93', '327', '432', '501', '498', '986', '945']
Query 3: Retrieved documents = ['18', '1203', '669', '572', '445', '306', '432', '397', '270', '192']
Query 4: Retrieved documents = ['160', '922', '668', '550', '986', '784', '8', '718', '1112', '555']
Query 5: Retrieved documents = ['941', '522', '533', '549', '553', '1083', '508', '223', '355', '875']
Query 6: Retrieved documents = ['160', '550', '668', '986', '922', '8', '1112', '718', '1143', '1077']
Query 7: Retrieved documents = ['860', '501', '1091', '195', '689', '563', '386', '1145', '1169', '17']
Query 8: Retrieved documents = ['457', '614', '761', '432', '176', '501', '1091', '498', '986', '843']
Query 9: Retrieved documents = ['866', '1083', '148', '802', '778', '1', '18', '176', '590', '505']
Query 10: Retrieved documents = ['176', '1227', '265', '7', '161', '177', '1086', '1090', '447', '912']
Query 11: Retrieved documents = ['333', '207', '770', '694', '559', '875', '443', '658', '763', '99']
Query 12: Retrieved documents = ['148', '1093', '910', '952', '941', '804', '765', '971', '258', '661']
Query 13: Retrieved documents = ['272', '401', '983', '51', '52', '77', '210', '222', '234', '245']
Query 14: Retrieved documents = ['742', '435', '498', '442', '504', '705', '740', '1234', '43', '264']
Query 15: Retrieved documents = ['257', '30', '883', '615', '606', '348', '441', '950', '241', '821']
Query 16: Retrieved documents = ['673', '455', '1088', '454', '322', '36', '29', '1084', '370', '607']
Query 17: Retrieved documents = ['924', '751', '958', '44', '435', '418', '1185', '426', '1225', '538']
Query 18: Retrieved documents = ['180', '360', '498', '1156', '1143', '1157', '562', '1118', '1184', '394']
Query 19: Retrieved documents = ['268', '324', '449', '42', '84', '149', '170', '312', '360', '440']
Query 20: Retrieved documents = ['183', '579', '1017', '392', '36', '818', '727', '782', '725', '370']
```

Συνεπώς, βλέπουμε ότι το πρώτο ανακτηθέν κείμενο είναι το 257 το οποίο υπάρχει στην λίστα των relevant => Άρα precision: 1/1=1 και recall 1/9=0.1111

Το επόμενο κέιμενο είναι το 30 το οποίο υπάρχει στη λίστα των relevant => Άρα precision=2/2=1 και recall=2/9=0.2222

Το επόμενο κείμενο είναι το 883 το οποίο δεν υπάρχει στην λίστα των relevant => Άρα precision = 2/3=0.667 και recall=2/7=0.2222

Παρακάτω βλέπουμε τις τιμές για τα 10 πρώτα κείμενα:

```
Query 15:
  Precision@1: 1.0000, Recall@1: 0.1111
  Precision@2: 1.0000, Recall@2: 0.2222
  Precision@3: 0.6667, Recall@3: 0.2222
  Precision@4: 0.5000, Recall@4: 0.2222
  Precision@5: 0.6000, Recall@5: 0.3333
  Precision@6: 0.5000, Recall@6: 0.3333
  Precision@7: 0.4286, Recall@7: 0.3333
  Precision@8: 0.3750, Recall@8: 0.3333
  Precision@9: 0.3333, Recall@9: 0.3333
  Precision@10: 0.3000, Recall@10: 0.3333
```

Όταν τρέχουμε το vect_metrics.py, επιστρέφει τις τιμές για τις 2 μετρικές για όλα τα queries και για κάθε ένα από τα 10 πρώτα κείμενα.

Παρατηρούμε εδώ ότι οι μετρικές τρέχουν πολύ καλύτερα συνολικά από το boolean μοντέλο λόγω του tf-idf που χρησιμοποιούμε και ανακτά κείμενα που προσεγγίζει τις συνθήκες της ερώτησης

Τέλος, το τελευταίο αρχείο είναι το **exec_time.py** με το οποίο υπολογίζουμε τον χρόνο εκτέλεσης για τα 2 αυτά μοντέλα αλλά και τον χρόνο ευρετηρίασης. Το παρακάτω screenshot δείχνει ένα παράδειγμα εκτέλεσης αυτού του αρχείου:

