מנוע חיפוש

דו"ח חלק ב'

אחזור מידע

מגישות:

רותם מיארה 301886776

יסמין אברהם 208063453

1. הסבר מפורט על אופן פעולת המנוע:

א. ביצוע שינויים במספר מחלקות מחלק א' של הפרויקט:

במחלקה Doc:

שדות:

- String title- שתפקידו לתת את כותרת המסמך.

במחלקה Parse:

במחלקה זו בוצעו רוב השינויים על מנת שיתאימו גם לפירסור שאילתה. הוספנו בנאי עבור פירסור של שאילתה.

שדות:

```
HashMap<String,Integer> tempToken
```

מתודות:

```
void mergeTokens()
```

ממזגת בין שני מבני נתונים של טוקנים.

```
void addEntity(String entity)
```

שמה את המילים המרכיבות את הישות במבנה נתונים נפרד.

במחלקה Indexer:

שדות:

```
Map<String,ArrayList<Entity>> docEntities
```

עבור כל מזהה מסמך שמרנו את הישויות שלו.

מתודות:

```
void findTop5Entities(String path)
```

המתודה מוצאת לכל מסך את ה 5 הישות הדומיננטיות ביותר

```
void writeEntitiesPosting(HashMap<String, Integer> tokens, String DocNo)
```

רושמת למסמך entities.txt רשימה של כל הישויות ה"חשודות" שיש בכל מסמך.

מסמך ה entities.txt בנוי כך לדוגמה:

FBIS3-80|NORTH KOREA:1&NODONG SINMUN:1&UNITED STATES:1&…

מזהה DOCNO של מסמך

יישות

במות המופעים של

יישות

במות המופעים של

במחלקת Controller:

שדות:

ReadQuery **queryFile**
אובייקט מסוג ReadQuery המאפשר לקרוא שאילתות מתוך קובץ השאילתות.

String **queryFilePath**
נתיב שבו נמצא קובץ השאילתות.

ListView<Node> **listOfDocNos**
רשימת המסמכים שנציג עבור כל שאילתא.

boolean semantics
משתנה בוליאני לבחירת אופצייה עם סמנטי או בלי.

Searcher **searcher**
אובייקט מסוג Searcher המאפשר לאחזר שאילתות.

HashMap<Integer, String> **docs**
מבנה נתונים מסוג HashMap השומר אינדקס לכל מזהה מסמך.

TreeMap<Integer, PriorityQueue<DocumentQ>> **queryInfo**
מבנה נתונים מסוג TreeMap השומר לכל מספר שאילתא את רשימת המסמכים שהוחזרו לה.

int queryNumber
משתנה עבור זהה שאילתא.

מתודות:

void browseQueryFile(ActionEvent event)
המתודה בוחרת תיקייה שבתוכה קובץ השאילתות.

void semantics(ActionEvent event)
המתודה מאפשרת למשתמש לבחור האם לבחור או לא בביצוע סמנטיקה לשאילתות.

void runQueryFromUser(ActionEvent event)
המתודה מריצה שאילתא בודד שהכניס המשתמש.

void OnClickEntities(ActionEvent event)
המתודה מאפשרת למשתמש לראות את ה5 יישיות הדומננטיות לכל מסמך.

void showAllDocs(PriorityQueue<DocumentQ> docno)
המתודה רושמת את 50 המסמכים הרלוונטים לשאילתא.

void runBrowseQuery(ActionEvent event)
המתודה מריצה קובץ של שאילתות שמכניסים.

void next(ActionEvent event)
המתודה מאפשרת מעבר בין התצוגה של כל שאילתא שקראנו מקובץ השאילתות והמסמכים הרלוונטים אליה.

void prev(ActionEvent event)

המתודה מאפשרת מעבר בין התצוגה של כל שאילתא שקראנו מקובץ השאילתות והמסמכים הרלוונטים אליה.

void writeTrecEvalResults()
המתודה שומרת את התוצאות של השאילתות שקראנו מקובץ השאילתות לתוך קובץ טקסט בפורמט של התוכנה trec-eval.

void writeUserQueryTrecEvalResults(PriorityQueue<DocumentQ> docs)
המתודה שומרת את התוצאות של השאילתא שהכניס המשתשמש לתוך קובץ טקסט בפורמט של התוכנה trec-eval.

void writeTrecEvalResultsButton(ActionEvent event)
המתודה כותבת את התוצאות ששמרנו משתי המתודות שרשמנו לעיל.

