Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών

Ανάκτηση Πληροφορίας

Εργαστηριακή Άσκηση

Χειμερινό Εξάμηνο 2023

Νικόλαος Βούλγαρης, 1084626, st1084626@ceid.upatras.gr

Όλα τα αρχεία υπάρχουν και στο https://github.com/NickVoulg02/Information-Retrieval.
Από εκεί μπορείτε να εκτελέσετε και το colbert_test_link.ipynb αρχείο για το colBERT.
Στο παράρτημα παρατίθεται ο κώδικας από όλα τα .py αρχεία.

Ερώτημα 1 - Ανάγνωση και Επεξεργασία της Συλλογής

Η δομή ενός ανεστραμμένου αρχείου αποτελείται από το λεξικό και τις εμφανίσεις.
Το λεξικό αποτελείται από όλες τις μοναδικές λέξεις που εμφανίζονται στη συλλογή κειμένων. Προκειμένου να μειώσουμε τους χώρους της δομής ευρετηριοδότησης εξαλείφουμε τα stopwords. Στη δική μου υλοποίηση εξάλειψα και τους αριθμούς που εμφανίζονται στα κείμενα και μείωσα τις εγγραφές του λεξικού κατά 1000. Το λεξικό περιέχει το αναγνωριστικό της κάθε λέξης και ένα συνολικό αριθμό κειμένων που εμφανίζεται η λέξη.
Το ανεστραμμένο αρχείο αντιστοιχίζει μια λίστα εγγραφών σε κάθε λέξη. Οι εγγραφές αποτελούνται από το αναγνωριστικό του κείμενου που περιέχει την λέξη και τον αριθμό εμφανίσεων της λέξης στο συγκεκριμένο κείμενο.
Στο κώδικα υλοποιείται με την class InvertedIndex, που έχει ένα dictionary τύπου python dictionary, ένα dic_id ακέραιο και μια Inv_index λίστα. Ανοίγουμε όλα τα κείμενα του υποφάκελου docs και για καθένα εκτελείται η ακόλουθη διαδικασία. Σπάμε τις γραμμές του με χρήση splitlnes() συνάρτησης, χρησιμοποιούμε το όνομα του αρχείου ως id και κρατάμε σε file_len και number_of_docs μεταβλητές για το μέγεθος του αρχείου και τον αριθμό των αρχείων αντίστοιχα (μεταβλητές που θα αξιοποιηθούν αργότερα).
Καλείται Inverted_Index της κλάσης InvertedIndex. Κάθε λέξη αντιμετωπίζεται σαν token. Αν αποτελεί αριθμό ή αν ανήκει στη stopwords.words('english') ττης ntlk βιβλιοθήκης δεν λαμβάνεται υπόψιν. Κρατάμε τον αριθμό των token μετά το φιλτράρισμα που εμφανίζονται σε κάθε κείμενο στη μεταβλητή length, που θα χρησιμοποιηθεί αργότερα. Αν το token δεν ανήκει

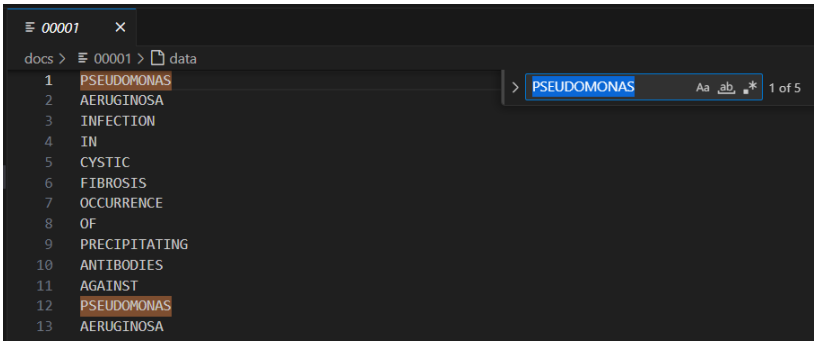
στο dictionary, προστίθεται εκεί και προστίθεται ένας πίνακας με το id και τον αριθμό 1, για τις εμφανίσεις στο κείμενο μέχρι τώρα, στο Inv_index. Αν ανήκει στο dictionary και πρόκειται για το ίδιο κείμενο που έχει εμφανιστεί ήδη τουλάχιστον μια φορά, απλά αυξάνουμε τον αριθμό εμφανίσεων της λέξης στο συγκεκριμένο κείμενο. Αλλιώς εισάγουμε στην εγγραφή για το συγκεκριμένο token άλλο ένα πίνακα 2 στοιχείων.
Ας δείξουμε ένα παράδειγμα για την πρώτη λέξη του πρώτου κειμένου, PSEUDOMONAS.

```
72   print('PSEUDOMONAS ID and NUMBER OF DOCS IT APPEARS IN')
73   id = inv.dictionary.get('PSEUDOMONAS')
74   print(id)
75   print('ID OF DOC IT APPEARS IN and NUMBER OF OCCURRENCES')
76   print(inv.Inv_index[id[0]])
77
```

