ניהול נתונים באינטרנט 2022, פרויקט תכנות information retrieval

ליאן גלנטי 211469614, יובל דביר 204256846

במסמך זה יתואר מנוע החיפוש בשני שלבים. ראשית, על בניית inverted index ספציפי על סמך המסמכים שניתנו. שנית, על האופן שבו המערכת מחזירה מסמכים עפ"י שאילתה ספציפית.

<u>תיאור כללי במילים:</u>

שלב א': בניית האינדקס

ראשית כדי שיהיה יותר ברור אציג את מבני הנתונים הראשיים שאיתם בונים את האינדקס:

document_similarities – a dictionary
key = record_num, value = {"REFERENCES": set(), "CITATIONS": set(), "MAJORSUBJ": set(), "MINORSUBJ": set()}} – each key holds a set of values for the element (key) in the paper for the specific record_num.

DocumentReference – a dictionary
key = record_num, value =

[0, 0, {"REFERENCES": [], "CITATIONS": [], "MAJORSUBJ": [], "MINORSUBJ": []}, 0, ""]=

[vector length, paper's length, {“REFERNCES”: [], “CITATIONS”: [], “MAJORSUBJ”: [], “MINORSUBJ”:[]}, paper's maximum frequency of word, publication name]

inverted_index – a dictionary
key = word, value =

{"df": #docs with word, "records": {record_num1: tf_word_record_num1_iD, ..., record_numk: tf_word_record_numk_iD}}

א. התכנית מקבלת כקלט נתיב לדאטא.
ב. לכל מאמר בדאטא נעבור על אלמנטי הxml הבאים:
   1. Recordnum עבור מזהה המאמר.
   2. Title כותרת.

3. Extract וכן abstract שכאמור חשוב מאוד משום שהוא תמצית המאמר.
4. Majorsubj, minorsubj, שהם הנושאים המרכזיים והמשניים בהתאמה.
5. References, citations, שמתארים ציטוטים שהמאמר עושה ונעשים בו בהתאמה.
6. Source, עבור שמירת מקום פרסום.

ג. נשמור את ערך אלמנט 1, כלומר record_num. לכל המילים שראינו נבצע tokenization, filter stopwords, stemming to the rest. ואז על כל מילה שהגיעה מאלמנטים 2-3, נוסיף 1 לtf של המילה במסמך record_num באינדקס. באותו אופן לכל מילה בערכי האלמנטים מ4, נוסיף adder>=1 (באופן אינטואיטיבי אלו מילים חשובות יחסית יותר ולכן נרצה להגדיל את המשקל שלהן יותר).

ד. במידה וrecord_num עדיין לא נמצא באינדקס עבור המילה הנוכחית, נוסיף 1 לdf של המילה ונעדכן את הtf מ0 לadder המתאים.

ה. עבור אלמנט 6 נשמור את מקום הפרסום שבערך, בDocumentReference.

ו. נוסף על כך נשמור מילון document_similarities שהמפתחות בו הם record_num והערכים בהם הם מילון שהמפתחות בו הוא האלמנטים 4-5 שערכיהם הם קבוצות של הערכים שאנו רואים באלמנטים 4 וזוגות של מקום פרסום וכותב מאלמנטים 5 בהתאמה.

ז. לאחר מכן נעבור על כל זוגות המאמרים. לכל זוג, נסתכל על האלמנטים 4,5 שמופיעים לכל בן-זוג מתוך document_similarities ונקח את הקבוצות שנמצאות בערכים. נחשב קרבת הקבוצות לפי הכללה של Jaccard similarity, Tversky index. במידה והקרבה עוברת סף או שהיא קרובה יחסית לסף וגם מקום הפרסום זהה נשמור לכל בן-זוג בDocumentReference את בן הזוג השני תחת האלמנט המתאים מתוך 4,5 (כך אם אחד מהם יוחזר בחלק ב' נחזיר גם את השני שכן הם דומים).

ח. נעדכן לכל מסמך את האורך שלו ב בDocumentReference ע"י חישוב ציוני tf-idf.

ט. נשמור את את inverted_index, DocumentReference כקורפוס וסיימנו.

שלב ב':

א. נטען מהזכרון את האינדקס ונקבל שאילתה, סוג ציון.

ב. ניצור אינדקס query_index עבור השאילתה.

ג. נעבור על השאילתה, נבצע tokenization, filter stopwords, stemming to the rest ולכל מילה שנשארה נוסיף במקום שלה באינדקס 1 על ההופעה שלה.

ד. נשמור את האורך של השאילתה המעודכנת.

ה. נשתמש באחת מפונקציות הדירוג: tfidf, bm25, mix (of bm25, tfidf), bm25+, piv+.

ו. נקצר את רשימת המסמכים שהוחזרו ע"י תנאי.

ז. לכל מסמך מאלו שנבחרו, נסתכל על הקבוצות של המסמכים שדומים לו בכל האלמנטים 4,5. נסתכל על חיתוך כל זוג מהקבוצות ונחזיר תת קבוצה שלהם.