ב. שינויים בקבצי posting:

קובץ הפוסטינג documentDic.txt:

הוספנו לכל מזהה מסמך ששמרנו בו את הכותרת שלו ואת אורך המסמך(שינויים בצבע אדום).

FBIS3-6|1256|43|television|FORMER YUGOSLAV REPUBLIC OF MACEDONIA: OPINION POLLS ON|738

מזהה DOCNO של מסמך

אורך המסמך

max של מילה הנפוצה ביותר

המילה הנפוצה ביותר

כותרת המסמך

מספר מילים ייחודיות במסמך

ג. מחלקות חדשות שהוספנו:

מחלקת DocumentQ:

<u>שדות:</u>

String **DocNo**

מזהה מסמך.

int size

גודל המסמך.

HashMap<String,Integer> **termsAndFreq**
מבנה אשר מחזיק את המילים במסמך ואת מספר ההופעות שלהן בו.

double rank

דירוג המסמך לשאילתה.

HashSet<String> **wordsInTitle**
מבנה אשר מחזיר את המילים הנמצאות בכותרת המסמך.

מתודות:

void setWordsInTitle(String wordsInTitle, **boolean** isStem, HashSet<String> stopWords)

מתודה אשר מקבלת את כותרת המסמך ומפרסרת אותה.

void addTermsAndFreq(String term, **int** freq)

מתודה אשר מוסיפה למבנה הנתונים את כל המילים במסמך ואת התדירות שלהן בו.

int compareTo(Object o)

פונקציית השוואה בין שני מסמכים על פי הדרוג שלהם.

מחלקת Entity:

המחלקה מייצגת ישות במסמך.

שדות:

int tf

כמות הופעות המילה במסמך.

String **term**

שם המילה.

מתודות:

int compareTo(Object o)

פונקציית השוואה בין שני ישויות על פי התדירות שלהם במסמך.

מחלקת Semantics:

המשתמש יכול לקבוע האם להשתמש באלגוריתם שמבצע סמנטיקה בעזרת חיבור לאינטרנט או לא. המטרה היא למצוא מילים נרדפות או דומות לכל המילים בשאילתה.
עבור חיבור offline השתמשנו באלגוריתם word2vec.

עבור חיבור online השתמשנו ב API עם חיבור לאינטרנט:

https://api.datamuse.com/words?ml=

שדות:

static HashMap<String, ArrayList<String>> *concept*

מבנה נתונים סטטי המחזיק לכל מילה את המילים הנרדפות או הדומות לה.

Object[] **json**

אובייקט מסוג json

ArrayList<String> **query**

כל המילים שנמצאות בשאילתה.

מתודות:

void startConnection()

מתודה עבור חיבור online . המתודה מאפשרת למערכת להתחבר לAPI המטפלת בסמנטיקה עבור כל המילים בשאילתה.

void word2Vec()
מתודה עבור חיבור offline משתמשת באלגוריתם word2vec על מנת למצוא מילים נרדפות או דומות למילים שבשאילתה.

מחלקת ReadQuery:

המחלקה אחראית לקרוא את השאילתות מקובץ.

שדות:

File **file**
קובץ השאילתות המשתמש הכניס.

Searcher **search**
אובייקט המאפשר לנו לאחזר מסמכים עברו השאילתות בקובץ.

boolean isStem
אופציה לבחירת המשתמש האם לבצע סטמינג או שלא.

String **path**
נתיב של קובץ השאילתות.

boolean isSemantic
אופציה לבחירת המשתמש האם לבצע טיפול סמנטי או שלא.

