מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

בניית מנוע לאחזור מסמכים – חלק ב'

a. הסבר מפורט על אופן פעולת המנוע - אם הוספתם מחלקות יש להסביר את מטרתן ואיך הן פועלות.

אופן פעולת המנוע:

ע"מ להפעיל את המנוע בתהליך האחזור בחלק ב' על המשתמש לבחור נתיב לתיקייה המכילה את כלל הקבצים שהוזכרו לעיל. לאחר מכן נטען את מילון המילים ומילון המסמכים לזיכרון באמצעות הכפתור Load Dictionaries . קיימת קופסת בחירה בשם stemming. במידה ונסמן אותה, ייטענו לזיכרון המילונים הרלוונטיים ל-stemming.

בסיום הטעינה, המשתמש יוכל לבצע reset לתוכנית (ואז יצטרך ליצור אינדקס מחדש ע"י כפתור Create Index ומעבר לחלון של יצירת האינדקס מחלק א') או להציג את מילון המילים, ע"מ להתקדם ולהריץ שאילתה מסוימת עליו לבחור בכפתורי הרדיו את רצונו – שאילתה חופשית או קובץ שאילתות.

במידה ובחר בשאילתה חופשית, יזין את השאילתה בתיבה וילחץ RUN, במידה ומעוניין להריץ קובץ שאילתות יבחר בכפתור התחתון, יבחר נתיב לקובץ הרצוי וילחץ RUN.

מתחת לתיבות ההזנה לשאילתות ישנן קופסאות בחירה נוספות הקשורות לסמנטיקה, במידה ונסמן v בתיבת ה-semantics נבצע סמנטיקה למילים בשאילתה ע"ב מודל Word2Vec , במידה ונסמן גם את התיבה השנייה Online אז נבצע סמנטיקה ע"פ מילים שנקבל ע"י API מסוים שיפורט בהמשך ולמעשה נבצע סמנטיקה באמצעות חיבור לרשת.

בשלב זה המנוע יחל באחזור המסמכים הרלוונטיים ויציג מסך תוצאות ובו נוכל לברור מבין השאילתות שהוכנסו, כשנבחר שאילתה מסוימת יוצגו לנו המסמכים הרלוונטיים לשאילתה. לאחר מכן ישנה אופציה ללחיצה על מסמך מסוים ויוצגו לנו 5 הישויות המדורגות ביותר במסמך זה(במידה וקיימות).

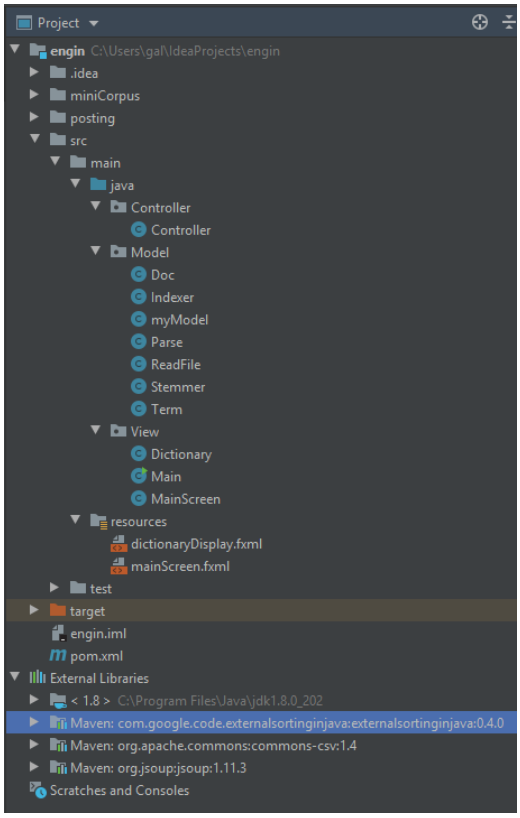
המשתמש יוכל להזין נתיב לתיקייה בה ישמור את קובץ הפלט בפורמט TRAC-EVAL ע"מ לבצע השוואות וניתוח בהמשך לתוצאות.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

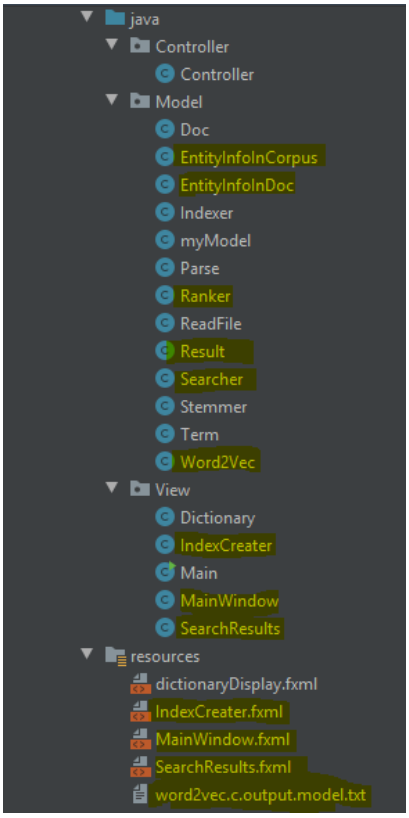
b. יש לכלול הסבר מפורט של כל המחלקות הרלוונטיות לחלק זה (אין צורך להסביר על מחלקות שבניתם בחלק א' אך לא ביצעתם בהן שינויים במסגרת ביצוע חלק זה).

מחלקות שנוספו לפרויקט:

הפרויקט בחלק א'



הפרויקט בחלק ב'



מבני נתונים שנוספו/עברו שינוי במחלקות מחלק א':

במחלקה Doc :

השדה topFiveEntities כעת מסוג TreeMap<String,Double> כך שנוכל להכניס אליו ישויות ודירוג לכל ישות ולמיין לפי סדר ע"י התכונה של המבנה.