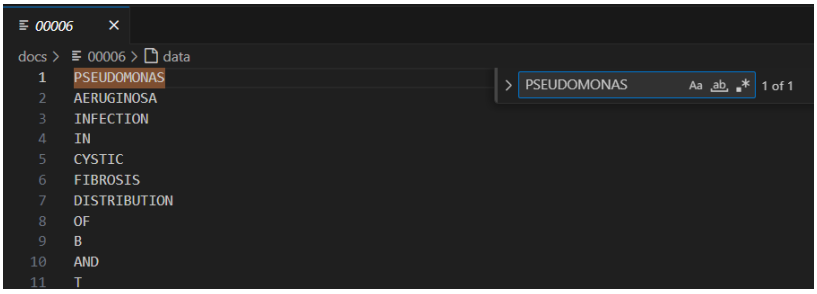
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\nickv\Documents\information_retrieval> & C:/Users/nickv/AppData/Local/Programs/Python/Python311/python.exe c:/Users/nickv/Documents/information_retrieval/vector_space_model.py
PSEUDOMONAS ID and NUMBER OF DOCS IT APPEARS IN
[0, 79]
ID OF DOC IT APPEARS IN and NUMBER OF OCCURRENCES
[[1, 5], [6, 1], [7, 3], [8, 1], [18, 2], [61, 2], [62, 5], [79, 2], [80, 2], [81, 5], [82, 4], [123, 4], [135, 1], [148, 1], [152, 1], [159, 2], [160, 1], [161, 2], [176, 2], [177, 3], [178, 2], [188, 5], [250, 1], [260, 2], [265, 2], [271, 1], [384, 2], [394, 1], [427, 3], [447, 2], [451, 1], [458, 2], [459, 2], [460, 1], [469, 2], [479, 2], [505, 1], [550, 2], [556, 3], [589, 1], [590, 4], [591, 1], [611, 1], [668, 2], [696, 1], [706, 1], [753, 2], [778, 9], [790, 1], [800, 3], [802, 2], [842, 1], [865, 1], [884, 2], [890, 3], [905, 7], [912, 3], [913, 2], [917, 1], [922, 1], [983, 1], [987, 2], [989, 1], [1000, 1], [1010, 2], [1042, 3], [1065, 5], [1071, 3], [1077, 2], [1086, 2], [1089, 2], [1090, 2], [1091, 3], [1112, 1], [1171, 1], [1180, 1], [1192, 1], [1203, 3], [1227, 2]]
PS C:\Users\nickv\Documents\information_retrieval>
```

Ελέγχουμε το πρώτο και το έκτο κείμενο σύμφωνα με τα αποτελέσματα του Inv_index.



6 εμφανίσεις στο πρώτο κείμενο



1 στο έκτο

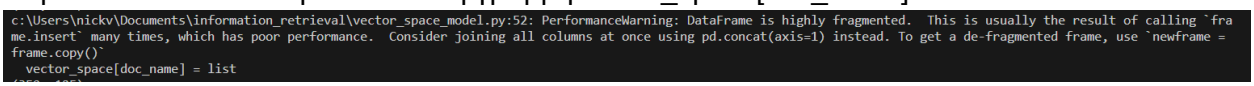
Μετά το τέλος της συνάρτησης στο file_len dictionary εισάγεται ένα στοιχείο με id αυτή του τρέχοντος κειμένου και τιμή το length που επιστράφηκε από την Inverted_Index()

Ερώτημα 2 - Υλοποίηση Vector Space μοντέλου

Το vector space είναι ένα μοντέλο ανάκτησης πληροφορίας για αναπαράσταση των κειμένων ως διανύσματα, με την απόσταση μεταξύ τους να αναπαριστά το πόσο σχετικά είναι τα 2 κείμενα. Το βάρος των όρων σε κάθε κείμενο χρησιμοποιείται για τον βαθμό ομοιότητας. Τα βάρη υπολογίζονται με χρήση της TF-IDF. Η Term Frequency υπολογίζει τον αριθμό εμφάνισης του όρου σε κάθε κείμενο. Η Inverse Document Frequency υπολογίζει τον αριθμό εμφάνισης του όρου σε όλα τα κείμενα της συλλογής, υπολογίζει ουσιαστικά τη σπανιότητα του.
Στην υλοποίηση μου τα βάρη idf υπολογίζονται με βάση τον τύπο απλής λογαριθμικής κανονικοποίησης $\log(1+N/ni)$, όπου N το πλήθος των κειμένων και ni ο αριθμός των κειμένων που εμφανίζεται το token.
Αποθηκεύουμε τα αποτελέσματα στην idf_values λίστα.
Τα βάρη tf υπολογίζονται με τον τύπο της απλής λογαριθμικής κανονικοποίησης log(1+fi,j), όπου fi,j το πηλίκο του αριθμού εμφανίσεων ενός token στο κείμενο προς το πλήθος των token μες στο κείμενο.
Σε κάθε κείμενο ελέγχουμε αν κάποιο token υπάρχει σε αυτό. Πρέπει να αναζητήσουμε ολόκληρη την εγγραφή στο Inv_index, έτσι ώστε να βρούμε τον αριθμό εμφανίσεων για τις τιμές που ταυτίζονται με το όνομα του κειμένου που εξετάζουμε. Αν το βρούμε υπολογίζουμε το tf βάρος, το πολλαπλασιάζουμε με το αντίστοιχο idf και τα αποθηκεύουμε σε μια λίστα. Αν δεν εμφανίζεται το token στο συγκεκριμένο κείμενο αποθηκεύουμε μηδέν.

Για κάθε κείμενο η λίστα αποθηκεύεται σε ένα pandas.Dataframe ονόματι vector_space. Στο τέλος εξάγεται σε ένα csv αρχείο, όπου κάθε στήλη αντιστοιχεί σε ένα document και κάθε γραμμή σε ένα token του dictionary.

Η γραμμή simplefilter(action="ignore", category=pd.errors.PerformanceWarning) αφορά το παρακάτω error που προκαλούσε η γραμμή vector_space[doc_name] = list



Εφόσον σκοπός της εργαστηριακής άσκησης δεν είναι η απόδοση όσο αφορά το χρόνο δημιουργίας του vector space, επέλεξα να αγνοήσω τις προειδοποιήσεις. Ο κώδικας εξάλλου εκτελείται σε λογικά χρονικά πλαίσια.

Εδώ είναι τα αποτελέσματα και η τελική μορφή του vector space model:

```
PS C:\Users\nickv\Documents\information_retrieval> & C:/Users/nickv/AppData/Local/Programs/Python/Python311/python.exe c:/Users/nickv/Documents/information_retrieval/ve
ctor_space_model.py
          doc1      doc2      doc3      doc4  doc5      doc6      doc7      doc8  ...  doc1232  doc1233  doc1234  doc1235  doc1236  doc1237  doc1238  doc1239
0     0.022994  0.000000  0.000000  0.000000   0.0  0.004004  0.010782  0.002764  ...  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1     0.023397  0.000000  0.000000  0.000000   0.0  0.027887  0.028770  0.013915  ...  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
2     0.017738  0.000000  0.000000  0.000000   0.0  0.015203  0.003474  0.000000  ...  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
3     0.002485  0.003052  0.003265  0.002600   0.0  0.002127  0.002875  0.002925  ...  0.009555  0.008512  0.004031  0.003900  0.006241  0.001101  0.002956  0.006311
4     0.002483  0.003050  0.003263  0.002598   0.0  0.002126  0.002873  0.002923  ...  0.009548  0.008506  0.004028  0.003898  0.006237  0.001100  0.002954  0.006307
...        ...       ...       ...       ...   ...       ...       ...       ...  ...       ...       ...       ...       ...       ...       ...       ...
10539 0.000000  0.000000  0.000000  0.000000   0.0  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.014168  0.000000
10540 0.000000  0.000000  0.000000  0.000000   0.0  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.014168  0.000000
10541 0.000000  0.000000  0.000000  0.000000   0.0  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.014168  0.000000
10542 0.000000  0.000000  0.000000  0.000000   0.0  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.014168  0.000000
10543 0.000000  0.000000  0.000000  0.000000   0.0  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.030429

[10544 rows x 1209 columns]
PS C:\Users\nickv\Documents\information_retrieval>
```

Εξάγεται ως vector_space2.csv

Ερώτημα 3 - Υλοποίηση colBERT μοντέλου

Στο ColBERT (Contextualized Late interaction over BERT), τα Queries και Documents κωδικοποιούνται ξεχωριστά σε contextual embeddings χρησιμοποιώντας δύο διαφορετικά μοντέλα BERT, γλωσσικά μοντέλα που βασίζονται στην αρχιτεκτονική των transformer. Τα contextual embeddings είναι απλά διανύσματα που παράγονται ως έξοδοι από τα μοντέλα BERT. Τα 2 σύνολα κωδικοποιήσεων (ένα σύνολο για το ερώτημα q και ένα άλλο σύνολο tokens για το έγγραφο d) επιτρέπεται να παρακολουθούν το ένα το άλλο και να υπολογίζουν μια βαθμολογία για το πόσο σχετικά είναι το ερώτημα με έγγραφο για κάθε ζεύγος. Το έγγραφο που επιτυγχάνει την υψηλότερη βαθμολογία συνάφειας για ένα ερώτημα λαμβάνει τη χαμηλότερη θέση και αντίστροφα. Με αυτόν τον τρόπο, κατατάσσουμε το σύνολο των εγγράφων.
Ένα προ-εκπαιδευμένο μητρώο ενσωμάτωσης χρησιμοποιείται για τη δημιουργία token για τα q και d.
Στην δική μου υλοποίηση χρησιμοποιώ το Google Colab Notebook, που αναφέρεται και στη εκφώνηση της άσκησης. Αφού εισαχθούν όλες οι κατάλληλες βιβλιοθήκες για το colBERT, εισάγω μέσω του προσωπικού μου GitHub repository 2 tsv αρχεία που αντιπροσωπεύουν τα αρχεία του φακέλου docs και των ερωτήσεων από το αρχείο Queries_20. Ο κώδικας για την δημιουργία αυτών των αρχείων περιλαμβάνεται στο αρχείο preprocess.py. Για κάθε αρχείο του φακέλου το αποθηκεύουμε ολόκληρο μαζί με μια στήλη που κρατά το id του. Προσθέτουμε και ένα header. Για τις ερωτήσεις από το Queries_20 κάνουμε παρόμοια διαδικασία αλλά δε συμπεριλαμβάνουμε header.
Δημιουργήθηκε collection Dataset της βιβλιοθήκης Hugging Face Transformers library μέσω pandas Dataframes που άνοιξε το doc_col αρχείο και query Queries που άνοιξε το queries_20.

