מנוע אחזור – חלק ב'

a. הסבר מפורט על אופן פעולת המנוע –

המשך מסדר הפעולות של חלק א':

בסוף חלק א' נבנה מילון, נבנו כל קבצי הpostingמחולקים ל27 תיקיות, נבנו ונשמרו מבני נתונים עבור פרטי המסמכים והישויות בתוכם.
כעת הבסיס מוכן על מנת לאפשר אחזור מידע על בסיס שאילתות/שאילתה של המשתמש.

פירוט תהליך הרצת השאילתה-

התחלת סדר האירועים לאחזור מסמכים עבור שאילתה מתחיל בהכנסת שאילתה לשדה הQuery או נתיב לקובץ שאילתות לשדה הQueries Path בחלון הראשי ולחיצה על Run\RunQueries.

כאשר מריצים שאילתה הGuiController בודק האם המילון ורשימת הstopwords טעונים לזיכרון ואם כן מעביר את הבקשה עם הנתונים מהמשתמש אל הMyViewModel. הMyViewModel מעביר את הקריאה אל MyModel.

עיבוד ראשוני של השאילתות

שאילתה בודדת תרוץ בMyModel בפונקציה runQuery אשר תקבל את השאילתה (ינתן לה מספר זיהוי רנדומלי בן שלוש ספרות).

קובץ של שאילתות יעובד בפונקציה runQueries – הנתיב יועבר למחלקת readFile על מנת לחלק את כל השאילתות לשאילתות נפרדות – כאשר נייחס רק לשדות הבאים:

1. מספר השאילתה.
2. כותרת השאילתה.
3. תיאור השאילתה.

מחלקת readFile מפרידה את קובץ הטקסט של השאילתות ויוצרת עבור כל שאילתה אובייקט מסוג Query (מחלקה המייצגת query יחידה) המאותחל עם השדות שתוארו לעיל.

במידה והוחלט, על ידי המשתמש, להשתמש בניתוח סמנטי על השאילתה יתבצע תהליך של הרחבת השאילתה במחלקת SemanticAnalyzer. הסבר על תהליך ה expansion יפורט בהמשך.

אחזור מסמכים

כל שאילתה, בין אם בודדת ובין אם כחלק מקבוצת שאילתות ממסמך, תטופל באופן זהה.

חלקי השאילתה ישלחו למחלקת Searcher אשר תחזיר רשימה מדורגת של המסמכים הרלוונטים לשאילתה זו. כותרת השאילתה, תיאורה (במידה וזו שאילתה בודדת תיאורה יהיה ריק), האם להשתמש ב stemming והאם להשתמש בניתוח סמנטי יועברו לSearcher לטובת פעולת האחזור.

במחלקה Searcher הפונקציה runQuery מריצה חיפוש עבור כל שאילתה באופן הבא:

עיבוד חלקי השאילתה

על מנת לאחזר מסמכים בצורה נכונה המתאימה למסמכים המאונדקסים נבצע פעולת פירסור זהה לשאילתה שהתקבלה באמצעות השיטה parseQuery. ב parseQuery נפרסר בזכות ה parse , שכתבנו בחלק א של הפרויקט, את הכותרת של השאילתה ואת תיאור השאילתה בנפרד. בשלב זה יהיה התייחסות לבחירת האפשרות לשימוש בPorter Stemmer.

בסוף עיבוד חלקי השאילתה נקבל זוגות של הterms שזוהו בחלקים והfrequency של כל term בכל חלק.

איסוף המידע על המסמכים הרלוונטיים

נאחד את כל הterms שהופיעו בחלקי השאילתה ובאמצעות רשימה זו נייצר מבנה נתונים המכיל את המידע הדרוש על המסמכים הרלוונטים.

מסמך רלוונטי שעבורו נאסוף מידע הוא מסמך המכיל לפחות מילה אחת מהכותרת או מהתיאור של השאילתה.

