Αλέξανδρος Ξιάρχος
st1059619@ceid.upatras.gr
1059619

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ · ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΚΤΗΣΗ ΠΛΗΡΟΦΟΡΙΑΣ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ · 2023 – 2024

ΠΕΡΙΕΧΟΜΕΝΑ

1 ΕΙΣΑΓΩΓΗ

1.1 ΕΠΙΛΟΓΗ ΣΥΣΤΗΜΑΤΩΝ ΣΤΑΘΜΙΣΗΣ TF-IDF

Καταρχάς πρέπει να επιλέξουμε τα δύο συστήματα στάθμισης των βαρών για τα διανύσματα που θα χρησιμοποιήσουμε.

1.1.1 ΠΡΩΤΟ ΣΥΣΤΗΜΑ ΣΤΑΘΜΙΣΗΣ

Το πρώτο σύστημα στάθμισης θα είναι μια παραλλαγή[1] του προτεινόμενου ως καλύτερου πλήρως σταθμισμένου συστήματος σύμφωνα με τους Salton-Buckley[2] (best fully weighted system). θα χρησιμοποιήσουμε την **απλή συχνότητα εμφάνισης** (raw term frequency) για το TF βάρος των εγγράφων,

$$\textbf{Σύστημα \#1:}\ \mathrm{TF}_{\text{εγγράφων}} = f_{i,j}$$

όπου f_{ij} οι φορές που ο όρος εμφανίζεται σε ένα έγγραφο, τη **διπλή 0,5 κανονικοποίηση** για το TF βάρος των ερωτημάτων (augmented normalized TF),

$$\textbf{Σύστημα \#1:}\ \mathrm{TF}_{\text{ερωτημάτων}} = 0.5 + 0.5\frac{f_{i,j}}{max_i\ f_{i,j}}$$

και τέλος την **απλή ανάστροφη συχνότητα εμφάνισης** για το IDF βάρος και των εγγράφων και των ερωτημάτων:

$$\textbf{Σύστημα \#1:}\ \mathrm{IDF}_{\substack{\text{εγγράφων}\\ \text{ερωτημάτων}}} = \log\frac{N}{n_i}$$

όπου N το πλήθος των εγγράφων και n_i ο αριθμός των εγγράφων στα οποία εμπεριέχεται ο όρος.

1.1.2 ΔΕΥΤΕΡΟ ΣΥΣΤΗΜΑ ΣΤΑΘΜΙΣΗΣ

Ως δεύτερο σύστημα στάθμισης θα χρησιμοποιήσουμε το καλύτερα σταθμισμένο πιθανολογικό σύστημα σύμφωνα με τους Salton-Buckley[1] (best weighted probabilistic weight) με

$$\textbf{Σύστημα \#2:}\ \text{βάρος όρου}_{\text{εγγράφων}} = 0.5 + 0.5\frac{f_{i,j}}{max_i\ f_{i,j}}$$

$$\textbf{Σύστημα \#2:}\ \text{βάρος όρου}_{\text{ερωτημάτων}} = \log\frac{N - n_i}{n_i}.$$

Και τα δύο αυτά συστήματα στάθμισης έχουν επιφέρει τα ακριβέστερα αποτελέσματα και στο σύνολο των συλλογών στα οποία έχουν εξεταστεί αλλά και ειδικότερα σε ιατρικές (MED) συλλογές. Επομένως, συνολικά έχουμε:

[1] Το σύστημα αναφέρεται ως παραλλαγή των Salton-Buckley για το λόγο ότι δεν έχει συμπεριληφθεί κάποιος παράγοντας κανονικοποίσης, μιας και τα έγγραφα είναι *περίπου* ισομεγέθη (μέσος όρος 350 λέξεις).

[2] Gerard Salton, Christopher Buckley, Term-weighting approaches in automatic text retrieval, Information Processing & Management, Volume 24, Issue 5, 1988, Pages 513-523, ISSN 0306-4573

Σύστημα στάθμισης	Βάρος όρου εγγράφου	Βάρος όρου ερωτήματος
1	$f_{i,j} \times \log \frac{N}{n_i}$	$(0.5 + 0.5\frac{f_{i,j}}{max_i\ f_{i,j}}) \times \log \frac{N}{n_i}$
2	$0.5 + 0.5\frac{f_{i,j}}{max_i\ f_{i,j}}$	$\log \frac{N - n_i}{n_i}$

2 ΥΛΟΠΟΙΗΣΗ ΜΟΝΤΕΛΩΝ

Η εργασία υλοποιήθηκε σε Python χρησιμοποιώντας τις βιβλιοθήκες:

Βιβλιοθήκη	Περιγραφή
os	σύνδεση με λειτουργικό σύστημα
json	αποθήκευση-ανάγνωση JSON αρχείων
nltk	αφαίρεση stopwords, stemming
numpy	υπολογισμός ομοιότητας συνημιτόνου
math	υπολογισμός λογαρίθμων
matplotlib	γραφικές παραστάσεις

Αυτή είναι η δομή των αρχείων κώδικα όπου έχει χωριστεί η υλοποίηση:

```
src
├─ main.py
├─ tools.py
├─ preprocessing.py
├─ inverted_index.py
└─ vector_space_model.py
```

```
src
├─ colBERT_preprocessing.py
├─ colBERT.ipynb
└─ evaluation_metrics.py
```

Τέλος, η υλοποίηση του colBERT μοντέλου έχει πραγματοποιηθεί στο Google Colab ως Juputer Notebook.

2.1 ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ ΕΓΓΡΑΦΩΝ & ΒΟΗΘΗΤΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

2.1.1 `tools.py`

Το αρχείο `tools.py` περιλαμβάνει βοηθητικές συναρτήσεις για κάποιες επαναλαμβανόμενες διαδικασίες της υλοποίησης.