```
#importing tsv files from personal github repository and creating docs dataset and queries
!git clone https://github.com/NickVoulg02/Information-Retrieval.git
import pandas as pd
from datasets import Dataset
dataset = 'test'
df1 = pd.read_csv("Information-Retrieval/colbert_test/doc_col.tsv", delimiter = '\t', index_col=0)
collection = Dataset.from_pandas(df1, preserve_index=True)
query = Queries("Information-Retrieval/colbert_test/queries_20.tsv")
f'Loaded {len(query)} queries and {len(collection):,} passages'
```

```
Cloning into 'Information-Retrieval'...
remote: Enumerating objects: 100, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 100 (delta 29), reused 15 (delta 15), pack-reused 59
Receiving objects: 100% (100/100), 562.16 KiB | 5.16 MiB/s, done.
Resolving deltas: 100% (38/38), done.
[Jan 25, 11:10:08] #> Loading the queries from Information-Retrieval/colbert_test/queries_20.tsv ...
[Jan 25, 11:10:08] #> Got 20 queries. All QIDs are unique.

'Loaded 20 queries and 1,209 passages'
```

```
print(query[0])
print(collection[0])
```

```
WHAT ARE THE EFFECTS OF CALCIUM ON THE PHYSICAL PROPERTIES OF MUCUS FROM CF PATIENTS
{'doc': 'PSEUDOMONAS AERUGINOSA INFECTION IN CYSTIC FIBROSIS OCCURRENCE OF PRECIPITATING ANTIBODIES AGAINST PSEUDOMONAS AERUG
```

Υπολογίζουμε την colBERT αναπαράσταση του κάθε κειμένου και τα δεικτοδοτούμε. Ο indexer χρησιμοποιεί το colbert checkpoint που χρησιμοποιείται για να κάνουμε search. Ψάχνοντας τα αποτελέσματα για την πρώτη ερώτηση τα αποτελέσματα είναι τα εξής.

```
question = query[0]

# Find the top-10 passages for this query
results = searcher.search(question, k=20)
#print(results)

passages_ranked = {}
#Print out the top-k retrieved passages
print("Rank\tScore\tId\tPassage")
for passage_id, passage_rank, passage_score in zip(*results):
    print(f"[{passage_rank}] \t{passage_score:.1f} \t{collection['doc_id'][passage_id]} \t{searcher.collection[passage_id]}")
    passages_ranked[passage_id] = passage_rank
```