<u>תיאור הפונקציות:</u>

1. הפונקציה שעוברת על המאמרים: עוברים עליהם בעזרת המזהה הייחודי. מעדכנים עבור כל מאמר את כל הערכים הרלוונטיים לפי האלמנטים שצוינו מעלה במבני הנתונים שצוינו מעלה.

```
def get_primary_data(file, DocumentReference, inverted_index,
document_similarities):
```

2. הפונקציה שמעדכנת בDocumentReference לכל מאמר את המסמכים שהכי דומים לו לפי כל סוג אלמנט מתוך 4,5 המצויינים לעיל.

```
def update_similarities(DocumentReference,
document_similarities, alpha=0.8, beta=0.8):
```

3. הפונקציה שמעדכנת את אורך הוקטור לכל מאמר בDocumentReference.

```
def update_vector_length(N, DocumentReference,
inverted_index):
```

למעשה מימוש זהה לקוד מהשיעור הנראה מטה, אך לtf מבוצע נרמול לפי אורך המסמך.

Computing Document Lengths

Assume the length of all document vectors (in DocumentReference) are initialized to 0.0;

```
For each token T in H:
    Let I be the IDF weight of T;
    For each TokenOccurence of T in document D
        Let C be the count of T in D;
        Increment the length of D by (I * C)^2;
For each document D in H:
    Set the length of D to be the square-root of the
        current stored length;
```

4. הפונקציה שנקראית כאשר רוצים לייצר את האינדקס, היא עוברת על הדאטא, קוראת לפונקציות הקדומות ולבסוף שומר את הקורפוס לזכרון.

```
def create_index(input_dir):
```

5. הפונקציה שמעדכנת את השאילתה ואת האינדקס שלה.

```
def update_query(question, query_index):
```

6. הפונקציות שמממשות את אלגוריתם ההחזרה מהכיתה לפי ציוני tf-idf. הן מחזירות את המסמכים הרלוונטיים והציון המתאים.

```
def RA(N, query_details, DocumentReference, inverted_index):
def RAc(N, query_details, inverted_index, DocumentReference):
```

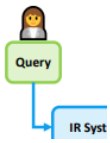
כלומר:

Retrieval Algorithm using Inverted-Index

```
Create HashMapVector, Q // for the query
Create empty HashMap, R // store retrieved documents with scores
For each token, T, in Q:
    Let I = IDF(T), K = tf(T, Q); // K is the count of T in Q
    Set W = K * I; // weight of token T in query Q
    Let L be the list of TokenOccurences of T from H; // H is the inverted index
    For each TokenOccurence, O, in L:
        Let D = document(O) and C = tf(T, D);
        If D ∉ R: // D was not previously retrieved
            Set R[D] = 0.0;
        Set R[D] += W * I * C; // product of T-weight in Q and D
```

Retrieval Algorithm cont.

```
Compute L = length(Q);  // square-root of the sum of the squares of its weights
For each retrieved document D ∈ R:
    Let S = accumulated score of D; // S is the dot-product of D and Q
    Let Y = length(D); // as stored in its DocumentReference
    Normalize D's final score to S/(L·Y);
Sort retrieved documents in R by final score
Return results in sorted array
```



לשים לב שquery_details זה מערך שבמקום ה0 יש ערך שהוא אורך השאילתה לאחר העדכון ובמקום 1 יש את האינדקס של השאילתה.

7. הפונקציה שמקבלת מדד מ+bm25, bm25, piv, ואת השאילתה ומחזירה רשימה של המסמכים הרלוונטיים עם הציון. המימוש: מסתכלים כל כל המסמכים שיש בהם לפחות מילה אחת מהשאילתה המעודכנת ואז לכל מסמך מחשבים את הscoring לפי ההגדרה.

קראתי על המדדים הנוספים בין השאר, במאמר Lower-Bounding Term Frequency Normalization by Yuanhua Lv, ChengXiang Zhai. (התמקדתי בשיפור tfidf, bm25 לשם הפרויקט. אני חושבת שניתן לשפר את המדדים הנוספים ואף לעקוף משמעותית את הציונים של המדדים שעליהם אנחנו נבחנים. למרות זאת, רציתי לממש כדי להתנסות עוד.)

```
def Func_score(N, query_details, inverted_index,
DocumentReference, Func="bm25", k1=3, b=0.75):
```

8. הפונקציה שקוראת לפונקציה מ6 או 7 לפי המדד המתאים.

```
def ranking_by(N, query_details, inverted_index,
DocumentReference, ranking="tfidf"):
```

9. פונקציה שמקבלת רשימה של סוגי מדדים ומחזירה את רשימת המסמכים שהם כולם מסכימים עליהם כאשר הציון החדש לכל מסמך הוא סכום של הציונים הקודמים מנורמלים.

```
def mixed_scoring(N, query_details, inverted_index,
DocumentReference, scoring_funcs):
```

10. הפונקציה שמוסיפה מסמכים למסמכים שנבחרו ע"י ranking_by.

```
def add_similar_docs(DocumentReference, chosen_docs, min_val,
ranking="bm25"):
```

11. הפונקציה שנקראת כשמריצים משורת הפקודה query, היא מחשבת מסמכים מתאימים לשאלה ושומרת אותם לזכרון. קוראת לranking_by, בהתאם לסוג הדירוג, מקצרת את הרשימה לפי אינדיקטור וקוראת לadd_similar_docs.

```
def ev_query(ranking, index_path, question):
```

12. הפונקציה שקוראת לcreate_index, ev_query.

```
def main():
```