Η συνάρτηση **`get_docs`**`()`, χρησιμοποιώντας την `os` βιβλιοθήκη, διαβάζει το πλήθος των αρχείων της συλλογής.[3] Η συνάρτηση, αφού αφαιρέσει το escape character `'\n'`, το οποίο προκύπτει από την μορφολογία

[3] Να σημειωθεί ότι το πλήθος των εγγράφων διαφέρει από την αύξουσα αρίθμησή τους. Συγκεκριμένα έχουμε 1209 έγγραφα αριθμημένα από το `000001` ως `01239`. Με άλλα λόγια υπάρχουν αριθμοί στη συλλογή που δεν αντιστοιχούν σε έγγραφα. Συνεπώς δεν θα μπορούσαμε να χρησιμοποιήσουμε κάποια αριθμητική επανάληψη, για παράδειγμα, για την εισαγωγή των εγγράφων.

των εγγράφων (η κάθε λέξη είναι τοποθετημένη σε διαφορετική γραμμή), δημιουργεί και επιστρέφει μια λίστα από tuples, με κάθε tuple να αντιστοιχεί σε κάθε αρχείο-έγγραφο. Έτσι τα έγγραφα έχουν τη δομή:

```
doc = ('docID', ['λήμμα_1', 'λήμμα_2' ...])
```

όπου `docID` η αρίθμηση του κάθε εγγράφου και `λήμμα_n` η κάθε λέξη-λήμμα του εγγράφου. Αυτός θα είναι ο τρόπος αποθήκευσης και προσπέλασης των εγγράφων σε όλη την υλοποίηση.

Η συνάρτηση `strip()` της `get_docs()` είναι απαραίτητη για την αφαίρεση των `\n` χαρακτήρων που προκύπτουν από τη μορφολογία των εγγράφων (μιας και κάθε λέξη είναι σε νέα γραμμή). Με παρόμοιο τρόπο η συνάρτηση **`get_queries`**`()` επιστρέφει τη λίστα με τα ερωτήματα της συλλογής, η **`get_relevant`**`()` τη λίστα με τα σχετικά έγγραφα ανά ερώτημα και η **`get_json_file`**`()` τα περιεχόμενα ενός JSON αρχείου.

2.1.2 `preprocessing.py`

Στο αρχείο `preprocessing.py`, και συγκεκριμένα στις συναρτήσεις **`preprocess_collection`**`()` και **`preprocess_queries`**`()` πραγματοποιείται η προεπεξεργασία των εγγράφων, συγκεκριμένα η αφαίρεση των stopwords και το stemming.

Χρησιμοποιούμε τη `nltk` βιβλιοθήκη. Κάθε λέξη από τη συλλογή των εγγράφων αφού περάσουν από τον `PorterStemmer` της `nltk` και ελεγχθούν ότι δεν ανήκουν στη λίστα των stopwords, επιστρέφονται σε παρόμοια μορφή όπως δημιουργήθηκαν στη `get_docs()`. Αντίστοιχη διαδικασία πραγματοποιείται και για την προεπεξεργασία των ερωτημάτων, στη `preprocess_queries()`.

2.2 ΑΝΕΣΤΡΑΜΜΕΝΟ ΕΥΡΕΤΗΡΙΟ

Το ανεστραμμένο ευρετήριο δημιουργείται στη συνάρτηση **`create_inverted_index`**`()` του αρχείου `inverted_index.py`. Τα έγγραφα, μετά την προεπεξεργασία τους, εισέρχονται σε μια δομή επανάληψης, η οποία δημιουργεί το ανεστραμμένο ευρετήριο ως μορφή λεξικού ως εξής:

```
inverted_index['λήμμα'] =
{('docID_1': <φορές εμφάνισης>), ('docID_2': <φορές εμφάνισης>), ... }
```

Κάθε value του dictionary είναι ένα σύνολο[4] το οποίο περιλαμβάνει ένα ή περισσότερα tuples με το `docID` και τη συχνότητα εμφάνισης του λήμματος στο συγκεκριμένο έγγραφο. Η συχνότητα υπολογίζεται μέσω της `count()` σε όλο το έγγραφο ανά λήμμα.

Αυτό είναι ένα παράδειγμα του τελικού ανεστραμμένου ευρετηρίου:

```
inverted_index = { ... 'coronari': {('01217', 2), ('00779', 1)},
    'mobil': {('00673', 2)}, 'strain': {('00179', 7)}, ... }
```

2.3 ΥΛΟΠΟΙΗΣΗ VECTOR SPACE ΜΟΝΤΕΛΟΥ

Η υλοποίηση του Vector Space μοντέλου γίνεται στο αρχείο `vector_space_model.py`. Η συνάρτηση **`run_vsm`**`(doc_collection, queries, inverted_index)`, μέσω μιας επανάληψης ώστε να καλυφθούν όλα τα queries, καλεί την συνάρτηση **`vsm`**`(doc_collection, query, inverted_index, model_type)`, όπου `model_type = '1'` ή `'2'`, ανάλογα με το σύστημα στάθμισης που θέλουμε να χρησιμοποιήσουμε.

Η συνάρτηση `vsm()` αρχικά υπολογίζει τις συχνότητες $f_{i,j}$ κάθε όρου του εκάστοτε ερωτήματος:

[4] Έχει επιλεχθεί set για εξοικονόμιση μνήμης, μας και δεν μας ενδιαφέρει η σειρά των tuples.

```
query_tfs len(query) = { ... 'calcium': 1, 'effect': 2 ... }
```

Στη συνέχεια διατρέχεται κάθε όρος από το ερώτημα `query`. Αν ο όρος του ερωτήματος υπάρχει στο ανεστραμμένο ευρετήριο, υπολογίζεται η IDF τιμή του. και στη συνέχεια το TF-IDF βάρος του ερωτήματος, όπου θα αποθηκευτεί στο λεξικό `query_tfidfs`.

```
query_ifidf len(query) = { ... 'calcium': 3.6318, 'effect': 1.76483 ... }
```

Στη συνέχεια, για κάθε όρο του ερωτήματος, διατρέχουμε όλα τα έγγραφα της συλλογής. Αν το έγγραφο περιλαμβάνει τον όρο του ερωτήματος που εξετάζουμε (κάτι που ελέγχουμε μέσω της `docs_containing_term`), τότε μέσω του ανεστραμμένου ευρετηρίου, αποθηκεύεται η TF τιμή του, υπολογίζεται το TF-IDF βάρος του εγγράφου και αποθηκεύεται στο `doc_tfidfs[docID]`. Αν το έγγραφο δεν περιλαμβάνει τον όρο, αποθηκεύουμε 0.

```
doc_ifidf len(doc_collection) = {'000001': [0, 0, 0.0, 1.76, 1.5]len(query) ...}
```

Η διαδικασία επαναλαμβάνεται για κάθε όρο του ερωτήματος. Επειδή τα δύο μοντέλα έχουν διαφορετικό σύστημα στάθμισης, ο υπολογισμός των βαρών βρίσκεται μέσα σε συνθήκες, ανάλογα με το `model_type` που χρησιμοποιούμε.