```
Rank    Score   Id      Passage
[1]     23.0    533     EFFECTS OF CALCIUM ON INTESTINAL MUCIN IMPLICATIONS FOR CYSTIC FIBROSIS A MAJOR FEATURE OF THE DISEASE CYSTIC FIBROSIS IS THE EXCESSIVE CONCENTRATION OF MUC
[2]     20.4    441     WATER AND ELECTROLYTES OF EXOCRINE SECRETIONS PP 17991 IT HAS BEEN RECOGNIZED THAT THE LEVELS OF WATER AND ELECTROLYTES ARE DERANGED IN MANY IF NOT ALL OF T
[3]     19.5    957     THE BIOLOGIC ACTIVITIES OF CYSTIC FIBROSIS SERUM II ULTRASTRUCTURAL ASPECTS OF THE EFFECT OF CYSTIC FIBROSIS SERA AND CALCIUM IONOPHORE A23187 ON RABBIT TRA
[4]     19.2    484     CALCIUM FLUX AND CYSTIC FIBROSIS LETTER IN AN EFFORT TO DETERMINE WHETHER INCREASED PERMEABILITY TO CALCIUM IONS COULD EXPLAIN THE CYSTIC FIBROSISSERUM CIL
[5]     18.6    741     THE BIOLOGIC ACTIVITIES OF CYSTIC FIBROSIS SERUM I THE EFFECTS OF CYSTIC FIBROSIS SERA AND CALCIUM IONOPHORE A 23187 ON RABBIT TRACHEAL EXPLANTS AN IONOPHOR
[6]     18.5    827     THE INFLUENCE OF CYSTIC FIBROSIS SERUM AND CALCIUM ON SECRETION IN THE RABBIT TRACHEAL MUCOCILIARY APPARATUS CYSTIC FIBROSIS SERUM OR ITS ISOLATED COMPONENT
[7]     17.9    505     RESEARCH IN CYSTIC FIBROSIS THIRD OF THREE PARTS RECENT AUTONOMIC NERVOUS SYSTEM STUDIES IN CYSTIC FIBROSIS ARE OUTLINED MISCELLANEOUS OBSERVATIONS REGARDIN
[8]     17.8    960     CALCIUM AND SODIUM TRANSPORT PROCESSES IN PATIENTS WITH CYSTIC FIBROSIS I A SPECIFIC DECREASE IN MG2DEPENDENT CA2ADENOSINE TRIPHOSPHATASE ACTIVITY IN ERYTHR
[9]     17.7    1201    ALTERED INTRACELLULAR CALCIUM IN FIBROBLASTS FROM PATIENTS WITH CYSTIC FIBROSIS AND HETEROZYGOTES THE IMPORTANCE OF INTRACELLULAR CALCIUM CA IN SECRETION AN
[10]    17.4    437     PULMONARY ASPECTS OF CYSTIC FIBROSIS PP 324 DEFECTIVE MUCOCILIARY TRANSPORT HAS BEEN IMPLICATED IN THE PATHOGENESIS OF THE LUNG DISEASE IN CYSTIC FIBROSIS B
[11]    17.1    754     EFFECTS OF GRAVITY ON TRACHEAL MUCUS TRANSPORT RATES IN NORMAL SUBJECTS AND IN PATIENTS WITH CYSTIC FIBROSIS A NONINVASIVE RADIONUCLIDE IMAGING TECHNIQUE FO
[12]    17.1    1185    MITOCHONDRIAL CALCIUM UPTAKE AND OXYGEN CONSUMPTION IN CYSTIC FIBROSIS REPORTS OF EXOCRINE GLAND SECRETORY ANOMALIES IN CYSTIC FIBROSIS AND INCREASED CALCIU
[13]    17.0    499     PATHOPHYSIOLOGY OF MUCUS SECRETION IN CYSTIC FIBROSIS ABNORMALITY OF MUCUS SECRETION EITHER BIOCHEMICAL OR PHYSICAL HAS LONG BEEN CLAIMED TO BE THE PRIMARY
[14]    17.0    1107    REDUCED SERUM 25HYDROXYVITAMIN D CONCENTRATION AND DISORDERED MINERAL METABOLISM IN PATIENTS WITH CYSTIC FIBROSIS VITAMIN D AND MINERAL METABOLISM WERE STUD
[15]    16.9    139     PURIFICATION AND PROPERTIES OF THE CALCIUMPRECIPITABLE PROTEIN IN SUBMAXILLARY SALIVA OF NORMAL AND CYSTIC FIBROSIS SUBJECTS A CALCIUMPRECIPITABLE PROTEIN C
[16]    16.8    501     IMPAIRMENT OF MUCOCILIARY TRANSPORT IN CYSTIC FIBROSIS OVER THE PAST FEW YEARS STUDIES FROM VARIOUS LABORATORIES HAVE SUGGESTED THE PRESENCE OF A FACTORS IN
[17]    16.8    564     EFFECTS OF CYSTIC FIBROSIS SERUM CILIARY INHIBITOR ON OYSTER GILL ULTRASTRUCTURE ANALYSIS BY SCANNING AND TRANSMISSION ELECTRON MICROSCOPY ULTRASTRUCTURAL A
[18]    16.6    52      ERYTHROCYTE MEMBRANE PROPERTIES IN CYSTIC FIBROSIS THE BIOCHEMICAL PROPERTIES OF THE ERYTHROCYTE MEMBRANE HAVE BEEN STUDIED IN CYSTIC FIBROSIS PATIENTS AND
[19]    16.6    147     LETTER RATES AND CALCIUM CONCENTRATIONS OF SWEAT IN RELATION TO CYSTIC FIBROSIS THE AUTHORS OF THE RECENT ARTICLE WHICH FAILED TO SHOW A SIGNIFICANT DIFFERE
[20]    16.4    568     CULTURED CELLS IN CYSTIC FIBROSIS RESEARCH A REVIEW CYSTIC FIBROSIS CF IS A COMMON INHERITED DISORDER WHICH IS CHARACTERIZED BY THE PRODUCTION OF EXOCRINE S
```

Ερώτημα 4 - Συγκρίσεις

Όσο αφορά το vector space μοντέλο, για να εξετάσουμε την απόδοση του με βάσει μετρικές αξιολόγησης θα πρέπει να εισάγουμε διανύσματα για τις ερωτήσεις από το Queries_20 αρχείο μέσα στο διανυσματικό χώρο. Η διαδικασία είναι παρόμοια με αυτή που κάναμε για τα κείμενα του docs, το βάρος idf παραμένει το ίδιο για κάθε token που περιλαμβάνεται μέσα σε ένα query.

Τα tf βάρη πρέπει να υπολογιστούν για την συχνότητα των token μέσα σε κάθε query και το τελικό αποτέλεσμα αποθηκεύεται στο query_vec τύπου pandas.Dataframe.
Εφόσον έχουμε στο vector space διανύσματα για τα κείμενα και για τα ερωτήματα αρκεί να βρούμε την cosine similarity των διανυσμάτων, προκειμένου να βρούμε ποιο κείμενο είναι σχετικότερο με ποιο ερώτημα. Το cosine similarity 2 διανυσμάτων ισούται με το πηλίκο του εσωτερικού γινομένου και γινομένου των μέτρων τους. Στην υλοποίηση υπολογίζεται με χρήση των dot() και norm() συναρτήσεων της numpy βιβλιοθήκης και το αποτέλεσμα αποθηκεύεται στη retrieved_docs λίστα για κάθε ερώτημα.
Εφόσον έχουμε μεθόδους που επιστρέφουν τα κείμενα σε μια διατεταγμένη ως προς την ομοιότητα λίστα, επέλεξα να χρησιμοποιήσω 2 μετρικές αξιολόγησης που είναι order aware και ελέγχουν αν τα σχετικά κείμενα είναι αυτά που κατατάσσονται υψηλότερα. Η Mean Average precision αξιολογεί κατά πόσον όλα τα στοιχεία που επιλέγονται από το μοντέλο ταυτίζονται με τα κείμενα που απορυθμίζονται στο Relevant_20 και κατατάσσονται υψηλότερα ή όχι. Η Mean Reciprocal Rank αξιολογεί κατά πόσο επιστρέφει το καλύτερο σχετικό στοιχείο και θέλουμε αυτό το στοιχείο να βρίσκεται σε υψηλότερη θέση.
Ο τύπος για το Average Precision at k είναι

$$\mathrm{AP@}n = \frac{1}{\mathrm{GTP}} \sum_{k}^{n} \mathrm{P@}k \times \mathrm{rel@}k$$

AP@n formula

Όπου GTP ο αριθμός των σχετικών κειμένων, P@k το precision at k δηλαδή το πηλίκο του αριθμού σχετικών ανακτηθέντων κειμένων ως το προς k σημείο που έχουμε φτάσει στην κατάταξη και rel@k το relevance score για το κάθε κείμενο. Στην δική μας περίπτωση το relevance score είναι 1 αν το κείμενο είναι σχετικό, αλλιώς μηδέν. Αφού βρούμε το AP για κάθε κείμενο η μέση ακρίβεια σε πολλά ερωτήματα έχει τύπο
MAP = sum(AP(i))/Q.

Ο τύπος για το Mean Reciprocal Rank είναι

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

Όπου Q το πλήθος των queries και rank i η πρώτη θέση που βρίσκουμε κείμενο που είναι relevant μέσα στα κείμενα που επιστρέφει το μοντέλο.

Όσο αφορά το MAP στο Vector Space μοντέλο, για κάθε query υπολογίζουμε το cosine similarity ανάμεσα στο διάνυσμα αυτού με κάθε διάνυσμα κειμένου. Από τα αποτελέσματα κρατάμε τα πρώτα 100 για κάθε ερώτημα. Ελέγχουμε αν τα κείμενα που έχουν ανακτηθεί ανήκουν στα relevant που μας δίνονται. Τα αποτελέσματα αποθηκεύονται στη precision_at_k λίστα και αθροίζονται στο average_precision. Τα average_precision των κειμένων αποθηκεύονται στη avg_pr_list για να υπολογιστεί η mean_average_precision. Η διαδικασία είναι παρόμοια για το colBERT, χρησιμοποιούμε το passages_ranked dictionary, όπου για κάθε id κειμένου κρατάμε το rank του στα top-k αποτελέσματα.