מבנה הנתונים שנייצר הוא HashMap<String,Pair<Integer , HashMap<String , Integer>>>

```
HashMap< DocID → Pair(Document length , HasMap< Term → Document frequency>)>
```

המפתח של המפה החיצונית הוא מזהה של מסמך, בכך נוכל להגיע לנתונים שלו במהירות O(1).
הנתונים הנשמרים על המסמך מוכלים בPair שמחזיק את אורך המסמך ואת כל הterms והtf שלהם במסמך.

הפונקציה שאחראית על בניית מבנה נתונים זה היא getAllDocumentsRelevantDetails.

ייבוא מידע מהאינקס ההופכי

עבור כל הterm מהשאילתה נייבא את השורה הרלוונטית מהאינדקס ההופכי בקובץ הposting בעזרת הIndexer. הפונקציה getTermPosting אחראית על ייבוא שורה מהאינדקס והיא פונה לdocumenter עבור הגישה לקבצים בדיסק. במידה והמילה שנחפש קיימת במילון יוחזר ArrayList של Pair המייצגים את כל הזוגות של הdocID וה tf שלהם במסמך. קובץ הposting המתאים עבור המילה מאותר ע"י חיפוש במילון. כאשר השורה המתאימה בקובץ זה נמצאה מתבצע פירסור של השורה וייצור הרשימה של זוגות המסמך-תדירות שתוחזר.

את הנתונים עבור כל **מילה-מסמכים** מהאינדקס ההופכי נכניס למפה שתוארה לעיל ובכך ניצור אינדקס חלקי ישיר של קשר **מסמך-מילים**.

חשוב לשים לב: מבנה הנתונים המתואר לעיל אינו מכיל את כל המילים במסמך, אלא רק את המילים המשותפות למסמך ולשאילתה.

דירוג המסמכים הרלוונטיים אל מול השאילתה

כל מסמך יישלח אל המחלקה Ranker על מנת לקבוע את מידת הרלוונטיות שלו לשאילתה. חישוב הדירוג של המסמך מתבצע בעזרת מספר שיטות שונות על מנת לנסות ולמקסם את תוצאות האחזור. כל אחד מהמדדים המחושבים לדמיון ורלוונטיות של המסמך לשאילתה **מוכפל במשקולת** אשר ערכה נקבע בתהליך אופטימיזציה על מנת למקסם את הPrecision והRecall של התוצאות. הסבר על תהליך בחירת המשקולות יפורט בהמשך.

איסוף והחזרת המסמכים לפי דירוגם

המסמכים שדורגו מוכנסים לפי דירוגם לתור עדיפויות – כך שדירוג גבוה יותר הוא בעדיפות גבוהה יותר. את 50 המסמכים הראשונים (שדירוגם הגבוהה ביותר) נחזיר כתוצאות האחזור לMyModel. תוצאות האחזור ישמרו בשדה מיועד לשמירת תוצאות האחזור האחרונות. שדה זה ישמש לשמירת התוצאות לקובץ במידת הצורך. התוצאות יוצגו על ידי המחלקה RetrievalResultView שאחראית על סידור והצגת התוצאות בממשק המשתמש.

שינוי שנעשה מחלק א:

הוספנו מבנה נתונים במחלקת Parse של hashMap, עבור כל מסמך, עם pair של string ו Integer , כאשר ה string זה הישות והInteger זה כמות המופעים של הישות באותו מסמך – נועד על מנת להחזיר את הישויות עבור כל מסמך – הישויות הנ"ל ישמרו שם אך ורק אם הן הופיעו במסמך נוסף – מתעדכן בזמן האיחוד של המילון וקבצי הposting ומול מבנה הנתונים מחלק א של הפרויקט הבודק לנו זאת.

b. המחלקות הנוספות של חלק ב':