Το αποτέλεσμα είναι να έχουμε δημιουργήσει το διάνυσμα ερωτήματος και 1209 διανύσματα εγγράφων. Στην λίστα `similarity` υπολογίζουμε την ομοιότητα συνημιτόνου μεταξύ του διανύσματος ερωτήματος και κάθε διανύσματος εγγράφου. Επειδή κάποια διανύσματα αποτελούνται από μηδενικά, στους υπολογισμούς οδηγούμαστε σε `nan` τιμές, που μετατρέπουμε σε `0`. Η συνάρτηση εν τέλει επιστρέφει τα 100 κείμενα με την μεγαλύτερη ομοιότητα.

2.4 ΥΛΟΠΟΙΗΣΗ colBERT

Στο αρχείο `colBERT_preprocessing.py` γίνεται η προεπεξεργασία των εγγράφων και ερωτημάτων ώστε να δοθούν ως inputs στο colBERT μοντέλο. Συγκεκριμένα γίνεται μετατροπή των ερωτημάτων σε κεφαλαία και αποθήκευση σε JSON αρχεία τα έγγραφα και ερωτήματα σε λεξικά ως εξής:

```
colBERTdocs = {"docID_i": "<doc_i>", ... }
colBERTqueries = {"queryID_i": "<query_i>", ... }
```

Η εκτέλεση του colBERT μοντέλου πραγματοποιείται μέσω του αρχείου `colBERT.ipynb` στο Google Colab. Τα λεξικά των JSON αρχείων αφού διαβαστούν, διασπώνται σε 2 ξεχωριστές λίστες: `doc_list.keys()` για τα IDs, `doc_list.values()` για το περιεχόμενο και αντίστοιχα για τα ερωτήματα.

Μετά τη δημιουργία του Indexer και του Searcher με `doc_maxlen = 300`, `kmeans_niters = 20` και `k = 100`, δημιουργείται μια λίστα τα 100 πιο σχετικά κείμενα, ο οποία γίνεται download ως JSON αρχείο.

3 ΜΕΤΡΙΚΕΣ ΑΞΙΟΛΟΓΗΣΗΣ

Μιας και ως δεδομένα έχουμε τα πραγματικά σχετικά έγγραφα (`Relevant_20`), για την σύγκριση των μοντέλων χρησιμοποιήθηκαν:

- **Διάγραμμα ανάκλησης-ακρίβειας** (Precision-Recall curve). Το διάγραμμα παρουσιάζει ενδιαφέρον γιατί –συνδυάζοντας τις μετρικές ανάκλησης και ακρίβειας– παρουσιάζει με προφανή τρόπο τα σημεία που το μοντέλο παρουσιάζει μεγαλύτερη ακρίβεια και ταυτόχρονα λιγότερη ανάκληση και αντίστροφα.

$$\text{Ακρίβεια} = \frac{\text{αριθμός σχετικών ανακτηθέντων κειμένων}}{\text{αριθμός εγγράφων που ανακτήθηκαν}} \qquad \text{Ανάκληση} = \frac{\text{αριθμός σχετικών ανακτηθέντων κειμένων}}{\text{αριθμός σχετικών εγγράφων στη συλλογή}}$$

- **Μέσος Όρος Μέσης Ακρίβειας** (Mean Average Precision – MAP). Χρησιμοποιούμε την συγκεκριμένη μετρική για τη δημιουργία μιας συνοπτικές ενιαίας τιμής της κατάταξης, χρησιμοποιώντας τα πολλαπλά ερωτήματα που έχουμε.

$$\text{Μέση Ακρίβεια} = \sum_{i=1}^{i=k} (\text{ανάκληση}[i] - \text{ανάκληση}[i-1]) \times \text{ακρίβεια}[i]$$

$$\text{MAP} = \frac{\text{Μέση Ακρίβεια για κάθε ερώτημα}}{\text{Σύνολο ερωτημάτων}}$$

Η συνάρτηση `recall_precision()` με εισόδους τα αποτελέσματα των μοντέλων και τα πραγματικά σχετικά έγγραφα, υπολογίζει τις τιμές ανάκλησης και ακρίβειας για κάθε ερώτημα και τις επιστρέφει σε μία λίστα. Η λίστα αυτή εισάγεται στην `mean_average_precision()` όπου υπολογίζεται ο Μέσος Όρος Μέσης Ακρίβειας.