```
121     mean_average_precision = sum(avg_pr_list)/length
122     print("Mean Average Precision:" + str(mean_average_precision))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
0.1450194536629586
0.3
0.11428067256029885
0.2567697876436131
0.14491833750384747
0.19269624336631108
0.11946497651284885
0.026515151515151512
0.2593752176941832
0.3576437889595785
0.5702232807346811
0.07873650047563092
0.20663674621870262
0.1301334043966274
0.17741114315577405
0.0895189345348637
0.11827302469698973
0.4543323404607048
0.04139544163229281
0.4470752400002268
Mean Average Precision:0.21152098428626429
```

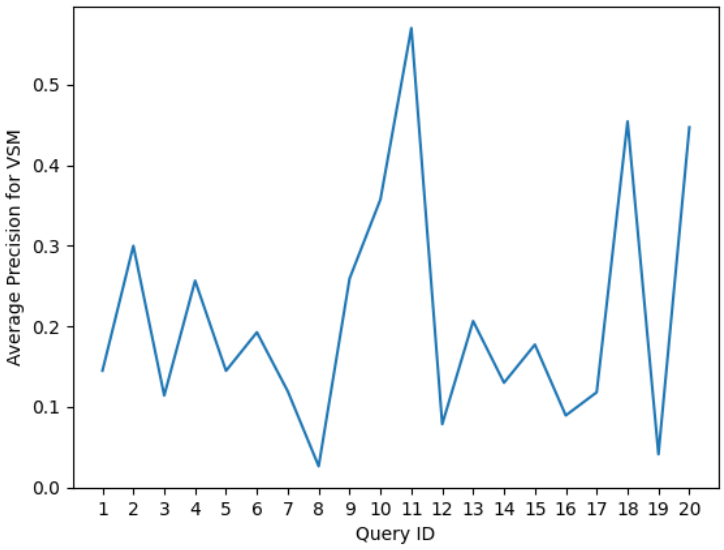
Αποτελέσματα MAP στο VSM

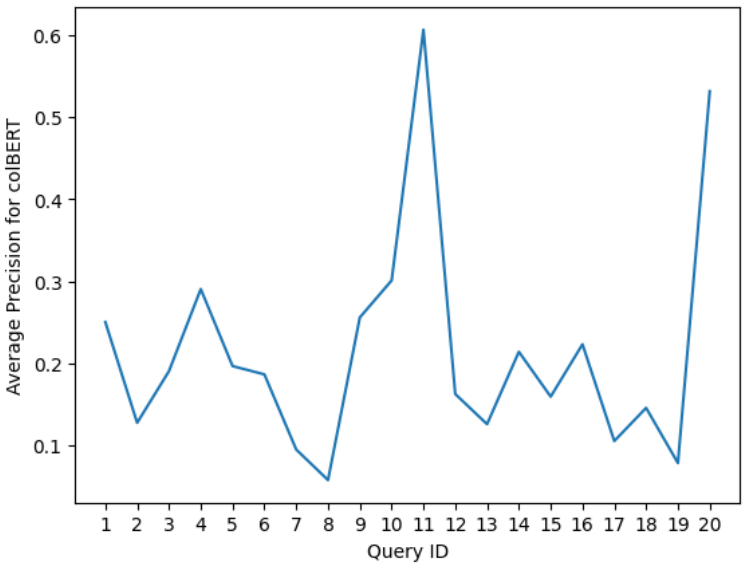
```
mean_average_precision = sum(avg_pr_list)/20
print("Mean Average Precision: " + str(mean_average_precision))

plt.xticks(numpy.arange(len(avg_pr_list)), numpy.arange(1, len(avg_pr_list)+1))
ypoints = numpy.array(avg_pr_list)
plt.xlabel("Query ID")
plt.ylabel("Average Precision")
plt.plot(ypoints)
plt.show()
```