שיטה ששונתה לקבלת הישויות מהשדה של טופ5 היא getTopFiveEntities() כך שכעת נקבל את הישויות שנמצאות במבנה הנתונים – גם אם יש שם פחות מ-5.

שיטה נוספת ששונתה היא setEntitiesList שהפכה ל-insertToTop5 , למעשה שיטה זו מקבלת ישות ודירוג ומכניסה למבנה הנתונים topFiveEntities .

במחלקה Indexer :

נוסף שדה entitiesDictTemp HashMap<String,EntityInfoInCorpus> כאשר המחרוזת היא הישות ובאובייקט מידע על ישות בקורפוס נשמור באיזה מסמך הופיעה הישות ואת כמות המופעים במסמך זה. (ע"מ לוודא שישות מופיעה במסמך אחד או יותר).

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

פונקציה שנוספה בהקשר של שדה זה היא clearEntitiesDictionary שתפקידה להסיר את הישויות אשר מופיעות פעם אחת בלבד בכל הקורפוס, בנוסף היא מעדכנת את מילון המסמכים במידע המעודכן של הישויות הנמצאות בכל מסמך, לאחר מכן עוברת על מילון המסמכים וממיינת את הישויות בכל מסמך ומשאירה את 5 הישויות עם הדירוג הגבוה ביותר, כמו כן מעדכנת לכל מסמך את 5 הישויות המובילות בו.

נוספה השיטה sortByValues שבעצם מקבלת אובייקט מסוג Map וממיינת אותו לפי ה-Value ולא לפי Key כך שנדרג את הישויות ע"פ הדירוג שלהם ולא ע"פ שמן.

במחלקה Parse :

האובייקט entitiesInDoc שהיה בשיטה parse עבר להיות שדה: Map<String,Integer> entitiesAndFreqInDoc כשבעצם לכל מסמך הוא שומר ישות וכמות המופעים שלה במסמך.

נוספה השיטה CheckIfEntityAndUpdateFreq שהיא בעצם החליפה את תהליך זיהוי הישויות שהיה בתוך השיטה parse והיא מזהה ישויות כרצף של 2 מילים או יותר (עד 4) כך שכולן מתחילות באות גדולה ופסיק או נקודה לא מפרידים בין המילים.

במחלקה ReadFile :

הוספנו שדה שיסדר לנו את כל המסמכים בקורפוס, HashMap<Doc, EntityInfoInDoc> allDocsInCorpus

כשבשיטה readFile אנו מוסיפים לו כל מסמך שחוזר מפרסור עם ישות NULL ומעדכנים ישויות לאחר מכן ב-INDEXER.

המחלקה MainScreen שונתה ל- IndexCreater כך שאם נרצה ליצור שוב אינדקס נחזור לחלון הזה והוא יהיה רק חלק מהחלון שבו מנוהל תהליך ההצגה של המנוע.

במחלקה myModel :שדות שנוספו:

repeatedPatterns List<Pattern> – מכיל מס' דפוסים שחוזרים על עצמם בשאילתות, השתמשנו במחלקה Pattern כדי לחלץ את המידע הרלוונטי מבין התגיות של קובץ השאילתות ע"י זיהוי הדפוס החוזר, מתן ביטוי רגולרי אשר מקבל את המלל החשוב ובסיום דפוס המסיים את החלק החשוב. לדוגמא:

עבור שאילתא מספר 351 נסתכל על חלק ה-NARRATIVE:

Any document discussing petroleum exploration in the South Atlantic near the Falkland Islands is considered relevant.

נשים לב כי מה שמסומן בצהוב למעשה חוזר על עצמו במס' שאילתות לכן ניתן להבין שמה שנמצא בין החלקים הצהובים חשוב לנו לפרסר ולהתייחס אליו בעת מציאת מסמכים רלוונטיים לשאילתה.

שדה Word2Vec המחזיק אובייקט מסוג זה. נעשה שימוש בספריית MAVEN חיצונית לסמנטיקה במצב של ללא חיבור לאינטרנט, מחזיק מאגר מידע לגבי דמיון בין מילים שונות.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

שדה resultsOfQueries < TreeMap<Integer , Vector<String> שמחזיק את התוצאות של השאילתות כאשר במפתח נחזיק את מס' הזיהוי של השאילתה ובערך נחזיק ווקטור של שמות המסמכים הרלוונטיים לשאילתה – עוזר בהצגה למשתמש לאחר מכן ב-GUI.

שדה ArrayList<Result> results – מחזיק את התוצאות לכל שאילתא, בהמשך נראה כיצד בנוי אובייקט ה-Result.

<u>נוספו מס' שיטות שנועדו לנהל את תהליך האחזור של השאילתות:</u>

```
public ArrayList<Result> processSingleUserQuery(String query,
boolean isSmanticsOnline, boolean isSemantics, boolean
userChoseStemming){
```

השיטה מקבלת שאילתה חופשית מהמשתמש בנוסף להגדרות להרצת השאילתה – שימוש בסטמינג או סמנטיקה. השיטה מגרילה לשאילתא של המשתמש מספר זיהוי רנדומלי בין 500-999, מגדירה את האובייקטים להגדרות שקיבלה ושולחת את הטיפול לפונקציה הבאה.

```
private ArrayList<Result> processSingleQuery(String qID, String
query, boolean isSmanticsOnline, boolean isSemantics, boolean
userChoseStemming) {
```

השיטה בונה את השאילתה, במידת הצורך מוסיפה לה מילים נרדפות לכל מילה מהשאילתה המקורית(אם בחרנו סמנטיקה), ומאתחלת מבני נתונים לתוצאות וקוראת לפונקציה getRelvantDocs של מחלקת searcher . כשמתקבלות התוצאות מכניסה אותם למבני הנתונים הרלוונטיים ומחזירה אותן.

```
private String getSemanticTermsToQueryTerms(String query) {
```