3.1 ΑΠΟΤΕΛΕΣΜΑΤΑ

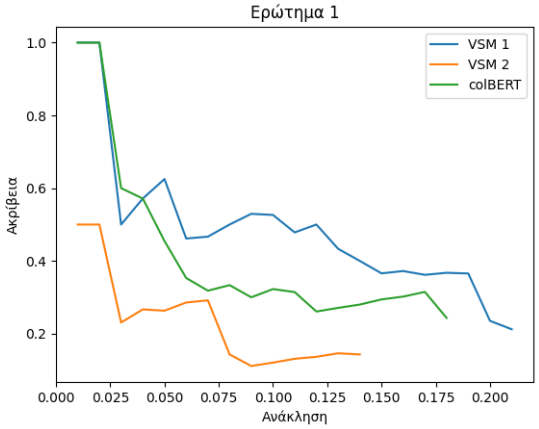
Οι τιμές του Μέσου Όρου Μέσης Ακρίβειας (MAP) για τα τρία μοντέλα είναι:

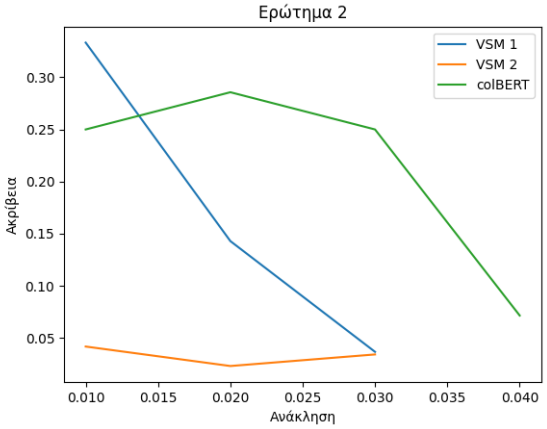
VSM 1	`0.016352`
VSM 2	`0.008598`
colBERT	`0.007089`

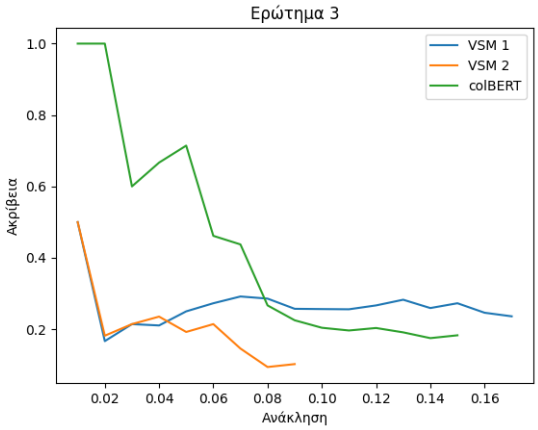
Στα διαγράμματα ανάκλησης-ακρίβειας που παρουσιάζονται παρακάτω, παρατηρούμε πως σχεδόν πάντα το **VSM #2** μοντέλο τερματίζει τελευταίο. Αυτό είναι κάτι που συνάδει με τα αποτελέσματα των Salton-Buckley, αλλά είναι και λογικό αν λάβουμε υπόψη τον τρόπο υπολογισμού των βαρών του, στα οποία δεν εμπεριέχεται ο παράγοντας IDF στα βάρη των εγγράφων.

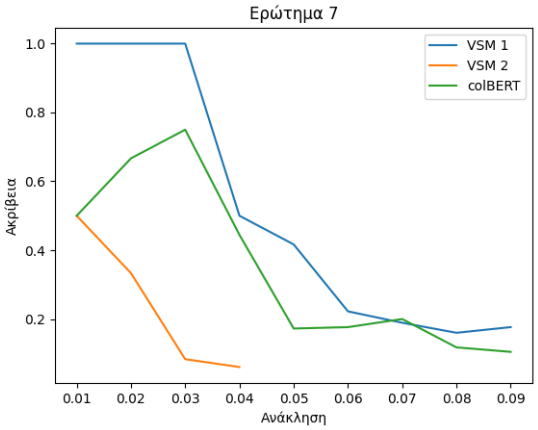
Από την άλλη, ξεκάθαρη υπεροχή παρουσιάζει το **VSM #1** μοντέλο και σε ακρίβεια αλλά και σε ανάκληση, κάτι που είναι εξίσου σημαντικό όταν έχουμε να κάνουμε με επιστημονικές/ιατρικές αναζητήσεις. Η υπεροχή του **VSM #1** αποτυπώνεται και από τη MAP τιμή του που είναι διπλάσια από τα υπόλοιπα μοντέλα, δείχνοντας μια σταθερότητα εν' όψει όλων των ερωτημάτων στο οποίο εξετάστηκε.

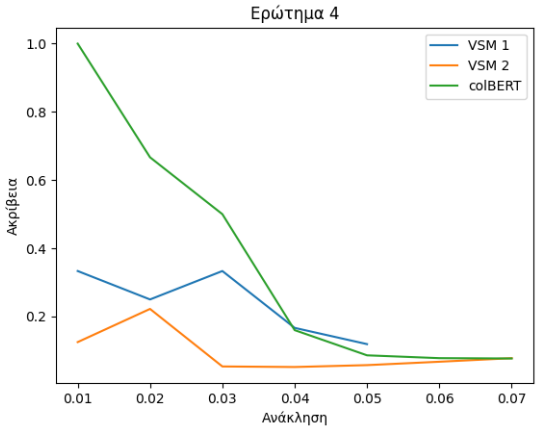
Θεωρώ πως το **colBERT** θα έπρεπε να τα είχε πάει καλύτερα, δεδομένης της σημασιολογικής του ικανότητας, αν και σε μεμονωμένες περιπτώσεις φαίνεται να υπερτερεί, ειδικά ως προς την ακρίβεια. Ίσως η συλλογή των κειμένων που χρησιμοποιήθηκε να μην ήταν η ιδανική για το συγκεκριμένο μοντέλο, ή ίσως το μοντέλο να επιδέχεται διαφορετικές παραμετροποιήσεις που θα μπορούσαν να βελτιώσουν την απόδοσή του.