TreeMap<Integer, PriorityQueue<DocumentQ>> **queryInfo**;
מבנה נתונים מסוג TreeMap השומר כל שאילתא את 50 המסמכים שמדורגים הכי גבוה.

מתודות:

void readQuery(HashSet<String> stopWords)
המתודה מפרסרת את השאילתא לפי התגיות: <num> מספר השאילתא ו<title> השאילתא עצמה.

מחלקת Ranker:

המחלקה אחראית על דירוג מסמכים פר שאילתה. תפקידה לדרג את התשובות לשאילתות על פי נוסחת דירוג שפתחנו.

שדות:

PriorityQueue<DocumentQ> **DocsRank**
מבנה נתונים השומר לכל מסמך את הדירוג שלו לשאילתא.

HashMap<String,DocumentQ> **docsInfo**
מבנה נתונים השומר לכל מסמך את הפרטים עליו.

HashMap<String,Integer> **termFreq**
מבנה נתונים השומר לכל מילה בשאילתא את כמות ההופעות שלה.

int numOfDocs

מספר המסמכים.

double avgOfDocsSize

ממוצע אורך המסמכים.

boolean isStem

אופציה לבחירת המשתמש האם לבצע סטמינג או שלא.

HashSet<String> **stopWords**

מבנה נתונים ששומר את כל הstop words.

מתודות:

PriorityQueue<DocumentQ> RankQuery(HashMap<String, Integer> queryTerms)
מתודה אשר מדרגת את המסמכים ומחזירה את ה50 מסמכים עם הדירוג הכי גבוה.

void BM25Algorithm(HashMap<String, Integer> queryTerms)
מתודה אשר מממשת את האלגוריתם BM25. האלגוריתם נותן משקל גבוה יותר למילים המופיעות בכותרת המסמך, ולישויות. בנוסף מתחשבת באופן מזערי במילים הסמנטיות-במידה והמשתמש בחר להשתמש בהם.

void get50RelevantDocs()
המתודה מחזירה את 50 המסמכים המדורגים הכי גבוה.

double calculateIDF(String term)
המתודה מחשבת את הפרמטר IDF.

void readDocsDictionary(String path)
המתודה קוראת ממילון המסמכים את כל הפרטים על המסמכים ששמנו בקבוץ הפוסטינג.

מחלקת Searcher:

המחלקה אחראית על חיפוש המסמכים הרלוונטיים עבור שאילתה ומדרגת אותם. מחזירה תשובות לשאילתות כאשר היא שולחת את השאילתה לפרסור ומחזירה את המסמכים הרלוונטיים ביותר לשאילתא באופן מדורג. (עד 50 מסמכים עבור כל שאילתה).

בנוסף, המחלקה משתמשת באלגוריתם לטיפול סמנטי שמטרתו להבין את הקשר הסמנטי בין השאילתה למסמכים ולמצוא מילים בשאילתה מילים נרדפות אשר יעזרו לאחזור טוב יותר.

שדות:

String **query**

סטרינג לשאילתה.

Parse **parse**

אובייקט מסוג Parse שמאפשר לפרסר את השאילתות.

String **path**

נתיב שבו נשמרים קבצי הפוסטינג והמילון.

HashMap<String, Integer> **queryTerms**

מבנה נתונים מסוג HashMap השומר לכל מילה את כמות ההופעות שלה.

Indexer **indexInstance**

אינסטנט למחלקה Indexer שמתוכה נקרא את קבצי הפוסטינג והמילון.

boolean isStem

אופציה לבחירת המשתמש האם לבצע סטמינג או שלא.

boolean isSemantic

אופציה לבחירת המשתמש האם לבצע סטמינג או שלא.

HashMap<String,DocumentQ> **docAndListOfTerms**

מבנה נתונים מסוג HashMap השומר לכל מזהה מסמך אובייקט מסוג DocumentQ המכיל פרטים על אותו מסמך.

HashMap<String,Integer> **numOfDocsToTerm**

מבנה נתונים מסוג HashMap השומר לכל מילה בשאילתא את מספר המסמכים שהיא הופיעה בהם.