השיטה למעשה עונה על הגדרת הסמנטיקה במידה ובחרנו Online , השיטה ניגשת לאתר https://api.datamuse.com/words?rel_syn= ועוברת על כל מילה בשאילתה, שולפת מילים נרדפות מהאתר ומוסיפה את המילים הללו לשאילתה ע"מ למצוא מסמכים נוספים שיכולים להיות רלוונטיים.

```
public ArrayList<Result> processFileOfQueries(String
pathToQueryFile, boolean isSemanticsOnline, boolean isSemantics,
boolean userChoseStemming){
```

השיטה מקבלת נתיב לקובץ והגדרות להרצת השאילתה. היא מכניסה למילון הדפוסים את הדפוסים הרלוונטיים, מגדירה אובייקטים. בנוסף עוברת על קובץ השאילתות, מפרקת את השאילתות ע"פ התגיות המתאימות – top,num,title,desc,narr ויוצרת מחרוזת המכילה את המידע הנשלף מכל החלקים הללו לכל שאילתה. עבור המידע המגיע מה-Narr אנו מפעילים את השמטת הדפוסים הלא רלוונטיים ומקבלים מילים נוספות המעלות את הרלוונטיות של השאילתה ומשפרות את התוצאות המאוחזרות. בסוף השיטה נקרא לפונקציה אשר מעבדת את השאילתה שפירטנו עליה לעיל.

```
public void writeResultsToFile(String path, boolean
userChoseStemming) {
```

השיטה כותבת לקובץ את התוצאות שקיבלנו בפורמט התואם ל-TRAC-EVAL כך שנוכל לבצע השוואות בעזרת התוכנה. נכתוב לקובץ המתאים בעזרת הבחירה של המשתמש לביצוע סטמינג.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

```
private void insertPatterns() {
```

מעדכנת את רשימת הדפוסים שלנו, למשל מכניסה דפוס בצורה הבאה:

```
Pattern p1 = Pattern.compile(Pattern.quote("document discussing") +
        "(?s)(.*?)" + Pattern.quote("is relevant."),
Pattern.CASE_INSENSITIVE);
```

למעשה אנו קובעים שביטוי רגולרי מסוג כלשהו הנמצא בין שני הדפוסים ייחשב לנו כרלוונטי.

```
private String eliminatePatterns(String content){
```

השיטה מקבלת את תוכן התגיות של שאילתה, למשל Narr ועוברת על כלל הדפוסים שלנו ומחלצת מידע רלוונטי שעבורו נרצה לבצע גם חיפוש למסמכים רלוונטיים.

מחלקות שנוספו בחלק ב':

המחלקה EntityInfoInCorpus :

מחזיקה מידע על ישות מסוימת המופיעה בקורפוס.

שדות:

```
private HashMap<Doc,Integer> docList; // < docID , EntityFreqInDoc >
private int numOfDiffDocs;
private HashMap<Doc,Double> rankInDoc; // <docID, rank>
```

שיטות המחלקה הן getters & setters לשדות ובנאים.

המחלקה EntityInfoInDoc :

מחזיקה מידע על ישות המופיעה במסמך.

```
private HashMap<String , Double> entitiesAndFreqInDoc; // <entity ,
freq>
```

שיטות המחלקה הן getters & setters לשדה ובנאים ושיטה המכניסה להאשמאפ.

המחלקה Result :

מחזיקה מידע על תוצאה רלוונטית החוזרת עבור שאילתה מסוימת.

```
private String queryId;
private String docName;
private String queryTitle;
private double rank;
```

שיטות המחלקה הן getters & setters לשדות ובנאים.

המחלקה Word2Vec :

מחלקה המשתמש בספריית מייבן עבור סמנטיקה בשימוש אופליין, המחלקה בעלת שיטה אחת.

```
public String similarTerms(String term) {
```

השיטה מקבלת מילה עבורה צריך למצוא מילים נרדפות, היא פותחת את הקובץ:

word2vec.c.output.model.txt המכיל מילים בשפה עם מטריצת דירוגים. השיטה שולפת את 3 המילים הדומות ביותר לבקשתנו, נבחר את המילים במידה והדמיון ביניהם עולה על 0.95 כך שרק מילים מאוד רלוונטיות יחזרו לשאילתה.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

המחלקה Ranker :

המחלקה אחראית על דירוג המסמכים המועמדים להיות רלוונטיים.

```
private Map<String, String> documentsDictionary;
private double k;
private double b;
private int avgDocLength;
```

מילון מסמכים – נטען מהקובץ שנכתב בקבצי האינדקס.

אורך ממוצע למסמך – מחושב בעת טעינת המילון.

שיטות במחלקה:

```
public TreeMap<String, Double> rankDocuments(Parse parser,
ArrayList<Pair<String, String>> postingDataOfAllTermsInQuery) {
```

מקבלת אובייקט פרסר שיבצע פרסור למידע הנמצא בכותרת המסמך, בנוסף מקבלת מערך של זוגות כאשר כל זוג מכיל term|docno,tfInDoc . השיטה מחלצת את המידע של המסמכים המועמדים להיות רלוונטיים, מחפשת אותם במילון המסמכים, במידה ונמצא כזה בודקת האם מילות השאילתה נמצאות בראש המסמך, האם הן מופיעות בכותרת המסמך והאם הן שייכות לאחת מחמשת הישויות המדורגות במסמך, כל אלו ישמשו לאחר מכן למציאת הדירוג המסמך. בסוף נחזיר TreeMap עם שמות המסמכים והדירוגים של כל אחד.

```
private double getFinalRank(int tfInQuery, int docFreq, int docLen,
int queryTermInHeadDoc, int queryTermInTitle, int tf, int
queryTermRelatedToEntity) {
```