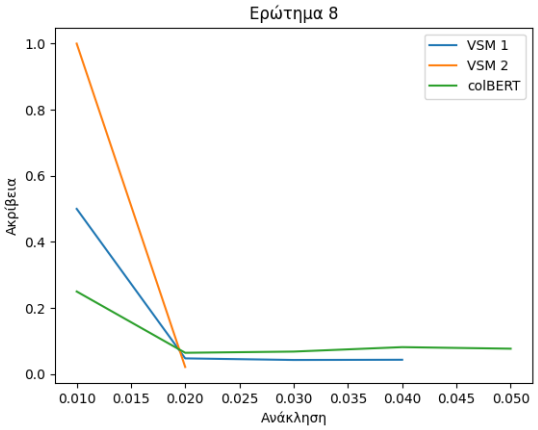


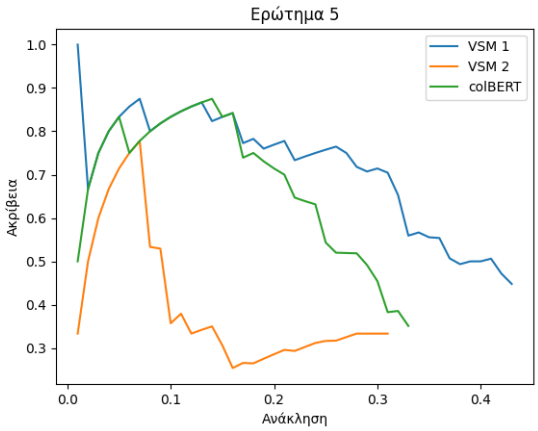


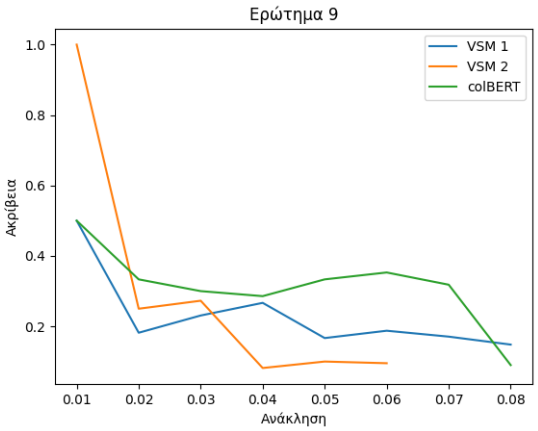

Ερώτημα 3
VSM 1
VSM 2
colBERT
Ακρίβεια
Ανάκληση

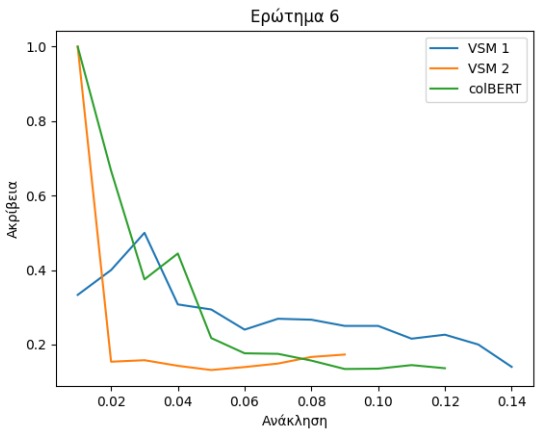

Ερώτημα 7
VSM 1
VSM 2
colBERT
Ακρίβεια
Ανάκληση

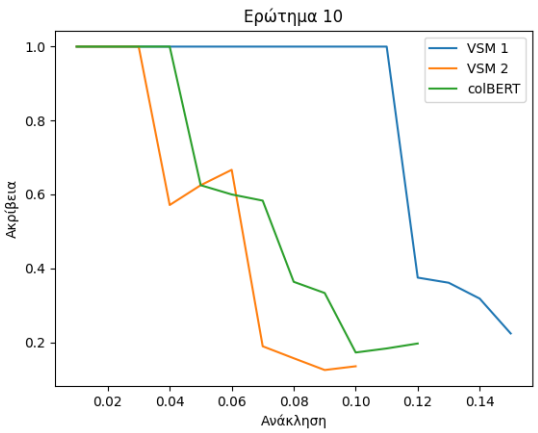

Ερώτημα 4
VSM 1
VSM 2
colBERT
Ακρίβεια
Ανάκληση


Ερώτημα 8
VSM 1
VSM 2
colBERT
Ακρίβεια
Ανάκληση

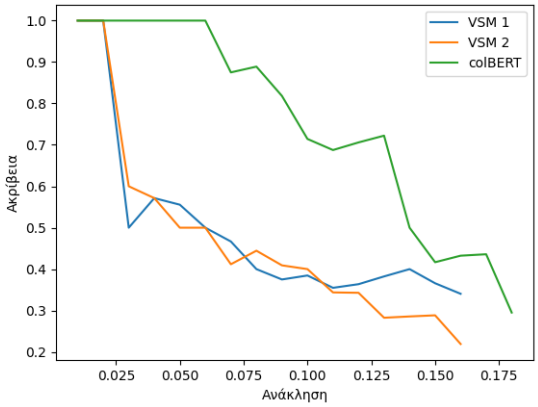

Ερώτημα 5
VSM 1
VSM 2
colBERT
Ακρίβεια
Ανάκληση


Ερώτημα 9
VSM 1
VSM 2
colBERT
Ακρίβεια
Ανάκληση

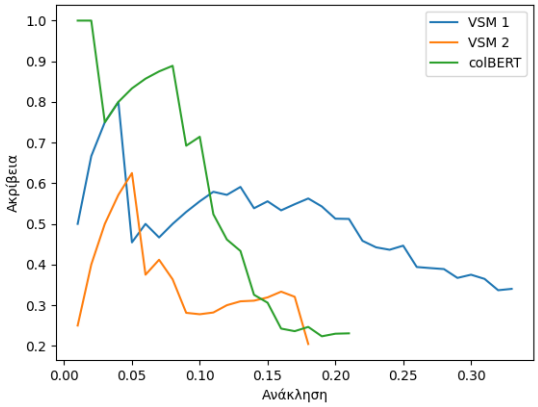

Ερώτημα 6
VSM 1
VSM 2
colBERT
Ακρίβεια
Ανάκληση


Ερώτημα 10
VSM 1
VSM 2
colBERT
Ακρίβεια
Ανάκληση

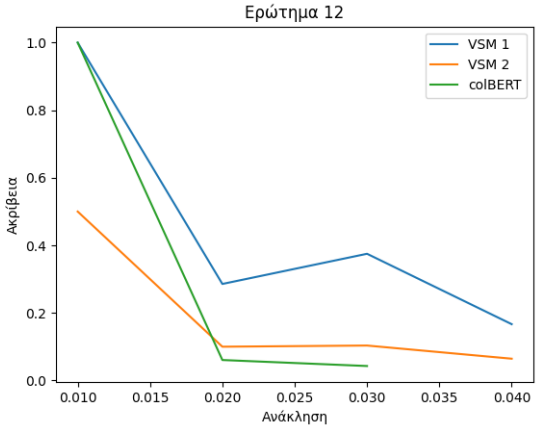
Ερώτημα 11



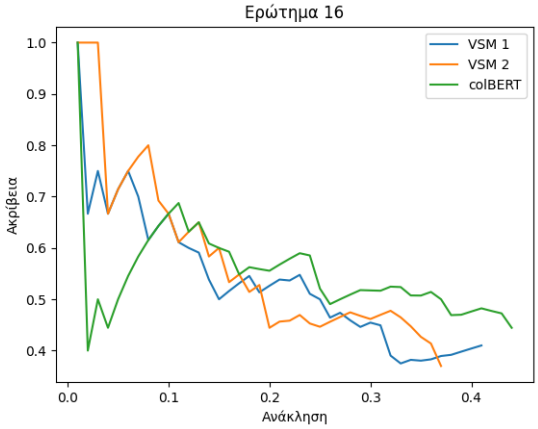
Ερώτημα 15



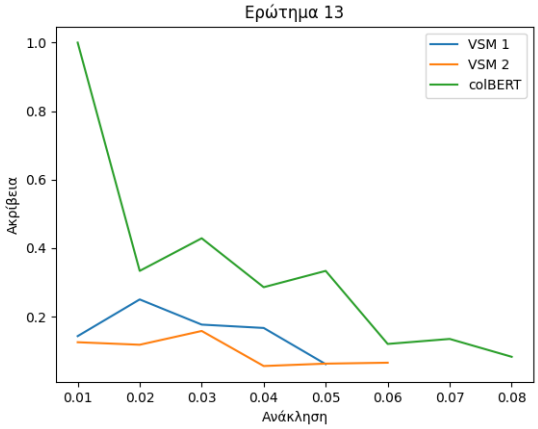
Ερώτημα 12



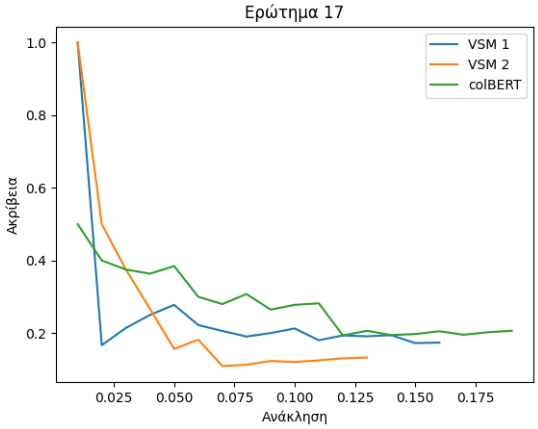
Ερώτημα 16



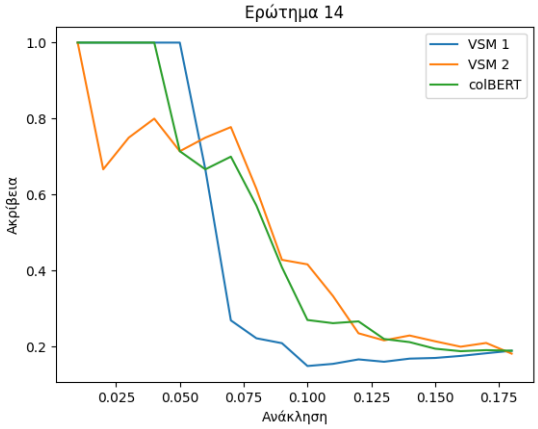
Ερώτημα 13



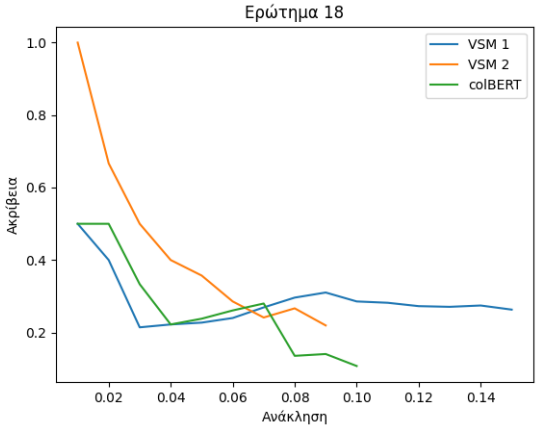
Ερώτημα 17

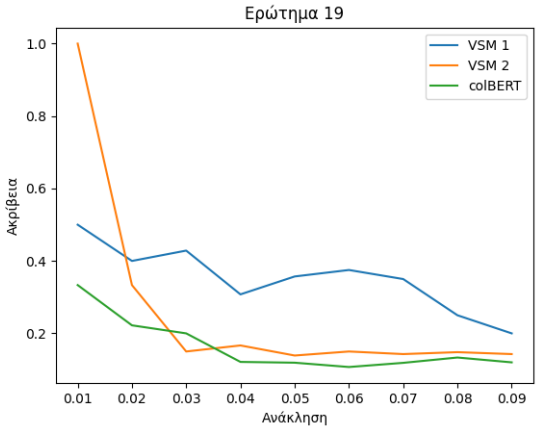


Ερώτημα 14



Ερώτημα 18





4 ΠΑΡΑΡΤΗΜΑ

4.0.1 tools.py

```python
import json
import os

def get_docs():
    docs_directory = 'Collection/docs'
    filename_list = [file for file in os.listdir(docs_directory)]

    doc_tuples_list = []

    for filename in filename_list:
        with open(os.path.join(docs_directory, filename), 'r') as doc_file:
            doc = doc_file.readlines()
            doc = [word.strip() for word in doc]
            doc_tuple = (filename, doc)  # (DocID, <doc>)
            doc_tuples_list.append(doc_tuple)

    return doc_tuples_list

def get_queries():
    filename = 'Collection/Queries_20'

    with open(filename, 'r') as queries_file:
        queries = queries_file.readlines()

    return queries


def get_json_file(json_file):
    with open(json_file, 'r') as file:
        json_data = json.load(file)

    return json_data


def get_relevant():
    relevant_docs = {}
    with open('Collection/Relevant_20', 'r') as file:
        for i, line in enumerate(file):
            relevant_docs[i] = [int(item) for item in line.split()]

    return relevant_docs
```