- View Package:
  - RetrievalResultView: המחלקה שאחראית על הצגת תוצאות השאילתות בצורה של טבלה.
    שיטות המחלקה:

```
/**
 * Displays a window that contains the retrieval results
 *
 * @param title           - String
 * @param rankedDocuments - ArrayList<Pair<String, ArrayList<String>>>
 * @param viewModel       - MyViewModel
 */
public static void display(String title, ArrayList<Pair<String, ArrayList<String>>>
rankedDocuments, MyViewModel viewModel)
```

  - RetrievalResultController: המחלקה שמחברת בין הGUI לRetrievalResultView .
    שיטות המחלקה:

```
/**
 * Handles the Display entities button being pressed
 *
 * @param actionEvent - ActionEvent - the event of the button press
```

```
     */
    public void displayEntities(ActionEvent actionEvent)
    /**
     * Handles the save Results button being pressed
     *
     * @param actionEvent
     */
    public void saveResults(ActionEvent actionEvent)
    /**
     * Displays a file selection window and returns the absolute path of the file chosen.
     *
     * @param event - ActionEvent - the button that was pressed
     * @return - String - the absolute path of the file chosen or an empty String
     */
    private String saveFileChooser(ActionEvent event)
```

- Model Package:
  - MyModel: מחלקה מחלק א, האחראית על כל החיבור בין המימוש הלוגי לממשק הגרפי.
    הפונקציות שנוספו –

```
    /**
     * Initializes all the needed components before a query is run.
     * Makes sure that the searcher and the parse fields have been initialized an if not
    initializes them.
     */
    private void preQueryInitialization()
    /**
     * Returns the relevant documents to a given query sorted by their relevance.
     *
     * @param query               - String - a phrase to retrieve relevant documents for.
     * @param useSemanticAnalysis - boolean - whether to use semantic analysis on the given
    query.
     * @return - ArrayList<Pair<String, ArrayList<String>>> - a list containing a pair of
    the query as key and the retrieved documents for it as a list.
     */
    ArrayList<Pair<String, ArrayList<String>>> runQuery(String query, boolean
    useSemanticAnalysis);
    /**
     * Returns the relevant documents to the queries in the given path, sorted by their
    relevance.
     *
     * @param queriesPath         - String - a path to the queries file.
     * @param useSemanticAnalysis - boolean - whether to use semantic analysis on the given
    query.
     * @return - ArrayList<Pair<String, ArrayList<String>>> - a list containing pairs of
    each query number as key and the retrieved documents for it as a list.
     */
    ArrayList<Pair<String, ArrayList<String>>> runQueries(String queriesPath, boolean
    useSemanticAnalysis)
    /**
     * Checks if a given String is a valid document number by comparing it with the pre-
    loaded documents details.
     *
     * @param documentNumber - String - a string to check.
     * @return - boolean - true if a given String is a valid document number, else false.
     */
    boolean checkValidDocumentNumber(String documentNumber)
    /**
     * Returns an sorted array of the entities, based on importance, in the document.
```

```
     *
     * @param documentNumber - String - a valid document number.
     * @return - String[] - an sorted array of the entities, based on importance.
     */
    ArrayList<Pair<String, Double>> getDocumentEntities(String documentNumber)
    /**
     * Saves the latest retrieval results to the given path.
     * If a file already exists in the destination path then overrides its.
     *
     * @param path - String - a path to save the latest retrieval results to.
     */
    void saveLatestRetrievalResults(String path)
    /**
     * Sets the latestQueryResult field
     *
     * @param rankedDocuments
     */
    private void setLatestQueryResult(ArrayList<Pair<String, ArrayList<String>>>
    rankedDocuments)
```