PriorityQueue<DocumentQ> **rankDocuments**

מבנה נתונים מסוג PriorityQueue השומר את המסמכים המדורגים.

Ranker **rankAlgorithm**

אובייקט מסוג Ranker האחראי על דירוג המסמכים.

int queryNum

משתנה השומר את מזהה השאילתה.

boolean online;

אופציה לבחירת המשתמש האם להריץ את המנוע עם חיבור לאינטרנט או לא.

<u>מתודות:</u>

void Search(HashSet<String> stopWords)

מתודה ראשית במחלקה אשר אחראית על חיפוש מסמכים רלוונטיים לשאילתה.

void parseQuery(HashSet<String> stopWords)

מתודה אשר אחראית על פירסור שאילתה.

void getQueryDocs()

מתודה אשר אחראית על חיפוש במילון המסמכים ששמרנו כקובץ פוסטינג את המסמכים שמכילים מילים מתוך השאילתה.

void findSemanticWords()

מתודה אשר משתמשת בAPI למציאת מילים נרדפות\דומות למילים בשאילתה.

ד. האלגוריתמים במנוע:

<u>הערה:</u> כל דוגמאות ההשפעה שנציג לא כוללות ביצוע סמנטיקה וסטמינג.

אלגוריתם BM25:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

$$\mathrm{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

<u>רכיבי הנוסחה:</u>

f(q_i,D)- כמות המופעים של המילה הנוכחית במסמך הנוכחי.

$k_1 = 1.25$

b= 0.5

D- אורך המסמך הנוכחי.

avgdl- ממוצע אורך המסמכים.

df-n(q_i)- כמות המסמכים בהם מופיעה המילה הנוכחית.

N-מספר המסמכים.

הנוסחה סוכמת את המשקלים עבור כל מילה משותפת למסמך ולשאילתה ובונה מכך את דירוג המסמך.

השתמשנו בערכים אלה לפרמטרים של k_1 ו- b לאחר הרצת התוכנית עבור כל ערך מהטווח האפשרי עבור k_1 ועבור b בקפיצות של 0.05 במטרה למצוא את ערכי ה- k_1 וה- b האופטימליים עבור מדדי הדירוג שבחרנו.

דוגמאות להשפעה לפי תוצאות התוכנה treceval:

עבור הפרמטרים k,b שבחרנו :

Rel_ret	מדד MAP
141	0.0471

עבור הפרמטרים k=1.45,b=0.6 שלא נבחרו :

Rel_ret	מדד MAP
133	0.045

מסקנה: עבור הפרמטרים k,b שבחרנו, תוצאות האחזור השתפרו.

הוספנו לדירוג את הרכיבים:

- מילה המופיעה בכותרת המסמך- מקבלת משקל גבוה יותר.

לפני

Rel_ret	מדד MAP
135	0.0450

- מילה שהיא יישות- מקבלת גם היא משקל גבוה יותר.

הוספנו מאפיינים אלו מאחר והנחנו שאם מילה מופיעה בכותרת המסמך או שהיא יישות אז סביר להניח כי חשיבותה עולה על חשיבות המילים האחרות ולכן רצינו לתת לכך התייחסות.

- מאפיין נוסף הוא מספר המילים המופיעות עם במסמך וגם בשאילתה. למאפיין זה נתנו משקל מועט מכיוון שזהו מאפיין חשוב אך לא משמעותי.

Rel_ret	מדד MAP
140	0.0460

- במידה והמשתמש בחר באופציית הסמנטיקה, הוספנו את המילים למאגר המילים כאשר מאחזרים את השאילתה אך משקלם היה נמוך מהשאר.

הגענו למסקנה כי שימוש בנוסחת CosSim שנלמדה בתרגול לא הניבה תוצאות טובות ששיפרו את האחזור ולכן בחרנו שלא להשתמש בה באלגוריתם הדירוג.