4.0.2 preprocessing.py

```
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import tools

def preprocess_collection():
    doc_tuples_list = tools.get_docs()

    stop_words = set(stopwords.words('english'))
    stemmer = PorterStemmer()

    stripped_docs = []

    for doc_tuple in doc_tuples_list:
        stripped_doc = [stemmer.stem(word.lower()) for word in doc_tuple[1] if word.lower() not in stop_words]
        stripped_doc_tuple = (doc_tuple[0], stripped_doc)  # (DocID, <stripped_doc>)
        stripped_docs.append(stripped_doc_tuple)

    return stripped_docs

def preprocess_queries():
    queries = tools.get_queries()

    stop_words = set(stopwords.words('english'))
    stemmer = PorterStemmer()

    stripped_queries = []

    for query in queries:
        stripped_query = [stemmer.stem(word.lower()) for word in query.split() if word.lower() not in stop_words]
        stripped_queries.append(stripped_query)

    stripped_queries_new = []
    for query in stripped_queries:
        query = list(set(query))
        stripped_queries_new.append(query)

    return stripped_queries_new
```

4.0.3 inverted_index.py

```
def create_inverted_index(stripped_docs_tuples):
    inverted_index = {}

    for doc_tuple in stripped_docs_tuples:
        for term in doc_tuple[1]:
            if term not in inverted_index:
                inverted_index[term] = set()
            inverted_index[term].add((doc_tuple[0], doc_tuple[1].count(term)))

    return inverted_index
```

4.0.4 vector_space_model.py

```
import math
```

```
import numpy as np
from numpy.linalg import norm

def idf1(N, n):
    return math.log(N / n)

def idf2(N, n):
    return math.log(N - n / n)

def get_value(item):
    return item[1]


def vsm(doc_collection, query, inverted_index, model_type):
    query_tfidfs = {}
    doc_tfidfs = {}
    for doc in doc_collection:
        doc_tfidfs[doc[0]] = []  # Αρχικοποίηση λεξικού tf-idf τιμών με άδειες λίστες για κάθε έγγραφο

    # Υπολογισμός TF όρων για το ερώτημα:
    query_tfs = {}
    for term in query:
        query_tfs[term] = query.count(term)

    # Υπολογισμός TF-IDF τιμών. Διατρέχουμε κάθε όρο από το ερώτημα...
    for term in query:
        docs_containing_term = set()

        # Αν ο όρος του ερωτήματος υπάρχει στο ανεστραμμένο ευρετήριο, υπολογίζουμε την IDF τιμή του.
        if term in inverted_index:
            if model_type == "1":
                idf = idf1(len(doc_collection), len(inverted_index[term]))
            elif model_type == "2":
                idf = idf2(len(doc_collection), len(inverted_index[term]))

            # και την TF-IDF τιμή, δηλαδή το ΒΑΡΟΣ ΟΡΟΥ ΕΡΩΤΗΜΑΤΟΣ
            if model_type == "1":
                query_tfidfs[term] = (0.5 + 0.5 * (query_tfs[term] / max(query_tfs.values()))) * idf
            elif model_type == "2":
                query_tfidfs[term] = idf

            # Υπολογισμός TF-IDF τιμών για τα έγγραφα:
            for item in inverted_index[term]:
                docs_containing_term.add(item[0])
                # το docs_containing_term περιέχει τα έγγραφα που περιέχουν τον συγκεκριμένο όρο.

            # διατρέχουμε τα έγγραφα του docs_containing_term...
            for doc in doc_collection:
                if doc[0] in docs_containing_term:
                    # έγγραφο βρέθηκε - υπολογίζουμε TF
                    for item in inverted_index[term]:
                        if item[0] == doc[0]:
                            doc_tf = item[1]
                            break
                    # υπολογισμός TFIDF
                    if model_type == "1":
                        doc_tfidfs[doc[0]].append(doc_tf * idf1(len(doc_collection), len(inverted_index[term])))
                    elif model_type == "2":
```

```
                        doc_tfidfs[doc[0]].append(0.5 + 0.5 * (query_tfs[term] / max(query_tfs.
    values())))
                else:
                    # δε βρέθηκε έγγραφο, 0 στο διάνυσμα
                    doc_tfidfs[doc[0]].append(0)

    similarity = {}
    for doc in doc_tfidfs:
        similarity[doc] = np.dot(list(query_tfidfs.values()), doc_tfidfs[doc]) / (
                    norm(list(query_tfidfs.values())) * norm(doc_tfidfs[doc]))

    # μετατροπή nan τιμών σε 0
    similarity = {k: 0 if np.isnan(v) else v for k, v in similarity.items()}
    sort_similarity = sorted(similarity.items(), key=get_value)

    return sort_similarity[-100:][::-1]


def run_vsm(doc_collection, queries, inverted_index):
    results1 = []
    results2 = []

    for query in queries:
        results1.append(vsm(doc_collection, query, inverted_index, "1"))
        results2.append(vsm(doc_collection, query, inverted_index, "2"))

    return results1, results2
```

4.0.5 colBERT_preprocessing.py

```
import json
import tools

docs = tools.get_docs()
queries = tools.get_queries()

ColBERTqueries = {}
for i, query in enumerate(queries):
    ColBERTqueries[str(i)] = query.strip().upper()

colBERTdocs = {}
for i, doc in enumerate(docs):
    colBERTdocs[doc[0]] = doc[1]

with open("colBERT_input/colBERT_docs", "w") as filename:
    json.dump(colBERTdocs, filename)
    filename.close()

with open("colBERT_input/colBERT_queries", "w") as filename:
    json.dump(ColBERTqueries, filename)
    filename.close()
```