מחשבת את דירוג המסמך בעזרת כל הפרמטרים, נרחיב בהמשך.

```
private int checkQueryTermRelatedToTopEntity(String entities, String
postingOfWordFromQuery) {
```

אם מילה מוכלת בישות מאחת הישויות המדורגות במסמך השיטה תחזיר 1 אחרת תחזיר 0.

```
private int checkQueryTermInTitle(Parse parser, String docName,
String postingOfWordFromQuery) {
```

אם מילה נמצאת בכותרת המסמך השיטה תחזיר 1 אחרת תחזיר 0.

```
public TreeMap<String, Double> getTopEntries(int numOfEntries,
TreeMap<String, Double> from) {
```

השיטה תחזיר את כמות האובייקטים המבוקשת המדורגים הכי גבוה בפרמטר from.

```
public boolean loadDocDetailsToMemo(boolean selected) {
```

השיטה טוענת את מילון המסמכים המתאים, selected==true יטען לנו את המילון של הסטמינג, אחרת נטען את המילון הרגיל.

המחלקה Searcher :

המחלקה אחראית למציאת המסמכים המועמדים להיות רלוונטיים ולקרוא ל-Ranker שידרג אותם.

```
private Parse parserOfSearcher; //parser object to parse queries
private Indexer indexerForSearch; // indexer object to index the
terms of query
private Ranker ranker; //which ranks the documents
public static String pathToWriteRead; //path for read and write
files
```

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

```
private Indexer indexerOfProgram; // indexer object that holds info about the corpus
```

שיטות המחלקה:

```
public TreeMap<String, Double> getRelevantDocs(String query, boolean userChoseStemming) {
```

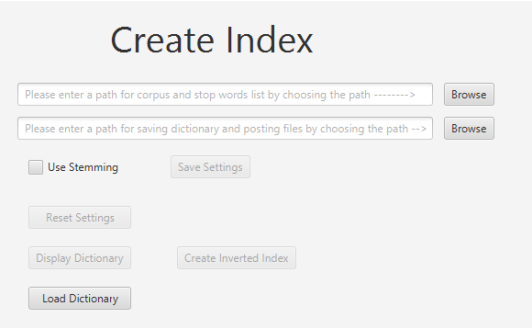
שיטה מרכזית במחלקה, אחראית לקליטת השאילתה, מעבר על כל מילה בה, שליפת המידע על המילים בקבצי האינדקס ושליחת המידע ל-ranker לביצוע דירוג. מחזירה את 50 התוצאות הרלוונטיות לשאילתה.

```
private String getPostingDataFromDisk(String term, long pointer, boolean userChoseStemming) {
```

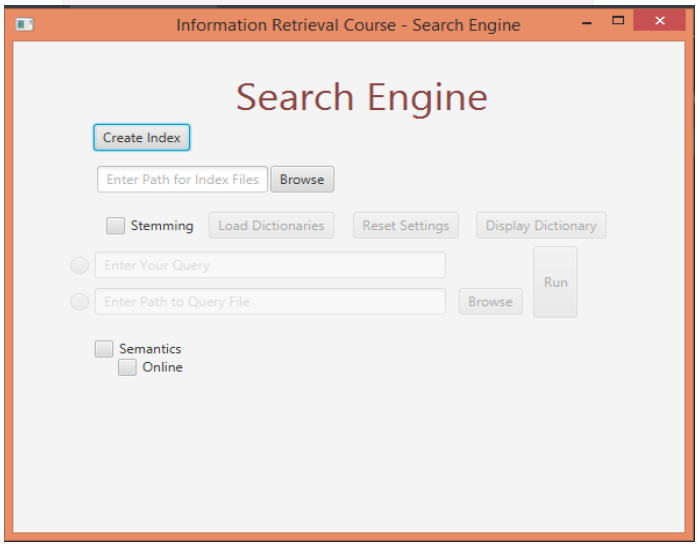
בשיטה זו נעשה שימוש ע"י השיטה הקודמת, שיטה זו אחראית לגשת אל קבצי האינדקס ולשלוף את המידע הרלוונטי של מילה מסוימת מקובץ האינדקס המתאים.

שינויי GUI :

בחלק א' היה לנו חלון אחד שעסק ביצירת האינדקס:



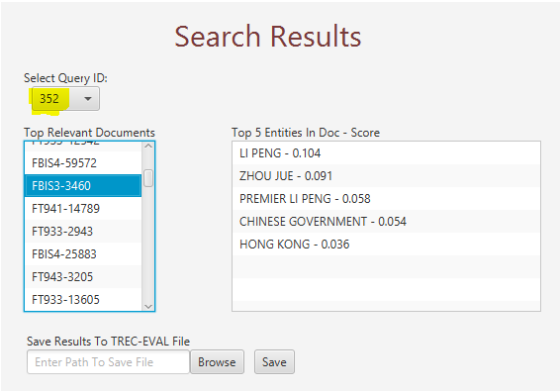
בחלק ב' נוכל להגיע לחלון מחלק א' ע"י לחיצה על כפתור Create Index .



החלון הראשי בחלק ב' יכיל את הפונקציונליות הנדרשות מחלק זה:

- הזנת נתיב לקבצי האינדקס והמילונים לטעינה
- טעינת המילונים עם/בלי stemming
- ביצוע אתחול – מחיקת כלל הקבצים בנתיב שהזנו בשורה העליונה.
- הצגת מילון – לאחר טעינה נוכל להציג את המילון שלנו.
- הזנת שאילתה חופשית/נתיב לקובץ שאילתות
- בחירה בסמנטיקה והאם לבצע אותה אונליין/אופליין.