אלגוריתם למציאת 5 הישויות הדומיננטיות במסמך:

בחרנו לדרג את הישויות במסמך עפ"י ערך ה-tf של מילה במסמך.

המימוש של האלגוריתם מתבצע במחלקת ה-Indexer בעזרת המתודה writeEntitiesPosting אשר מקבלת מהפרסר את מילון המילים הזמני עבור כל מסמך ורושמת לקובץ פוסטינג בשם entites.txt אם המילה מתחילה באות גדולה, אלו המילים שאנו חודשים שהן ישיות.

בסוף תהליך האינדוקס אנו בודקים בעזרת המתודה שנמצאת האינדקסר findTop5Entities האם המילה שרשמנו בקובץ הפוסטינג entities.txt נמצאת במילון אשר מכיל את כל היישיות בקורפוס, במידה וכן נוסיף אותה לתור עדיפויות ומתחזק את 5 הישויות הדומיננטיות במסמך. התור מבצע השוואה ע"י כמות המופעים של המילה במסמך. התור מכיל תמיד 5 מילים ובכל הכנסה בודקים אם התור קטן מ5 אז נכניס את הישות. אחרת, נבדוק האם הישות עם ערך tf גדול יותר מערך ה tf של הישות היוצאת מהתור (שהיא הישות עם ה tf הכי קטן), במידה וכן נכניס אותו במקום הישות היוצאת.

<u>דוגמא להשפעה:</u>
במידה והמסמך מכיל מילה ששייכת לשאילתה והיא אחת מ-5 הישויות הדומיננטיות במסמך, העלנו את דירוג המסמך ב-40%.
מספר המסמכים שאוחזרו עלה ב-2, זוהי השפעה לא משמעותית אך ראינו כי אם נעלה את דירוג המסמך ביותר מ-40% תוצאות האיחזור יורדות.

אלגוריתם לשיפור סמנטי:

המימוש של האלגוריתם מתבצע במחלקת ה-Semantic ע"י שימוש ב- API בעל כתובת ה-URL : https://api.datamuse.com/words?ml=

אם המשתמש מריץ שאילתv עם שיפור סמנטי, האלגוריתם עובר על ערך ה-String של השאילתה ועבור כל מילה בשאילתv ניגש ל API הנ"ל ובודק האם קיימות מילים נרדפות לאותה מילה. אם קיימת מילה נרדפת, האלגוריתם מוסיף אותה לשאילתה.

אלגוריתם נוסף בו השתמשנו לטיפול הסמנטי הוא word2vec. אלגוריתם זה מקבל את הטקסט של הקורפוס ומייצר מרחב ווקטורי ,כאשר לכל מילה ייחודית בקורפוס מוקצה וקטור במרחב.

הוחלט לתת לאותה מילה משקל נמוך יותר בנוסחת ה-BM25 (מכפילים את המילה פי 0.5) מאשר מילה מקורית בשאילתה מאחר ולדעתנו צריך לאזן בין המילים הנרדפות החוזרות מהסמנטיקה לבין המילים המקוריות שבשאילתה.

<u>דוגמא להשפעה:</u>

Rel_ret	מדד MAP
142	0.0589

ה. הנתונים בקובץ ה-posting ובמילון התומכים באלגוריתמים שמימשנו:

כל מילה בשאילתה חפשנו בקובץ הפוסטינג dictionary.txt ובמידה והיא קיימת השתמשנו במידע שלה שכולל את שם קובץ הפוסטינג שבו שמרנו אותה ומספר השורה שלה. בנוסף בתדירות שלה ששמרנו גם שם השתמשנו באלגוריתם בBM25. בעזרת מידע זה נגשנו לקובץ הפוסטינג המתאים שבו שמורה מילה ושורה שבה היא ממוקמת בעזרת פעולת get פשוטה.

בקובצי הפוסטינג A-Z.txt ו Symbols&Numbers# לכל מילה ששמרנו שם ומצאנו בעזרת האמור לעיל, השתמשנו באחזור המסמכים שלה ודירוגה אם הופיעה בשאילתה.