4.0.6 colBERT_.ipynb

Εισαγωγή Βιβλιοθηκών/ColBERT

```
!git -C ColBERT/ pull || git clone https://github.com/stanford-futuredata/ColBERT.git
import sys; sys.path.insert(0, 'ColBERT/')
try: # When on google Colab, let's install all dependencies with pip.
```

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

Εισαγωγή αρχείων

```
import json

with open('colBERT_docs', "r") as file:
  doc_list = json.load(file)

with open('colBERT_queries', "r") as file:
  query_list = json.load(file)

doc_ids = list(doc_list.keys())
doc_content = list(doc_list.values())

query_ids = list(query_list.keys())
query_content = list(query_list.values())

nbits = 2
doc_maxlen = 300
```

Δημιουργία Indexer - Searcher

```
checkpoint = 'colbert-ir/colbertv2.0'

with Run().context(RunConfig(nranks=1, experiment='notebook')):  # nranks specifies the number of GPUs to use
    config = ColBERTConfig(doc_maxlen=doc_maxlen, nbits=nbits, kmeans_niters=20) # kmeans_niters specifies the number of iterations of k-means clustering; 4 is a good and fast default.
                                                                                 # Consider larger numbers for small datasets.

    indexer = Indexer(checkpoint=checkpoint, config=config)
    indexer.index(name='index_name', collection=doc_content, overwrite=True)

with Run().context(RunConfig(experiment='notebook')):
    searcher = Searcher(index='index_name', collection=doc_content)
```

Αποτελέσματα:

```
results = []
for query in query_content:
  result = searcher.search(query, k=100)

  passages = []
  for passage_id, passage_rank, passage_score in zip(*result):
      passages.append(int(doc_ids[passage_id]))
  results.append(passages)
```

```
print(results)

from google.colab import files
import json

with open("colBERT_output.json", "w") as file:
  json.dump(results, file)

files.download("colBERT_output.json")
```

4.0.7 evaluation_metrics.py

```
import tools
import matplotlib.pyplot as plt
import numpy as np

def recall_precision(results, model):
    # results: λίστα όλων( των queries) που περιλαμβάνει λίστες με τα πιο σχετικά έγγραφα, όπως υπολογίστηκαν από το cosine similarity
    relevant_docs = tools.get_relevant()
    # relevant_docs: dictionary με τις τιμές των πραγματικά σχετικών εγγράφων του Relevent_20

    recall_precision_values = []

    if model == "vsm":
        # Το results περιλαμβάνει συγκεντρωτικά όλα τα αποτελέσματα για κάθε query. Διατρέχουμε το κάθε query μεμονομένα...
        for i, results_query in enumerate(results):
            precision = []
            recall = []
            truly_relevant_docs = 0
            # διατρέχουμε τα πιο σχετικά έγγραφα για το συγκεκριμένο query...
            for j, doc in enumerate(results[i]):
                if type(doc) == tuple:
                    current_doc = int(doc[0])
                    if current_doc in relevant_docs[i]:
                        truly_relevant_docs += 1

                        recall.append(truly_relevant_docs / len(results_query))
                        precision.append(truly_relevant_docs / (j + 1))

            recall_precision_values.append((recall, precision))

    if model == "colBERT":
        for i, results_query in enumerate(results):
            precision = []
            recall = []
            truly_relevant_docs = 0
            for j, doc in enumerate(results[i]):
                if doc in relevant_docs[i]:
                    truly_relevant_docs += 1

                    recall.append(truly_relevant_docs / len(results_query))
                    precision.append(truly_relevant_docs / (j + 1))

            recall_precision_values.append((recall, precision))

    return recall_precision_values

def mean_average_precision(recall_precision_values):
```

```
    average_precision_list = []
    for i in range(len(recall_precision_values)):
        recalls, precisions = recall_precision_values[i]
        average_precision = 0

    for j in range(1, len(recalls)):
        average_precision += (recalls[j] - recalls[j - 1]) * precisions[j]
        average_precision_list.append(average_precision)

    return np.mean(average_precision_list)


def run_metrics(vsm1_results, vsm2_results):
    colBERT_results = tools.get_json_file("colBERT_output.json")

    recall_precision_vsm1 = recall_precision(vsm1_results, "vsm")
    recall_precision_vsm2 = recall_precision(vsm2_results, "vsm")
    recall_precision_colBERT = recall_precision(colBERT_results, "colBERT")

    map_vm1 = mean_average_precision(recall_precision_vsm1)
    map_vm2 = mean_average_precision(recall_precision_vsm2)
    map_colBERT = mean_average_precision(recall_precision_colBERT)

    print("vm1", map_vm1)
    print("vm2", map_vm2)
    print("colBERT", map_colBERT)

    for i in range(len(recall_precision_vsm1)):
        plt.figure()
        plt.plot(recall_precision_vsm1[i][0], recall_precision_vsm1[i][1], label="VSM 1")
        plt.plot(recall_precision_vsm2[i][0], recall_precision_vsm2[i][1], label="VSM 2")
        plt.plot(recall_precision_colBERT[i][0], recall_precision_colBERT[i][1], label="colBERT")
        plt.xlabel('Ανάκληση')
        plt.ylabel('Ακρίβεια')
        plt.title(f'Ερώτημα {i + 1}')
        plt.legend(loc='upper right')
        plt.savefig('Precision_Recall_Curve/' + str(i + 1) + '.png')
```

4.0.8 main.py

```
import inverted_index as ii
import vector_space_model as vsm
import preprocessing as prep
import evaluation_metrics as metrics

def main_app():
    doc_collection = prep.preprocess_collection()   # stripped_docs
    queries = prep.preprocess_queries()     # stripped_queries

    inverted_index = ii.create_inverted_index(doc_collection)
    results1, results2 = vsm.run_vsm(doc_collection, queries, inverted_index)

    metrics.run_metrics(results1, results2)


if __name__ == '__main__':
    main_app()
```