```
0.25025136218216215
0.1275958667167708
0.19042972362417085
0.2904289240181963
0.19663148346453152
0.18654405298509594
0.09508615757294149
0.057845607349380226
0.2558358131230472
0.30091203317584414
0.6061873311125703
0.16263126778194076
0.12594239063843188
0.21404384548037259
0.15943431927260845
0.22319432312345341
0.1053961082824507
0.14575299700765537
0.07857902536760272
0.5313826644821248
Mean Average Precision: 0.21520526483806762
```

Αποτελέσματα MAP στο colBERT

Παρακάτω φαίνονται 2 pyplot για σύγκριση των αποτελεσμάτων των 2 μεθόδων.

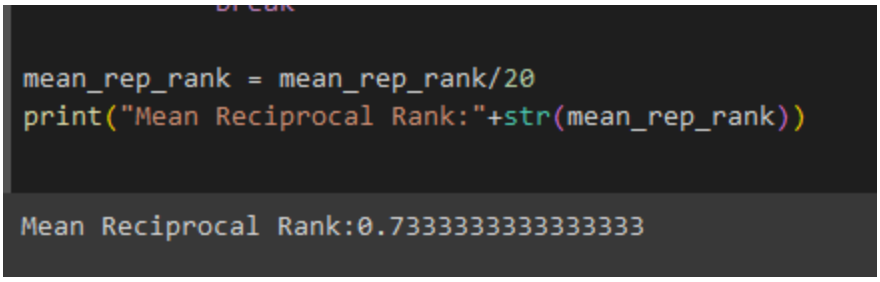




Όσο αφορά το MRR στο Vector Space μοντέλο, για κάθε query κρατάμε τη θέση του πρώτου relevant κειμένου που ανακτήθηκε στη μεταβλητή mean_rep_rank.

```
PS C:\Users\nickv\Documents\information_retrieval> & C
trics.py
Mean Reciprocal Rank:0.806547619047619
PS C:\Users\nickv\Documents\information_retrieval>
```

MRR αποτελέσματα στο VSM



MRR στο colBERT

Παρατηρούμε πως το Vector Space μοντέλο επιστρέφει καλύτερα αποτελέσματα σε σχέση με το colBERT, αλλά βρίσκονται σχετικά κοντά. Είναι πιθανό να υπερτερεί λόγω της μορφής των ερωτημάτων. Στηριζόμαστε στις εμφανίσεις των όρων μες στο κείμενο και όχι στην σημασιολογική σχέση όρων που έχουν κάποιο context. Αν το ζήτημα ήταν να επιστρέψουμε αποτελέσματα που θα ταιριάζουν και σημασιολογικά με τις ερωτήσεις το colBERT σίγουρα θα απέδιδε καλύτερα.

Αναφορές

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

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

Breaking Down Mean Average Precision (mAP), Ren Jie Tan, https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52

Mean reciprocal rank, https://en.wikipedia.org/wiki/Mean_reciprocal_rank

Παράρτημα

Inverted_index.py

```
import os
import nltk
from nltk.corpus import stopwords

#nltk.download('stopwords')
stopping = []
for x in stopwords.words('english'):        # A stop word is a commonly used word that a search engine has been programmed to ignore
    stopping.append(x.upper())

class InvertedIndex:
```

```
    def __init__(self):
        self.dictionary = {}              # a dictionary of all the words used in the articles, along with
an id
                                  # and a number of total articles they appear in
        self.dic_id = -1                  # used for the token ids in the dictionary
        self.Inv_index = []               # contains article id and number of occurences of token in said
article

    def Tokenization(self, doc):
        words= []
        for x in range(len(doc)):
            if doc[x].isdigit() == False:                          # removing every number from
dictionary
                words.append(doc[x])

        return words

    def Inverted_Index(self, id, doc):
        length=0                                                   # length variable only considers terms that
aren't stopwords
        tokenlist = self.Tokenization(doc)                         # a list of every word in the article
        for x in range(len(tokenlist)):
            if tokenlist[x] not in stopping:                       # checking for stopwords
                length+=1
                if self.dictionary.get(tokenlist[x]) == None:
                    self.dic_id+=1
                    self.dictionary[tokenlist[x]] = [self.dic_id, 1]
                    self.Inv_index.append([[id , 1]])

                elif self.dictionary.get(tokenlist[x]) != None:
                    token_id = self.dictionary.get(tokenlist[x])[0]
                    if self.Inv_index[token_id][-1][0] == id:          # checking if we're still in the same
article by article id
                        temp1 = self.Inv_index[token_id][-1]
                        temp1[1]+=1
                    else:
                        self.dictionary.get(tokenlist[x])[1]+=1
                        self.Inv_index[token_id].append([id , 1])

        return length

def main(inv, file_len, number_of_docs):
    cwd = os.getcwd()                                  # open working directory
    cwd = cwd + "\docs"
```