```
Boolean Indexing Time: 3.3947 seconds
Vector Space Model Indexing Time: 3.5860 seconds
Boolean Model Execution Time: 0.0013 seconds
Vector Space Model Execution Time: 0.0523 seconds
```

Για να υπολογίσουμε τους χρόνους, χρησιμοποιούμε την βιβλιοθήκη time και την συνάρτηση time.time(). Πριν και μετά από τις κατάλληλες συναρτήσεις για τις ευρετηριάσεις και τα μοντέλα (τα έχουμε κάνει import), χρησιμοποιούμε τις μεταβλητές **start_time** και **end_time** και την διαφορά τους για να υπολογίσουμε τον χρόνο που κάνουν να εκτελεστούν.

Παρατηρούμε ότι οι ευρετηριάσεις κάνουν πολύ παραπάνω χρόνο για να τρέξει από τα μοντέλα και το vector space κάνει παραπάνω χρόνο για να τρέξει λόγω του reranking.

Ερώτημα 5

Σ’αυτό το ερώτημα καλούμασταν να υλοποιήσουμε δύο εκδοχές ενός chatbot, μία που θα δεχόταν ως είσοδο τα κείμενα που παράγει το Boolean μοντέλο ανάκτησης και μία με αυτά που παράγει το vsm μοντέλο. Για να υλοποιήσουμε το chatbot ακολουθήσαμε τις οδηγίες της εκφώνησης. Συγκεκριμένα, πρώτα κατεβάσαμε το Ollama και έπειτα επιλέξαμε και

κατεβάσαμε ένα από τα διαθέσιμα γλωσσικά μοντέλα με βάσει τους περιορισμούς που μας επέβαλλε το σύστημα μας. Κατεβάσαμε όλα τα απαραίτητα πακέτα, όπως το LangChain framework και όλες τις σχετικές βιβλιοθήκες του. Επιπλέον, τροποποιήσαμε τους κώδικες για τα δύο μοντέλα, έτσι ώστε κάθε φορά που θα εκτελείται το κάθε μοντέλο να αποθηκεύει το περιεχόμενο των κειμένων που επέστρεψε ως απάντηση στο query. Στην περίπτωση του Boolean μοντέλου το περιεχόμενο των κειμένων αποθηκεύεται σε ένα αρχείο με όνομα "bool_retrieved_docs.txt" στο directory του Task2, ενώ στην περίπτωση του vsm μοντέλου επιλέγονται τα κορυφαία 5 κείμενα που επιστρέφονται για κάθε query και το περιεχόμενο αυτών αποθηκεύεται ξεχωριστά για κάθε query σε ένα φάκελο με όνομα "vsm_retrieved_docs". Σε αυτά τα αρχεία έχει πρόσβαση το chatbot, τα "διαβάζει" και έπειτα είναι σε θέση να απαντήσει σε οποιαδήποτε ερώτηση του χρήστη σχετική με αυτά. Η διεπαφή του χρήστη γίνεται μέσω γραμμής εντολών του τερματικού.

Screenshots

Παρακάτω παραθέτουμε ορισμένα screenshots που αναδεικνύουν τη λειτουργία του chatbot.

Αρχικά, τρέχουμε το Boolean μοντέλο με το εξής query:

```
Enter your Boolean query: immunoassay or canadian
Query: immunoassay or canadian
Postfix Query: ['IMMUNOASSAY', 'CANADIAN', 'OR']
Matching Document IDs: {'00222', '01105', '01028'}
Matching documents have been saved to bool_retrieved_docs.txt
```

Οπότε και αποθηκέυονται στο "bool_retrieved_docs.txt" το περιεχόμενο αυτών των τριών κειμένων.

Εδώ έχουμε μόλις εκκινήσει το chatbot και καλούμαστε να επιλέξουμε ποιά εκδοχή του θέλουμε να χρησιμοποιήσουμε.

```
(myenv) C:\codes\langChainTest>python chatbot.py
Select Chatbot: 1 for Boolean, 2 for VSM
Enter your choice:
```

Επιλέγουμε το Boolean chatbot

```
Select Chatbot: 1 for Boolean, 2 for VSM
Enter your choice: 1

Boolean Chatbot is ready! Type 'exit' to quit.

Your question:
```

Του κάνουμε μια ερώτηση για τα κείμενα και λαμβάνουμε την εξής απάντηση

```
Your question: What's the main subject of the documents?
Boolean Response:  The main subjects of the documents are sweat tests, cystic fibrosis diagnosis in newborns, enzyme imm
unoassay of albumin as an aid to diagnosis of cystic fibrosis, pancreatic insufficiency in children with cystic fibrosis
, protein concentration in meconium, and reliability of sweat test in cystic fibrosis.
```

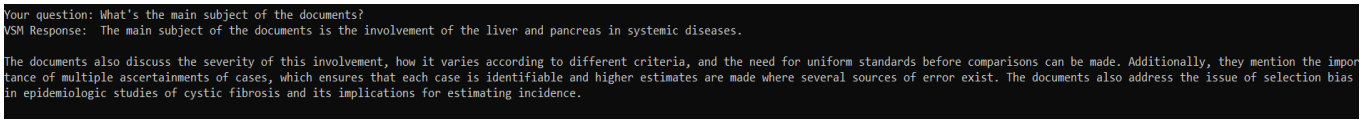
Τερματίζουμε το Boolean chatbot, επιλέγουμε το vsm και καλούμαστε να επιλέξουμε το query του οποίου τα κορυφαία 5 κείμενα που επιστρέφονται ως απάντηση από το vsm μοντέλο ανάκτησης θα τροφοδοτήσουν το chatbot μας.