- Retrieval Package:
  - ממשק IRanker:

```
/**
 * Ranks a document compared to a query based on the given parts of both.
 *
 * @param processedQuery            - ArrayList<TermDocumentTrio> - The query after
parsing
 * @param processedQueryDescription - ArrayList<TermDocumentTrio> - The query description
after parsing.
 * @param processedExpandedQuery    - ArrayList<TermDocumentTrio> - The expanded query
after parsing.
 * @param semanticExpandedTerms     - HashMap<String, Integer> - All the terms and their
frequencies from the expanded query.
 * @param documentLength            - int - The length of the document.
 * @param documentTerms             - HashMap<String, Integer> - All the terms and their
frequencies from the document.
 * @param documentHeader            - ArrayList<TermDocumentTrio> - The document header
after parsing.
 * @param documentEntities          - ArrayList<String> - The document entities.
 * @return double - The rank of the similarity between the query and the document.
 */
double rankDocument(ArrayList<TermDocumentTrio> processedQuery,
ArrayList<TermDocumentTrio> processedQueryDescription, ArrayList<TermDocumentTrio>
processedExpandedQuery, HashMap<String, Integer> semanticExpandedTerms, int
documentLength, HashMap<String, Integer> documentTerms, ArrayList<TermDocumentTrio>
documentHeader, ArrayList<String> documentEntities)
/**
 * Returns an sorted array of the entities, based on importance.
 *
 * @param documentEntities        - HashMap<String, Integer> - A Map of all the entities
in a document.
 * @param processedDocumentHeader - ArrayList<TermDocumentTrio> - The document header
after parsing.
```

```
 * @return - ArrayList<Pair<String, Double>> - an sorted list of the entities, based on
importance.
 */
ArrayList<Pair<String, Double>> rankEntities(HashMap<String, Integer> documentEntities,
ArrayList<TermDocumentTrio> processedDocumentHeader)
```

- Ranker: מממשת את הממשק IRanker , מחלקה שאחראית על כל החישובים של דירוג המסמכים, כאשר היא מכילה כמה פונקציות חישוב לפי מדדים שונים – BM25 , מדד Jaccard , מדד דמיון של Entity. – ואת המשקל שנרצה לתת לכל אחד מהמדדים.

המשקלים והנתונים :

```
private static double b = 0.7;
private static double k = 1.81;

private static double WEIGHT_QUERY_BM25 = 1;
private static double WEIGHT_QUERYDESC_BM25 = 1.5;
private static double WEIGHT_HEADER = 0.05;
private static double WEIGHT_ENTITIES = 0.2;
```