בסיום ההרצה של השאילתות ייפתח חלון נוסף שבו ניתן יהיה לבחור מס' זיהוי של שאילתה אשר יציג מסמכים שאוחזרו כרלוונטיים. לכל מסמך כזו ישנה אפשרות לחיצה כך שנוכל לראות את הישויות המדורגות הגבוהות ביותר באותו מסמך.



כמו כן, ניתן יהיה לשמור את התוצאות בתיקייה לבחירת המשתמש.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

c. הסבירו בצורה מפורטת את האלגוריתמים הכלולים במנוע. בפרט:
   i. אלגוריתם הדירוג
   ii. אלגוריתם למציאת 5 הישויות הדומיננטיות במסמך, כולל 2 דוגמאות
   iii. אלגוריתם לשיפור סמנטי

<u>אלגוריתם הדירוג:</u>

באחזור המידע במנוע החיפוש השתמשנו באלגוריתם BM25 על מנת לחשב את דירוגו של כל מסמך.
את החישוב ביצענו באמצעות הנוסחה הבאה:

$$f(q,d)=\sum_{w\in q\cap d} c(w,q)\frac{(k+1)c(w,d)}{c(w,d)+k(1-b+b\frac{|d|}{avdl})}\log\frac{M+1}{df(w)}$$

כאשר:

- b=0.75 – בחרנו לאחר הבנה כי זהו הערך הטוב ביותר עבורו.
- בחרנו לחשב באמצעות $\log_{10}\frac{M+1}{df(w)}$
- c(w.d)- מספר הפעמים שמופיע הביטוי w במסמך d.
- c(w.q)- כמות הפעמים שמופיע הביטוי w בשאילתה q.
- |d|- אורך המסמך d (ללא stop words).
- avdl- אורך המסמך הממוצע במאגר המסמכים.
- M- כמות המסמכים במאגר.
- df(w)- מספר המסמכים בהם מופיע הביטוי w במאגר המסמכים.
- בחרנו את k להיות 1.6 לאחר ניסוי וטעיה איזה מספר ייתן לנו את התוצאות הטובות ביותר.

בנוסף, הוספנו לנוסחה חישובים נוספים ע"מ לשפר את הדיוק והנוסחה הכוללת שבה השתמשנו היא:

$$Rank(q,d)=0.5*BM25_{0.75,1.4}(q)+0.2*q\ in\ title+0.2*q\ in\ head\ of\ doc+0.1*q\ related\ to\ entity$$

תהליך הדירוג:

כל שאילתה הורכבה מהכותרת שלה + תיאור השאילתה + מילים רלוונטיות מהנרטיב(מפורט למעלה כיצד עשינו זאת ע"י שימוש בדפוסים חוזרים), עבור כל מילה בשאילתה מצאנו את המסמכים המועמדים להיות רלוונטיים להחזרה למשתמש. חישבנו עבור כל מילה בשאילתה את מדד ה-BM25 , הוספנו 0.2 במידה והמילה נמצאה כחלק מהכותרת של המסמך(מילה הנמצאת בכותרת היא כנראה בעלת משמעות ומשפיעה מאוד על מהות הטקסט), הוספנו 0.2 במידה והמילה נמצאת ב-10% המילים הראשונות של המסמך(מילה המופיעה בפסקה הראשונה בטקסט ככל הנראה בעלת חשיבות גבוהה יותר שכן לרוב הפסקה הראשונה היא פתיח המתמצת את המהות בו הטקסט יעסוק) והוספנו 0.1 במידה והמילה הייתה חלק מישות מסוימת מתוך 5 הישויות המדורגות ביותר במסמך(מילה השייכת לישות המדורגת גבוה, ככל הנראה הופיעה הרבה פעמים או שהישות מספיק חשובה ע"מ לתת תוספת דירוג). סכמנו הכל וקיבלנו דירוג למסמך עבור מילה בשאילתה. את התוצאה שמרנו במבנה נתונים ולאחר מכן ביצענו מיון עבור 50 המדורגים ביותר מבין כל המסמכים שחושבו עבור כל המילים בשאילתה.

במידה והשתמשנו גם בסמנטיקה אופליין או אונליין, הוספנו 2-3 מילים נרדפות נוספות לכל מילה בשאילתה וביצענו דירוג למסמכים שחזרו גם עבור מילים אלו.

מדוע בחרנו באלגוריתם זה: 2 סיבות מרכזיות לבחירה הן עומס על הזיכרון וזמן ההמתנה של המשתמש. ע"מ לממש את נוסחת ה-CosSim נצטרך לעבד שוב את הקורפוס כדי למשקל כל מילה במסמך, לקחנו כשיקול את מהירות המנוע ולא רק את דיוק תוצאות האחזור.

לא היו הבדלים גדולים מדיי בין הערכים השונים שהזנו עבור K .

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

דוגמא: עבור ערכים k=1.2 , b=0.75 והשאילתא 385 נקבל

$$recall(id = 385) = \frac{25}{50} = 0.5$$

עבור ערכים k=1.6 , b=0.75 והשאילתא 385 נקבל

$$recall(id = 385) = \frac{26}{50} = 0.52$$

<u>מציאת ישויות דומיננטיות:</u>

אנו ביצענו פרסור עבור כל מסמך, במהלך הפרסור בדקנו עד 4 טוקנים קדימה האם יש לנו רצף של טוקנים המתחילים באות גדולה כך שזה ייחשב כישות, שמרנו את כל הרצפים הללו במילון ישויות תוך כדי שאנו מעדכנים מס' הופעות לכל ישות במידה ונתקלנו בה יותר מפעם אחת. לאחר מכן הכנסנו את כלל הישויות של המסמך למילון זמני הבנוי ממפתח – שם הישות וערך – אובייקט מסוג EntityInCorpus המכיל מילון עבור <מסמך,כמות הופעות במסמך> שאליו נבצע הכנסה, במידה והמילון האחרון יהיה גדול מ-1 בסוף הפרסור, נקבל שזוהי ישות כיוון שהופיעה ב-2 מסמכים או יותר. האובייקט EntityInCorpus מכיל גם מילון <מסמך,דירוג> עבור דירוג ישות מסוימת במסמך, בשלב זה חישבנו עבור ישות את $\frac{\text{כמות הפעמים שמופיעה הישות במסמך הנוכחי}}{\text{אורך המסמך}}$ והכנסנו למילון הזמני – דירוג ה-tf . (פונקציה אחראית – setStructures במחלקה indexer).