```
    #print(cwd)
    for filename in os.listdir(cwd):
        with open(os.path.join(cwd, filename), 'r') as f:     # open in readonly mode
            file = f.read().splitlines()
            id = int(filename)                              # id depends on filename
            number_of_docs+=1
            file_len[id] = inv.Inverted_Index(id, file)

    return number_of_docs



vector_space_model.py
import math
from warnings import simplefilter
import pandas as pd
import inverted_index

simplefilter(action="ignore", category=pd.errors.PerformanceWarning)

def idf_calculation(doc_num, term_dic):
    idf_result = []

    for value in term_dic.items():
        #print(value)                                       #value[1][1] is the number of documents
where the term appears
        idf_result.append(math.log10((doc_num / value[1][1]) + 1))



    return idf_result

inv = inverted_index.InvertedIndex()
file_len = {}
number_of_docs = 0

number_of_docs = inverted_index.main(inv, file_len, number_of_docs)

idf_values = idf_calculation(number_of_docs, inv.dictionary)  # calculate idf values for every
token (how rare it is)
#print(idf_values)

vector_space = pd.DataFrame()  # vector space consisting of tokens and documents
# filling the vector space
for doc in file_len.items():
```

```
    #print(doc)
    list = []
    for token in inv.dictionary.items():
        detected = 0
        id = token[1][0]                            # document id
        term = inv.Inv_index[id]                        # Inv_index record for the specific token
        for i in range(len(term)):                     # record for every doc the token appears in
            if term[i][0] == doc[0]:                    # checking if it appears in the doc we're currently
checking
                # print(token[0])
                # print(doc[1])
                # print(term[i][1])                     # occurences in specific file
                x1 = math.log10((term[i][1] / doc[1])+1)    #logarithmically scaled frequency
                #print(x1)
                x2 = idf_values[id]
                # print(x2)
                list.append(x1 * x2)
                # print(x1*x2)
                detected = 1
        if detected == 0:
            list.append(0)
    doc_name = "doc" + str(doc[0])
    vector_space[doc_name] = list

print(vector_space)
vector_space.to_csv('vector_space2.csv', encoding='utf-8')



metrics.py
avg_pr_list = []
    length = len(query_vec.columns[0:])

    for i in range(length):
        retrieved_docs = cosine_similarity(vector_space, query_vec.iloc[:,i])       #first column
        retrieved_docs.sort(key=lambda ret:ret[1], reverse=True)
        retrieved_docs_filtered = retrieved_docs[:100]
        # for x in range(len(retrieved_docs_filtered)):
        #     print(retrieved_docs_filtered[x][0])

        precision_at_k = []
        true_positives = 0

        for doc in retrieved_docs_filtered:
            if(doc[0] in relevant_docs[i]):
```

```
                true_positives+=1
                precision_at_k.append(true_positives/(retrieved_docs_filtered.index(doc)+1))

        #print(precision_at_k)

        average_precision = 0
        for x in range(len(precision_at_k)):
                average_precision += precision_at_k[x]

        average_precision = average_precision/len(relevant_docs[i])
        print(average_precision)

        avg_pr_list.append(average_precision)

    mean_average_precision = sum(avg_pr_list)/length
    print("Mean Average Precision:" + str(mean_average_precision))
    plt.xticks(numpy.arange(len(avg_pr_list)), numpy.arange(1, len(avg_pr_list)+1))
    ypoints = numpy.array(avg_pr_list)
    plt.xlabel("Query ID")
    plt.ylabel("Average Precision for VSM")
    plt.plot(ypoints)
    plt.show()

# #Mean Reciprocal Rank
    # mean_rep_rank = 0
    # length = len(query_vec.columns[0:])
    # list=[]
    # for i in range(length):

    #     retrieved_docs = cosine_similarity(vector_space, query_vec.iloc[:,i])       #first column
    #     retrieved_docs.sort(key=lambda ret:ret[1], reverse=True)
    #     #retrieved_docs_filtered = retrieved_docs[:250]
    #     #print(retrieved_docs_filtered)

    #     for doc in retrieved_docs:
    #         if(doc[0] in relevant_docs[i]):
    #             value = retrieved_docs.index(doc)+1
    #             #print(value)
    #             break
    #     mean_rep_rank += 1/value
    #     list.append(mean_rep_rank)

    # mean_rep_rank = mean_rep_rank/length
    # print("Mean Reciprocal Rank:" + str(mean_rep_rank))
```

Preprocess.py

```
import os
import csv

cwd = os.getcwd()                                   # open working directory
docs = cwd + "\docs"

with open(cwd+"/doc_col.tsv", 'wt', newline='', encoding='utf-8') as out_file:
    tsv_writer = csv.writer(out_file, delimiter='\t')
    tsv_writer.writerow(["doc_id", "doc"])
    for filename in os.listdir(docs):
        with open(os.path.join(docs, filename), 'r') as f:     # open in readonly mode
            file = f.read().splitlines()
            passage = " "
            passage = passage.join(file)
            id = int(filename)
            tsv_writer.writerow([id, passage])

with open(cwd+"/queries_20.tsv", 'wt', newline='\n', encoding='utf-8') as out_file:
    tsv_writer = csv.writer(out_file, delimiter='\t')
    with open(os.path.join(cwd, 'Queries_20'), 'r') as f:     # open in readonly mode
            file = f.read().splitlines()
            id=0
            for x in file:
                x = x.upper()
                tsv_writer.writerow([id, x])
                id+=1
```