פונקציות המחלקה:

- ```
  /**
   * Constructor
   *
   * @param corpusSize - int - The corpus size
   * @param avdl       - double - average document length
   * @param indexer    - Indexer - indexer
   */
  public Ranker(int corpusSize, double avdl, Indexer indexer)
  ```
- ```
  // rank calculators -
  // 1. BM25
  // 2. Jaccard Similarity between the query and documentHeader.
  // 3. DSC (Dice) between the query and the document entities.
  @Override
  public double rankDocument(ArrayList<TermDocumentTrio> query,
  ArrayList<TermDocumentTrio> queryDescription, ArrayList<TermDocumentTrio>
  expandedQuery, HashMap<String, Integer> semanticExpandedTerms, int documentLength,
  HashMap<String, Integer> documentTerms, ArrayList<TermDocumentTrio> documentHeader,
  ArrayList<String> documentEntities)
  ```
- ```
  /**
   * Computes BM25 between a given query and a given document.
   * https://en.wikipedia.org/wiki/Okapi_BM25
   *
   * @param query          - ArrayList<TermDocumentTrio> - The query after parsing
   * @param documentLength - int - The length of the document.
   * @param documentTerms  - HashMap<String, Integer> - All the terms and their
  frequencies from the document.
   * @return - double - BM25 rank between the given query and the given document.
   */

  public double BM25calculator(ArrayList<TermDocumentTrio> query, int documentLength,
  HashMap<String, Integer> documentTerms)
  ```
- ```
  /**
   * Computes dice coefficient
   * https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient
   *
   * @param queryTerms       - List<String> - A list of the terms in the query.
   * @param documentEntities - ArrayList<String> - A list of the document entities.
   * @return - double - DSC rank between the given queryTerms and the given
  ```

```
documentEntities.
 */
private double DSCCalculator(List<String> queryTerms, ArrayList<String>
documentEntities)
/**
 * Computes Jaccard similarity.
 * https://en.wikipedia.org/wiki/Jaccard_index
 *
 * @param queryTerms     - List<String> - A list of the terms in the query.
 * @param documentHeader - List<String> - A list of the terms in the document header.
 * @return - double - Jaccard similarity rank between the given queryTerms and the
given documentHeader.
 */
private double JaccardCalculator(List<String> queryTerms, List<String> documentHeader)
/**
 * Receives two hash maps and merges them.
 *
 * @param semanticExpandedTerms - HashMap<String, Integer> - A map to merge.
 * @param documentTerms         - HashMap<String, Integer> - A map to merge.
 * @return - HashMap<String, Integer> - A merged map.
 */
private HashMap<String, Integer>
mergeQueryAndExpandedQueryDocumentsTerms(HashMap<String, Integer>
semanticExpandedTerms, HashMap<String, Integer> documentTerms)
/**
 * Receives two array lists and merges them.
 *
 * @param query         - ArrayList<TermDocumentTrio> - A list to merge.
 * @param expandedQuery - ArrayList<TermDocumentTrio> - A list to merge.
 * @return - ArrayList<TermDocumentTrio> - A merged list.
 */
private ArrayList<TermDocumentTrio>
mergeQueryAndExpandedQueryTrios(ArrayList<TermDocumentTrio> query,
ArrayList<TermDocumentTrio> expandedQuery)
/**
 * return list of terms from ArrayList<TermDocumentTrio> documentTrios
 *
 * @param documentTrios - ArrayList<TermDocumentTrio> - A list to extract terms from.
 * @return - List<String> - All the terms in the given list.
 */
private List<String> extractTerms(ArrayList<TermDocumentTrio> documentTrios)
/**
 * Ranks a single entity based on its Frequency in the document and in the document
header.
 *
 * @param docFrequency       - int - The number of times that the entity appears in the
documents.
 * @param docHeaderFrequency - int - The number of times that the entity appears in the
document header.
 * @return
 */
private double rankEntity(int docFrequency, int docHeaderFrequency)
```

- Searcher: המחלקה אחראית על אחזור מסמכים רלוונטיים עבור שאילתה מסוימת בהתבסס על מאגר מסמכים (אשר אונדקס בעבר) תוך דירוג המסמכים על פי הרלוונטיות שלהם באמצעות שימוש בממשק IRanker.

```
/**
 * Total number of results per query.
```

```
 */
private static final int RESULTNUMBER = 50;
```