לאחר סיום פרסור כלל המסמכים נעבור על מילון הישויות הזמני ונחלץ ישויות ונבנה מילון מסמכים אשר יכיל מסמך כמפתח ואובייקט מסוג EntityInDoc המכיל מילון <ישות, דירוג הישות במסמך> , עבור כל המסמכים שנוציא מישות מסוימת מהמילון הזמני, נכניס אותם למילון המסמכים בתוספת הישות וחישוב סופי של דירוג הישות, מהפסקה הקודמת יש לנו את מדד ה-Tf של הישות, נכפול אותה ב-$\log_{10}\frac{N}{df}$ ולמעשה נחשב עבור הישות את מדד ה-tf*idf שלה ונכניס למילון. (פונקציה אחראית – clearEntitiesDictionary במחלקה indexer).

בסיום שלב זה נקבל מילון מסמכים כשעבור כל מסמך יש את כל הישויות במסמך מדורגות, נשאיר את 5 הישויות המדורגות הכי גבוה ונעדכן אותן באובייקט Doc שלנו עבור כל מסמך.

דוגמאות לישויות ניתן למצוא בממשק משתמש לאחר הרצה, ניתן לבחור מסמך שחזר לשאילתה וללחוץ עליו ויוצגו הישויות.

<u>אלגוריתם לשיפור סמנטי:</u>

במנוע זה הוצעו למשתמש שתי אפשרויות לשיפור סמנטי, אחת ללא חיבור לאינטרנט ואחת עם חיבור לאינטרנט.

במידה והמשתמש סימן וי ב-GUI , הוספנו לשאילתה 2-3 מילים נרדפות לכל מילה בשאילתה כך שנוכל לקבל גם מסמכים המכילים מילים דומות ואולי נצליח לשפר את הרלוונטיות שלהם.

ללא חיבור לאינטרנט – השתמשנו בספריית קוד פתוח - https://github.com/medallia/Word2VecJava , שמרנו קובץ טקסט המכיל המון מילים ודירוגים עבור כל מילה למילים דומות לה. קראנו לאובייקט מסוג word2vec אשר החזיר לנו רק מילים דומות בדירוג של מעל 0.95 כאשר המקסימום הוא 1. כך וידאנו שיוחזרו מילים עם משמעות דומה ולא מילים אשר יכולים להרוס את סיכויי החישוב לרלוונטיות של מסמך מסוים.

עם חיבור לאינטרנט – השתמשנו בחיבור לאתר https://api.datamuse.com/words?rel_syn= - אשר מקבל בסוף כתובת ה-URL מילה כלשהי ומחזיר רצף של מילים עם משמעות דומה לפי דירוג מסוים, לקחנו 2 מילים ראשונות בעלות דירוג הכי גבוה והוספנו לשאילתה.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

d. יש להסביר על הנתונים בקובצי ה-posting ובמילון התומכים באלגוריתמים שמימשתם.

1. קבצי אותיות – לכל אות שמרנו קובץ בשם finalPostingA למשל ו-finalPostingA_stemmed במידה והמשתמש בחר לבצע stem. קבצים אלו הכילו בתוכם את כל המילים המתחילים באות שבשם הקובץ. 26 קבצים כאלו קיימים בסה"כ – אחד לכל אות בא"ב האנגלי.
2. קבצי מספרים באותו אופן עם קבצי האותיות, שם הקבצים הללו הוא finalPosting0 ו-finalPosting0_stemmed. מכילים את המספרים המתחילים בספרה הראשונה של שם הקובץ בו נמצאים – סה"כ 10 קבצים של הספרות 0-9.
3. קובץ תווים מיוחדים – באותו אופן רק שבקובץ זה יופיעו מילים שעברו פרסור והם מתחילים למשל ב-$ ותווים מיוחדים אחרים שהם לא אות או ספרה – קובץ בודד.

בקבצים הסופיים מסעיפים 1-3 נראה את המידע הבא עבור מילה כלשהי:

FBIS4-40628:1132,1348,1352,1392,1524,1888#6

בכחול – שם המסמך בו הופיעה המילה.

בירוק – המיקומים בהם המילה מופיעה במסמך.

באדום – סך כמות המופעים במסמך.

כאשר ':' מפריד בין מסמכים ו-# מפריד בין מיקומים לכמות מופעים.

המידע הנ"ל זהה בצורתו בכלל קבצי האינדקס, אנו יודעים לומר איזו שורה שייכת לאיזו מילה ע"פ הפוינטר במילון, נמנענו מלכתוב את המילה גם בקבצי האינדקס ע"מ לחסוך בזמן כתיבה.

