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 δύο συστήματα στάθμισης των βαρών για τους όρους των εγγράφων και των ερωτημάτων.

1.1.1 ΠΡΩΤΟ ΣΥΣΤΗΜΑ ΣΤΑΘΜΙΣΗΣ

Το πρώτο σύστημα στάθμισης 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 στο σύνολο των συλλογών στα οποία έχουν εξεταστεί αλλά και ειδικότερα για 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
nltk	αφαίρεση stopwords, stemming
numpy	υπολογισμός ομοιότητας συνημιτόνου
math	υπολογισμός λογαρίθμων
matplotlib	γραφικές παραστάσεις
json	αποθήκευση-ανάγνωση JSON αρχείων
pickle	αποθήκευση αντικειμένων Python

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
```

Η υλοποίηση του colBERT μοντέλου έχει πραγματοποιηθεί στο Google Colab ως Juputer Notebook.

2.1 ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ ΕΓΓΡΑΦΩΝ & ΒΟΗΘΗΤΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Το αρχείο `tools.py` περιλαμβάνει βοηθητικές συναρτήσεις για κάποιες επαναλαμβανόμενες διαδικασίες της υλοποίησης. Περιλαμβάνονται οι συναρτήσεις **`get_docs`**`()` και **`get_queries`**`()`.

Η συνάρτηση **`get_docs`**`()`, χρησιμοποιώντας την `os` βιβλιοθήκη διαβάζει το πλήθος των αρχείων της βιβλιοθήκης.[3] Η συνάρτηση, αφού αφαιρέσει το escape character `'\n'`, ο οποίος προκύπτει από την μορφολογία των εγγράφων (η κάθε λέξη είναι τοποθετημένη σε διαφορετική γραμμή), δημιουργεί

[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.

και επιστρέφει μια λίστα από tuples, με κάθε tuple να αντιστοιχεί σε κάθε αρχείο-έγγραφο. Τα tuples έχουν την δομή:

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

όπου `docID` η αρίθμηση του κάθε εγγράφου και `doc_term_n` η κάθε λέξη-λήμμα του εγγράφου. Η συνάρτηση `strip()` είναι απαραίτητη για την αφαίρεση των `\n` χαρακτήρων που προέκυψαν από την μορφολογία των εγγράφων (κάθε λέξη είναι σε νέα γραμμή). Με παρόμοιο τρόπο η συνάρτηση **`get_queries`**`()` επιστρέφει τη λίστα με τα ερωτήματα της συλλογής.

Στις συναρτήσεις **`preprocess_collection`**`()` και **`preprocess_queries`**`()` πραγματοποιείται η προεπεξεργασία των εγγράφων, συγκεκριμένα η αφαίρεση των stopwords και το stemming.

Χρησιμοποιούμε τη `nltk` βιβλιοθήκη. Τα `doc_tuples` της `get_docs()` αφού περάσουν από τον `PorterStemmer` της `nltk` αποθηκεύονται σε μια λίστα, η οποία στη συνέχεια επιστρέφεται. Αντίστοιχη διαδικασία πραγματοποιείται και για την προεπεξεργασία των ερωτημάτων, στην `preprocess_queries()`.

2.2 ΑΝΕΣΤΡΑΜΜΕΝΟ ΕΥΡΕΤΗΡΙΟ

Το ανεστραμμένο ευρετήριο δημιουργείται στη συνάρτηση **`create_inverted_index`**`()` του αρχείου `inverted_index.py`. Στη συνάρτηση εισάγονται οι λίστες που δημιουργήθηκαν στις προηγούμενες συναρτήσεις. Τα tuples από τα οποία αποτελούνται, αποθηκεύονται σε ένα dictionary που θα αποτελέσει το ανεστραμμένο ευρετήριο με την εξής δομή:

```
                    inverted_index['λήμμα'] =
{ ('docID στο οποίο εμφανίζεται' = <φορές εμφάνισης>), (···), ... }
```

Κάθε value του dictionary είναι ένα set[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[5]:

```
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 **`vsm1`**`(doc_collection, query, inverted_index)` ή **`vsm2`**`()`, ανάλογα με το σύστημα στάθμισης που θέλουμε να χρησιμοποιήσουμε.