בקובץ הפוסטינג documnetDic.txt חפשנו מסמך לפי מזהה מסמך ששמרנו והשתמשנו בפרטים ששמרנו לו: אורך המסמך בשביל הנוסחה BM25 וכותרת המסמך בשביל לדרג את המסמך לפי השאילתא.

בקובץ הפוסטינג entities.txt עבור כל מזהה מסמך ששמרנו בו השתמשנו ברשימת ישיות "חשודות" והתדירות שלהן במסמך על מנת למצוא את 5 הישיות הדומיננטיות ביותר במסמך.

ו. שימוש בקוד פתוח:

בחלק זה של הפרויקט השתמשנו ב- API לצורך ביצוע סמנטיקה:

https://api.datamuse.com/words

עשינו שימוש בקוד זה במחלקת ה-Semantics כשהמשתמש בוחר באופציה של שיפור סמנטי. פרטנו לגבי המחלקה בפירוט בסעיף 1.ג לעיל.

2. הערכת ביצועי המנוע:

הרצה ללא stemming:

מדד MAP: 0.0471

מספר שאילתה	המילים בשאילתה	Precision לשאילתה	Recall לשאילתה	precision@5	precision@15	precision@30	precision@50	זמן ריצה (דקות)
351	Falkland petroleum exploration	0.2917	0.291	0.2	0.3333	0.2667	0.2917	0.0234
352	British Chunnel impact	0.0203	0.0203	0.4	0.2667	0.1333	0.0203	0.0341
358	blood-alcohol fatalities	0.2353	0.2352	0.2	0.4	0.2667	0.2353	0.0111
359	mutual fund predictors	0.0714	0.0714	0	0.1333	0.0667	0.0714	0.05145
362	human smuggling	0.1795	0.2307	0.4	0.2667	0.2	0.1795	0.02013
367	piracy	0.0541	0.054	0	0	0.0667	0.0541	0.00782
373	encryption equipment export	0.1875	0.3125	0	0.2	0.1667	0.1875	0.01901
374	Nobel prize winners	0.1078	0.1078	0.8	0.4667	0.5333	0.1078	0.012303
377	cigar smoking	0.1667	0.25	0	0.0667	0.1667	0.1667	0.010625
380	obesity medical treatment	0.1429	0.7142	0	0.2667	0.1333	0.1429	0.016217
384	space station moon	0.1373	0.1372	0.2	0.1333	0.1	0.1373	0.02181
385	hybrid fuel cars	0.1882	0.1882	0	0.0667	0.2	0.1882	0.016217
387	radioactive waste	0.0822	0.0821	0	0.1333	0.1	0.0822	0.01006
388	organic soil enhancement	0.16	0.16	0	0.2	0.2	0.16	0.012862
390	orphan drugs	0.0902	0.0901	0.4	0.2	0.3	0.0902	0.012303

הרצה עם stemming:

מדד MAP: 0.0589

מספר שאילתה	המילים בשאילתה	Precision לשאילתה	Recall לשאילתה	precision@5	precision@15	precision@30	precision@50	זמן ריצה
351	Falkland petroleum exploration	0.3333	0.3333	0	0.3333	0.3	0.3333	0.02461
352	British Chunnel impact	0.0203	0.0203	0.4	0.2667	0.1333	0.0203	0.032995
358	blood-alcohol fatalities	0.3137	0.3137	0.2	0.4	0.3333	0.3137	0.013421
359	mutual fund predictors	0.0357	0.0357	0	0.0667	0.0333	0.0357	0.039706
362	human smuggling	0.1282	0.1794	0	0.1333	0.1333	0.1282	0.074379
367	piracy	0.0432	0.0432	0	0	0.0667	0.0432	0.008948
373	encryption equipment export	0.3125	0.375	0	0.3333	0.2	0.3125	0.022929
374	Nobel prize winners	0.0931	0.0931	0.6	0.4	0.3333	0.0931	0.01454
377	cigar smoking	0.2778	0.4444	0	0.2	0.2	0.2778	0.011744
380	obesity medical treatment	0.1429	0.714	0	0.2667	0.1333	0.1429	0.016218
384	space station moon	0.1373	0.1372	0.2	0.1333	0.1333	0.1373	0.025166
385	hybrid fuel cars	0.1647	0.1647	0.6	0.4	0.3667	0.1647	0.019014
387	radioactive waste	0.0959	0.0958	0	0	0.1667	0.0959	0.0095071
388	organic soil enhancement	0.12	0.12	0.2	0.3333	0.2	0.12	0.029080
390	orphan drugs	0.0738	0.0737	0.6	0.4667	0.3	0.0738	0.01454