4. מילון מילים – המכיל את כל המילים בקורפוס אחרי פרסור.
   כל שורה במילון מופיעה באופן הבא:
   Term | term freq in corpus | doc freq in corpus | pointer(line number in index file)
   For Example - rightfully|206|203|37577
5. מילון המסמכים – כל מסמך שעבר בפרסור נשמר בצורה הבאה:

Docno | title | date | city | language | doc length | most frequent term total appearances | number of unique terms | top 5 entities and their rank

For example:

FT932-15027|International Company News: Waste Management sees flat term|15 APR 93|CHICAGO|none|135|7|86|WASTE MANAGEMENT,0.289,WHEELABRATOR TECHNOLOGIES,0.081,RUST INTERNATIONAL,0.08,CHEMICAL WASTE MANAGEMENT,0.074,WMX TECHNOLOGIES,0.071

6. קובץ dictionaryToGUI שמכיל את המילים במילון עם כמות המופעים שלהם, נקרא בעת הצגת המילון למשתמש.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

e. אם השתמשתם במהלך העבודה בקוד פתוח לפרט את השירות, כתובת, היכן השתמשתם, כיצד השתמשתם.

Jsoup- ספריית Java (https://jsoup.org/) המספקת עבודה נוחה עם קבצים המכילים תגיות (כמו HTML). מאפשרת לחלץ ולבצע מניפולציות על הdata בקלות. נעזרו במחלקה זו כחלק מתהליך הפרסור לשם חילוץ מסמכים, טקסט, ונתונים נוספים עבור מסמך כגון שם עיר, שפה, כותרת וכדומה. הקוד פתוח לשימוש וזמין גם בgithub:
https://github.com/jhy/jsoup.git.

מחלקת porterStemmer – כחלק מתהליך ה-parse אנו מבצעים stemming (ע"פ דרישה). אנו לקחנו גרסת java של Porter Stemmer מהאתר:

https://tartarus.org/martin/PorterStemmer/java.txt

Com.google.code.externalsortinginjava – השתמשנו במתודות externalSort למיזוג כל קבצי ה posting הזמניים (פירוט למעלה) (כתובת – "
https://mvnrepository.com/artifact/com.google.code.externalsortinginjava/externalsortingi
njava/0.1.9").

מחלקת Word2Vec – שימוש בקוד פתוח ע"מ לייבא מודל מאומן עם קובץ טקסט המכיל את דירוגי הדמיון בין מילים. כתובת: https://github.com/medallia/Word2VecJava

שימוש ב-API – בכתובת https://api.datamuse.com/words?ml= , אתר המחזיר לכל מילה שנכניס לו כקלט, מספר מילים עם משמעות דומה לצורך שיפור סמנטי.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

הערכה של המנוע:

ללא stemming :

query ID	query title	precision	recall	precision5	precision15	precision30	precision50	MAP	Time (sec)
351	Falkland petroleum exploration	0.42	0.4375	0.2	0.2667	0.4	0.42	0.1624	12.887
352	British Chunnel impact	0.06	0.012195	0.2	0.1333	0.1	0.06	0.0051	47.662
358	blood-alcohol fatalities	0.36	0.352941	0.2	0.4667	0.3333	0.36	0.1393	14.196
359	mutual fund predictors	0.14	0.25	0	0.1333	0.2	0.14	0.0484	12.224
362	human smuggling	0.18	0.230769	0.4	0.2667	0.2333	0.18	0.0894	2.807
367	piracy	0.22	0.059459	0.2	0.0667	0.1	0.22	0.0097	12.591
373	encryption equipment export	0.14	0.4375	0	0.2667	0.2333	0.14	0.117	22.787
374	Nobel prize winners	0.6	0.147059	0.8	0.6	0.6	0.6	0.0991	29.544
377	cigar smoking	0.18	0.25	0	0.0667	0.667	0.18	0.0351	18.057
380	obesity medical treatment	0.1	0.714286	0	0.2667	0.1333	0.1	0.1395	12.64
384	space station moon	0.22	0.215686	0.6	0.3333	0.2	0.22	0.0787	53.094
385	hybrid fuel cars	0.52	0.305882	0.2	0.2	0.4667	0.52	0.1296	29.415
387	radioactive waste	0.18	0.123288	0.2	0.0667	0.2	0.18	0.0221	8.244
388	organic soil enhancement	0.2	0.2	0.2	0.1333	0.2333	0.2	0.0499	5.386
390	orphan drugs	0.28	0.114754	0.4	0.3333	0.3667	0.28	0.0471	43.544
All Queries	total	0.253333	0.153102	0.24	0.24	0.2578	0.253333	0.0782	325.078

עם stemming :

query ID	query title	precision	recall	precision5	precision15	precision30	precision50	MAP	Time (sec)
351	Falkland petroleum exploration	0.46	0.479167	0	0.4	0.4333	0.46	0.1982	17.348
352	British Chunnel impact	0.14	0.028455	0.4	0.2	0.1333	0.14	0.0113	68.121
358	blood-alcohol fatalities	0.32	0.313725	0.2	0.4667	0.3667	0.32	0.1238	21.245
359	mutual fund predictors	0.16	0.285714	0	0.2	0.2	0.16	0.0566	23.164
362	human smuggling	0.14	0.179487	0.4	0.2667	0.1667	0.14	0.0654	4.614
367	piracy	0.28	0.037838	0.2	0.0667	0.1667	0.28	0.0157	16.351
373	encryption equipment export	0.12	0.375	0	0.2	0.2	0.12	0.0835	46.182
374	Nobel prize winners	0.48	0.117647	0.6	0.5333	0.4	0.48	0.0656	39.469
377	cigar smoking	0.32	0.444444	0	0.1333	0.2	0.32	0.1046	32.046
380	obesity medical treatment	0.08	0.571429	0	0.2	0.1	0.08	0.1026	30.848
384	space station moon	0.24	0.235294	0.6	0.2667	0.2	0.24	0.0835	83.49
385	hybrid fuel cars	0.48	0.282353	0.2	0.1333	0.4333	0.48	0.1125	48.689
387	radioactive waste	0.18	0.123288	0.2	0.0667	0.1	0.18	0.0285	18.538
388	organic soil enhancement	0.26	0.26	0	0.2	0.2667	0.26	0.0677	13.77
390	orphan drugs	0.34	0.139344	0.6	0.6	0.5333	0.34	0.0831	69.14
All Queries	total	0.266667	0.16116	0.2267	0.2622	0.26	0.2666667	0.0802	533.015