פונקציות במחלקה:

- ```
  /**
   * Constructor
   *
   * @param indexer    - Indexer - the indexer that will be used for the retrieval process.
   * @param corpusSize - int - The number of documents in the entire corpus
   * @param avdl       - double - The average document length
   */
  public Searcher(Indexer indexer, int corpusSize, double avdl)
  ```

- ```
  /**
   * Returns the relevant documents to a given query sorted by their relevance.
   *
   * @param query                 - String - A phrase to retrieve relevant documents for.
   * @param queryDescription      - String - The query description as extracted from a file. if a single query was run then an empty String.
   * @param semanticExpandedTerms - String - The query after semantic analysis and expansion.
   * @param indexer               - Indexer - The indexer that will be used for the retrieval.
   * @param parser                - Parse - A parser for the query sections.
   * @return - ArrayList<String> - The retrieved documents for it as a list.
   */
  public ArrayList<String> runQuery(String query, String queryDescription, String semanticExpandedTerms, Indexer indexer, Parse parser)
  ```
- ```
  /**
   * Returns a map of all the documents which contains both the original query terms and terms from the semantic expansion.
   *
   * @param semanticExpandedTerms    - String - The query after semantic analysis and expansion.
   * @param processedExpandedQuery   - ArrayList<TermDocumentTrio> - The expanded query after parsing.
   * @param indexer                  - indexer - The indexer that will be used for the retrieval.
   * @param relevantDocumentsDetails - HashMap<String, Pair<Integer, HashMap<String, Integer>>> - The relevant documents details
   * @return - HashMap<String, Pair<Integer, HashMap<String, Integer>>> - A Map of all the documents which contains both the original query terms and terms from the semantic expansion.
   */
  private HashMap<String, Pair<Integer, HashMap<String, Integer>>> getRelevantDocumentsForExpandedQuery(String semanticExpandedTerms, ArrayList<TermDocumentTrio> processedExpandedQuery, Indexer indexer, HashMap<String, Pair<Integer, HashMap<String, Integer>>> relevantDocumentsDetails)
  ```
- ```
  /**
   * Merges all the given Lists to create an aggregated HashSet
   * of all the terms in them.
   *
   * @param secondList - ArrayList<TermDocumentTrio> - a list of TermDocumentTrio to merge all the terms in it.
   * @param thirdList  - ArrayList<TermDocumentTrio> - a list of TermDocumentTrio to merge all the terms in it.
   * @return - HashSet<String> - all the terms in the given Lists.
  ```

```
 */
private HashSet<String> mergeLists(ArrayList<TermDocumentTrio> secondList,
ArrayList<TermDocumentTrio> thirdList)
```

- ```
  /**
   * Parses an processes a single given query using the given parser.
   *
   * @param elementID - String - The query number.
   * @param query     - String - A phrase to parse.
   * @param parse     - Parse - the parser that will be used.
   * @return - ArrayList<TermDocumentTrio> - the processed query.
   */
  private ArrayList<TermDocumentTrio> parseQuery(String elementID, String query, Parse
  parse)
  ```
- ```
  /**
   * Gathers and returns all the documents details for the documents which contains terms
  from the given processedQuery.
   *
   * @param processedQuery - HashSet<String> - The processed query.
   * @param indexer        - indexer - The indexer that will be used for the retrieval of
  the documents details.
   * @return - HashMap<String, Pair<Integer, HashMap<String, Integer>>> - DocID -->
  Pair(Document length , HasMap( Term , Document frequency))
   */
  private HashMap<String, Pair<Integer, HashMap<String, Integer>>>
  getRelevantDocumentsDetails(HashSet<String> processedQuery, Indexer indexer)
  ```
- ```
  /**
   * Returns an sorted array of the entities, based on importance.
   *
   * @param documentEntities - HashMap<String, Integer> - A Map of all the entities in a
  document.
   * @param documentHeader   - String - The document header.
   * @return - ArrayList<Pair<String, Double>> - an sorted list of the entities, based on
  importance.
   */
  public ArrayList<Pair<String, Double>> rankEntities(HashMap<String, Integer>
  documentEntities, String documentHeader, Parse parser)
  ```

- Query: מחלקה שמייצגת שאילתה , כאשר היא מכילה את מספר הזיהוי של השאילתה, כותרת השאילתה, תיאור השאילתה.
  פונקציות המחלקה:

- ```
  /**
   * Constructor
   *
   * @param number      - int - query number.
   * @param title       - String - The query title, the main part of the query.
   * @param description - String - The query description.
   */
  public Query(int number, String title, String description)
  ```
  בנוסף גם Setter ו Getter עבור כל שדה.
- SemanticAnalyzer: מחלקה אשר אחראית על הסמנטיקה – כאשר משתמש רוצה להשתמש בסמנטיקה, הSearcher יפנה למחלקה הנ"ל על מנת לקבל את המילים, מהמילון של הסמנטיקה, בהתאם לכל term בשאילתה. – מחלקה שהיא Singleton

```
private static final int EXPENDEDWORDSPERTERM = 3;
```