Πριν προχωρήσουμε στις συναρτήσεις `vsmX()`, από το τρόπο που έχει δομηθεί το ανεστραμμένο ευρετήριο, είναι σαφές πως μπορούμε να υπολογίσουμε αμέσως το πλήθος των εγγράφων όπου υπάρχει ένα συγκεκρίμενο λήμμα ως `len(inverted_index[term])`, με άλλα λόγια δηλαδή το n_i. Είναι επίσης προφανές ότι $N =$ `len(inverted_index)`.

Στην συνάρτηση `vsm1()` αρχικά υπολογίζονται στο λεξικό `query_tfs` οι 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.

[5] Τα λήμματα έχουν τη stemming μορφή τους.

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

Στα λεξικά `query_tfidfs` και `doc_tfidfs` θα αποθηκευτούν οι τελικές τιμές-βάρη που θα αποτελέσουν τα διανύσματα ερωτήματος και εγγράφων. Το λεξικό `doc_tfidfs` αρχικοποιείται με άδειες λίστες για κάθε έγγραφο.

Διατρέχουμε κάθε όρο του ερωτήματος `query` και αν ο όρος υπάρχει στο ανεστραμμένο ευρετήριο υπολογίζουμε την IDF τιμή του στην συνάρτηση `idfX()`, και στη συνέχεια το TF-IDF βάρος του ερωτήματος στο `query_tfidfs`.

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

Στη συνέχεια διατρέχουμε όλα τα έγγραφα της συλλογής. Αν το έγγραφο περιλαμβάνει τον όρο του ερωτήματος που εξετάζουμε (κάτι που ελέγχουμε από τη λίστα `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) ...}
```

Εν τέλει έχουμε δημιουργήσει το διάνυσμα ερωτήματος και 1209 διανύσματα εγγράφων, τα οποία έχουν μήκος όσο και το διάνυσμα ερωτήματος. Στην λίστα `similarity` υπολογίζουμε την ομοιότητα συνημιτόνου μεταξύ του διανύσματος ερωτήματος και κάθε διανύσματος εγγράφου. Τέλος, ταξινομούμε την λίστα και επιστρέφουμε τα 100 κείμενα με την μεγαλύτερη ομοιότητα.

Η ίδια διαδικασία ακολουθείται και στην συνάρτηση `vsm2()`, με τις απαραίτητες αλλαγές για το διαφορετικό σύστημα στάθμισης.

2.4 ΥΛΟΠΟΙΗΣΗ colBERT

Στο αρχείο `colBERT_preprocessing.py` γίνεται η προεπεξεργασία των εγγράφων και ερωτημάτων ώστε να δωθούν ως inputs στο colBERT μοντέλο. Συγκεκριμένα έγινε μετατροπή των ερωτημάτων σε κεφαλαία και αποθηκεύτηκαν 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 μετατρέπονται σε 2 ξεχωριστές λίστες (id και περιεχόμενο). Μετά τη δημιουργία του Indexer και του Searcher, επιστρέφονται τα 100 κείμενα με την μεγαλύτερη ομοιότητα.

3 ΜΕΤΡΙΚΕΣ ΑΞΙΟΛΟΓΗΣΗΣ

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 ακρίβεια τα σημεία που το μοντέλο παρουσιάζει μεγαλύτερη ακρίβεια και λιγότερη ανάκληση και αντίστροφα.

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

- **???????** (Mean Average Precision – mAP). Χρησιμοποιούμε την συγκεκριμένη μετρική λόγω των πολλαπλών ερωτημάτων όπου εφαρμόζουμε τα μοντέλο σε αυτά, για την αξιολόγηση των αποτελεσμάτων.

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

3.1 ΑΠΟΤΕΛΕΣΜΑΤΑ