```
Select Chatbot: 1 for Boolean, 2 for VSM
Enter your choice: 2
Choose a query (1-20) for the VSM chatbot:
Enter query number:
```

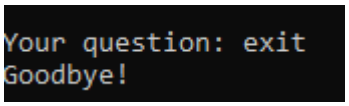
Επιλέγουμε το query 5

```
Enter query number: 5

VSM Chatbot is ready! Type 'exit' to quit.

Your question:
```

Κάνουμε την ίδια ερώτηση και παίρνουμε την εξής απάντηση



Και τερματίζουμε τη λειτουργία του



Απαντήσεις των chatbots

Δε θα λέγαμε ότι εντοπίζουμε μεγάλη διαφορά ως προς την ποιότητα και ακρίβεια των απαντήσεων. Το Boolean chatbot είναι πιο ακριβές και εύστοχο στις απαντήσεις του, ενώ το vsm κάποιες φορές απαντάει με αχρείαστα πολύπλοκο τρόπο. Αν έπρεπε να επιλέξουμε κάποιο για καλύτερο θα λέγαμε το Boolean, αλλά χωρίς ξεκάθαρο προβάδισμα.

Προβλήματα

Συναντήσαμε πληθώρα δυσκολιών στην προσπάθεια μας να υλοποιήσουμε το chatbot, κυριώς σχετικά με την εγκατάσταση όλων των απαραίτητων βιβλιοθηκών και πακέτων, αλλά και με την επιλογή μοντέλου. Το σύστημα στο οποίο θα τρέχαμε το chatbot διέθετε λιγότερα από 6GB RAM για χρήση με αποτέλεσμα η εύρεση κατάλληλου μοντέλου που να μπορεί να τρέξει αποδοτικά να γίνεται δύσκολη. Εν τέλει, καταλήξαμε στο μοντέλο phi της Ollama, που είναι αρκετά lightweight σε σύγκριση με τα υπόλοιπα της Ollama, αλλά οριακά καλύπτονταν οι απαιτήσεις του σε μνήμη από το λάπτοπ που χρησιμοποιήσαμε. Επιπρόσθετα, δυσκολευτήκαμε με την εγκατάσταση διάφορων βιβλιοθηκών του LangChain framework, αλλά και άλλων λόγω μη συμβατών εκδόσεων. Συγκεκριμένα, έπρεπε να δημιουργήσουμε ένα python virtual environment με μία παλαιότερη έκδοση της python από αυτή που χρησιμοποιούσαμε στο υπόλοιπο project, ώστε να μπορέσουμε να χρησιμοποιήσουμε πολλές βιβλιοθήκες απαραίτητες για την υλοποίηση του chatbot, όπως το PyTorch και το Sentence Transformers.

Μειονεκτήματα και Περιορισμοί

Θεωρούμε ότι το σύστημα μας έχει αδυναμίες που οφείλονται κυρίως σε ανεπάρκεια πόρων του συστήματος μας, αλλά και στη δική μας, περιορισμένη χρονικά από το φόρτο εργασίας, ενασχόληση με τη συγκεκριμένη υλοποίηση. Εκτός από τους περιορισμούς στο σύστημα που περιγράψαμε, παρατηρείται κάποιες φορές το chatbot να μη μπορεί να διακρίνει τα ids των documents, ακόμη και τον αριθμό τους, παρόλο που οι απαντήσεις του συμβαδίζουν πλήρως με το περιεχόμενο τους. Άλλες φορές οι απαντήσεις του -και στις δύο εκδοχές του- ακολουθούν μία συλλογιστική αφύσικη που μπερδεύει περισσότερο το χρήστη, αντί να τον βοηθάει. Επιπλέον, θα μπορούσαμε να κάνουμε ίσως μία προεπεξεργασία στα κείμενα πριν τα τροφοδοτήσουμε στο chatbot, ώστε να έχουν μία πιο σαφή και κατανοητή δομή. Ίσως αυτό βελτίωνε την απόδοση του μοντέλου μας.