שיטות המחלקה:

- ```
  /**
   * Private constructor.
   */
  private SemanticAnalyzer()
  ```

- ```
  /**
   * Returns a SemanticAnalyzer
   *
   * @return - SemanticAnalyzer - this.semanticAnalyzer
   */
  public static SemanticAnalyzer getInstance()
  ```
- ```
  /**
   * Receives a term and expands it using semantic analysis of the terms within it.
   * for each term in the query finds a list of words with close semantic meaning.
   *
   * @param queryToExpand - String - query to expand
   * @return - String - expanded query
   */
  public String expandQuery(String queryToExpand)
  ```
- ```
  /**
   * Receives a term and finds other words with close semantic meaning.
   *
   * @param queryTerm - String - a word to expand
   * @return - ArrayList<String> - words with close semantic meaning to queryTerm
   */
  private ArrayList<String> expandTerm(String queryTerm)
  ```

c. <u>מדדי דירוג המסמכים</u>

- BM25 - פונקציית דמיון בין השאילתה למסמך. מטרתה לתת דירוג למסמך לפי רמת הדמיון שלו לשאילתה בהתבסס על תדירות המילים במסמך, בשאילתה עצמה ובכלל הקורפוס (שימוש בDocumentTF, בQueryTF ובDF עבור term מסוים). בנוסף, מדד זה "מעניש" מסמכים ארוכים, כלומר, הציון של מסמכים הארוכים מהממוצע ירד והציון של מסמכים הקצרים מהמוצע יעלה (מתבסס על ההנחה שמסמכים קצרים יותר אינפורמטיביים יותר, בדומה להתייחסות למילים נדירות במילון). **באלגוריתם הדירוג שלנו** משומש לאמידת הדמיון בין השאילה למסמך ובין תיאור השאילתה למסמך. במידה והרחבת השאילתה על ידי ניתוח סמנטי נדרש, החישוב מתבצע על השאילתה המורחבת.

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

- Jaccard Similarity - מחשב דמיון בין שתי קבוצות. **באלגוריתם הדירוג שלנו** משמש לאמוד את הדמיון בין השאילתה לבין הheader של המסמך.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- DSC (Dice) coefficient – מדד נוסף לדמיון בין שתי קבוצות. **באלגוריתם הדירוג שלנו** משמש לאמוד את הדמיון בין השאילתה לבין הישויות שזוהו במסמך.

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}$$

ניתוח סמנטי

אנו משתמשים במודל Word2Vec מאומן על מנת לבצע הרחבה לשאילתה. השאילתה המקורית נשלחת למחלקה SemnticAnalyzer בה עבור כל מילה מוחזרות 3 מילים אשר קרובות אליה מבחינה סמנטית. שימוש זה מאפשר לנו לדרג מסמכים אל מול שאילתה מורחבת זו. בכך, מסמכים אשר מכילים גם את השאילתה המקורית וגם מילים מהשאילתה המורחבת ידורגו גבוה יותר. שימוש זה מתבסס על ההבנה כי מסמכים רלוונטיים יותר יכילו גם מילים הקרובים סמנטית למילים בשאילתה.

בחירת משקולות

בחירת המשקולות התבצעה לאחר הרצה של פונקציית אופטימיזציה. פונקציה זו השימה ערכים שונים למשקולות והחזירה את סט המשקולות עבורן המדדים של הRecall והPrecision היו מקסימליים.

אלגוריתם למציאת ודירוג ישויות במסמך

כאשר מוכנס שם של מסמך עבורו נדרש למצוא את הישויות הרלוונטיות מתבצע חיפוש של שם המסמך במבנה נתונים המוחזק בIndexer הראשי של הMymodel המחזיק את רשימת הישויות שזוהו במסמך זה. רשימה זו מועברת לRanker על מנת לדרג את הישויות לפי חשיבותן במסמך.
הדירוג מתבצע לכל ישות כתלות במספר הפעמים שהישות חוזרת במסמך ובמספר הפעמים שהישות חוזרת בכותרת המסמך. המשקל למספר החזרות של הישות בכותרת גבוה משמעותית מאשר המשקל למספר החזרות של הישות בגופו.