3. סיכום

קשיים ודרך התמודדות:

- זמני ריצה:

על מנת לשפר את זמני הריצה עשינו שימוש נרחב במבני נתונים מסוג hash ת המאפשרים לנו חיפוש ושליפה ב O(1) . דבר המשפר משמעותית את זמן הריצה בייחוד על קורפוס ומילון גדולים מאוד.

- זיכרון:

לעיתים נתקלנו בהרצת העבודה בשגיאות מסוג garbage collector ו-heap המעידות על ניצול מקסימלי של הזיכרון בRAM. על מנת לפתור בעיה זו, שמרנו קבצים נוספים בDISK שיעזרו לנו לשמור נתונים ולגשת אליהם במהלך ריצת התוכנית.

- דירוג המסמכים:

עיקר העבודה בפרויקט זה היא החזרת מסמכים רלוונטיים עבור שאילתה. בהרצות הראשונות של התוכנית נתקלנו באחזור מאוד לא טוב עבור השאילה- החזרת מסמכים לא רלוונטיים ואי החזרת המסמכים הרלוונטיים.

על מנת להתמודד עם בעיה זו ולאחזר טוב יותר שיפרנו את אלגוריתם BM25 הבסיסי:

- שינוי הפרמטרים k ו-b
- הוספת מאפיינים נוספים לאלגוריתם הדירוג התורמים להעלאת מספר אחזור המסמכים הרלוונטיים כותרת המסמך, ישויות, מספר המילים המופיעות גם בשאילתה וגם במסמך, וסמנטיקה (לפי בחירת המשתמש).
- שיפור אלגוריתם הפירסור על מנת למצוא מילים טובות יותר לאחזור.

האתגר בפרויקט:

נשים לב כי קיים trade-off בין שיפור זמן הריצה למגבלת הזיכרון. כתיבה וקריאה לדיסק לוקחות זמן רב יותר משמירה בRAM. ולכן השתדלנו להימנע מכך כמה שיותר ויחד עם זאת לא להגיע למגבלת הזיכרון בRAM.

בנוסף, מציאת האלגוריתם האופטימלי ביותר לאחזור מסמכים היוותה קושי רב בעבודה. יש לאחזר כמה שיותר מסמכים רלוונטיים ולהחזיר בזמן ריצה הקצר ביותר.

המלצות לשיפור האלגוריתמים:

ניתן לשפר עוד אלגוריתמי הפירסור והדירוג , אך מקוצר זמן שיפרנו אותם עד כמה שיכולנו ושהמנוע יאחזר תוצאות המספקות אותנו.

- אלגוריתם הפירסור-יכולנו לדאוג לתיקון שגיאות כתיב, מציאת ישויות בצורה טובה יותר, התייחסות לסימני הפיסוק השונים ולהקשר בין המילים והסמוכות.
- אלגוריתם הדירוג- ניתן להוסיף מאפיינים נוספים לדירוג כמו: תיאור, תאריך, שפה וכדומה. ובנוסף לנסות להגיע לערכי b ו-k הנותנים תוצאות אחזור טובות יותר.

הערכת ביצועים: ניתן לראות עפ"י השוואת מדדי תוכנת הtreceval כי אלגוריתמי הסטמינג והסמנטיקה משפרים את ביצועי המנוע, ובעזרתם המנוע מחזיר יותר מסמכים רלוונטיים.