סיכום:

בעיות שנתקלנו בהן במהלך חלק ב' של הפרויקט והתמודדות עמן:

- אחזור מעט מסמכים רלוונטיים – בתחילה התייחסנו רק לכותרת השאילתה ולתיאור וקבלנו תוצאות מעט נמוכות מהמצופה, מרחב החיפוש של המילים שבדקנו היה קטן מדיי, בחרנו גם לברור את החלקים הרלוונטיים מחלק הנרטיב ע"מ להתייחס לדברים ספציפיים יותר ורלוונטיים יותר ע"י כותב השאילתה. מרחב החיפוש שלנו גדל וכך אחזרנו יותר מסמכים.
- פגיעת טיפול סמנטי בתוצאות – בתחילה לקחנו 5 מילים דומות לכל מילה בשאילתה, מרחב החיפוש גדל משמעותית ולאו דווקא במילים רלוונטיות, אחזרנו גם מסמכים פחות רלוונטיים, החלטנו להוריד את כמות המילים הנרדפות שניקח ל-3 בטיפול אופליין ו-2 בטיפול אונליין, מה שהעלה לנו את כמות המסמכים המאוחזרים.
- חיפוש מילה מהשאילתה באותיות קטנות/גדולות – גילינו שכאשר מחפשים מילה מסוימת מהשאילתה, היא לאו דווקא תופיע כך במילון שלנו, שכן אם נחפש UNITED בשאילתה ובמהלך הפרסור ויצירת האינדקס ראינו גם את המילה united אז שמרנו אותה באותיות קטנות ע"פ הגדרות המנוע, לכן בדקנו מצב שאם המילה לא מופיעה במילון כמו שהיא ננסה למצוא אותה בווריאציה שונה כדי לאחזר מסמכים רלוונטיים שמופיעה בהם המילה אך בתצורה שונה – דבר שהעלה לנו את כמות המסמכים המאוחזרים.

מגישים : גל בוזגלו - 203886528
שי ארץ קדושה - 203276258

אתגרים

- זמני ריצה
בחלק א' בעיקר נאלצנו להתמודד עם זמני ריצה ארוכים, במחלקת Parse היינו צריכים להבין כיצד לעבוד עם מחרוזות באופן מיטבי ולא להשתמש בפונקציית REGEX יותר מדיי כיוון שעם כמה שהיא נוחה היא גוזלת זמן יקר, כמו כן ניסינו לסדר את האת התנאים של כל הפרסור לחוקים בסדר כזה שימזער זמן ריצה(למשל לא נרצה שתאריך יעבור את כל תנאי הפרסור של משקל וכן תאריכים היו נפוצים יותר בטקסט לכן נפרסר תאריך קודם), נוכחנו לדעת כי אנו חייבים בנוסף לנקות מבני נתונים במהלך הריצה שכן אחרת הזיכרון שלנו מתמלא וזמני הריצה מתארכים. במחלקה Indexer היה אתגר נוסף עם זמני הריצה משום שהתמודדנו עם כתיבה וקריאה מדיסק, חילקנו את העבודה עם הקבצים לתרדים כך שנבצע עבודה מקבילים עם קבצים אשר מתעסקים עם מילים מסוג שונה (מילים עם אותיות גדולות או קטנות ומספרים בנפרד) במקום לעשות את התהליך איטרטיבי ולהמתין לכל קובץ שיסיים בנפרד.
- זיכרון ומקום
לא היה ביכולתנו לשמור את כל המידע שפרסרנו ורצינו לשמור בזיכרון ולכן עבדנו עם קבצים, שמרנו בזיכרון התוכנית את מילון המילים(העיקרי בתוכנית) עם מילים ומצביעים לקבצים כך שנוכל לשלוף את המידע שאנו רוצים בקלות משום שלא ניתן להחזיק את כל קבצי האינדקס למשל בזיכרון בשל משקלם. בחלק ב' של אחזור המסמכים שלפנו את המסמכים המועמדים להיות רלוונטיים בלבד ע"י השימוש במצביעים ממילון המילים וגישה למידע על מסמכים אלו בלבד.

האתגר המרכזי בפרויקט הוא למעשה שילוב האתגרים לעיל, מציאת האיזון האופטימלי שבין ריצה מהירה עם זיכרון מלא לבין הרצה איטית יותר עם זיכרון יותר פנוי בשילוב שיפור איכות התוצאות הסופיות של אחזור המסמכים הרלוונטיים באמת.

שיפורים אפשריים לתוכנית:

- מציאת ישויות באופן מהיר יותר – הגדרנו ישויות רק לאחר שעברנו על כל המידע שעבר פרסור ע"מ לראות האם ישות מסוימת נמצאת ב-2 מסמכים או יותר, בפועל ניתן למקבל את זה כך שישות שמופיעה בפעם השנייה כבר אפשר להכניס אותה למילון ישויות סופי.
- סמנטיקה – מימוש טוב יותר ל-Word2Vec ע"י יצירת מודל המאומן בזמן ריצה וע"י כך לשפר את הסמנטיקה ב-אופליין, כמו כן אפשר להשתמש בקובץ גדול יותר עם יותר מילים וכך יחזרו מילים דומות יותר למילה שאנחנו מחפשים.
- הוספת פונקציות דירוג נוספות – למשל CosSim , ניסינו לממש אך ללא הצלחה רבה במבחן התוצאה.