דוגמות:

1. בהרצות של האופטימיזציה ראינו שכאשר הרצנו ללא stemming וללא semantic:

```
private static final double b = 0.5;
private static final double k = 1.5;

private static double WEIGHT_QUERY_BM25 = 0.5;
private static double WEIGHT_QUERYDESC_BM25 = 0.5;
private static double WEIGHT_HEADER = 0.2;
private static double WEIGHT_ENTITIES = 0.2;
```

קיבלנו 188 במקום הנתונים הקודמים (רשומים למעלה) שקיבלנו 200 – עבור השאילתה 351 החזרנו מסמך אחד פחות (18 במקום 19) מהנתונים שבחרנו.

2. בהרצות של האופטימיזציה ראינו שכאשר הרצנו ללא stemming וללא semantic:

```
private static final double b = 0.5;
private static final double k = 1.2;

private static double WEIGHT_QUERY_BM25 = 0.1;
private static double WEIGHT_QUERYDESC_BM25 = 0.1;
private static double WEIGHT_HEADER = 0.2;
private static double WEIGHT_ENTITIES = 0.2;
```

קיבלנו 83 במקום הנתונים הקודמים (רשומים למעלה) שקיבלנו 200 – עבור שאילתה 351 החזרנו רק 5 מסמכים רלוונטיים. (-בניגוד לנתונים שנבחרו שהחזרנו 19)

d. קבצי הpostingשלנו מחולקים לפי אותיות, כאשר כל term נמצא בקובץ posting בו האות התחלתית שלו היא בתקיה בה שמה הוא האות הנ"ל ובתוכה קובץ ה posting .
גם במילון נשמרו כל הנתונים הנ"ל – ה term ובאיזה נתיב הקובץ posting נמצא.
כאשר נקבל את השאילתה, אנו מפרסרים אותה (בדיוק באותו פרסר של פירסור המסמכים).
נעבור עבור כל term בשאילתה ונבדוק אם הוא קיים במילון – אם לא קיים נבדוק אם קיים באות קטנה-במידה וקיים נעבור לקובץ posting שהמילון מפנה אותנו ונייבא משם את כל זוגות המסמכים הרלוונטיים.

e. שימוש בקוד פתוח:
בחלק זה של הפרויקט השתמשנו בקוד פתוח לטיפול הסמנטי-
https://github.com/medallia/Word2VecJava/blob/master/src/main/java/com/medallia/word2vec/Word2VecModel.java
ע"י ייבוא של jar לפרויקט. כאשר המשתמש לוחץ על טיפול סמנטי, פתחנו מחלקה של semantic Analyzer , ואנו נשתמש בה במקרה זה – כאשר מחלקה משתמשת בקוד הפתוח ומחזירה את כל ה Terms הרלוונטיים עבור כל term של השאילתה הנבדקת. – נתנו ערך של 3 איטרציות.

חלק ב' – הערכה על המנוע –
Unstemming ללא Semantic:

מספר שאילתה	מילות השאילתה	Precision	Recall	Precision5	Precision15	Precision30	Precision50	זמן ריצה
351	Falkland petroleum exploration	0.4	0.4167	0	0.2	0.3333	0.3958	6.749
352	British Chunnel impact	0.7	0.1423	0.6	0.8	0.8	0.1423	8.479
358	blood-alcohol fatalities	0.44	0.4314	0.2	0.4667	0.4	0.4314	6.495
359	mutual fund predictors	0.06	0.1071	0	0.0667	0.0667	0.1071	3.931
362	human smuggling	0.18	0.2308	0.2	0.2	0.2	0.1795	1.376
367	piracy	0.32	0.0865	0.4	0.3333	0.3	0.0865	5.33

6.319	0.25	0.2333	0.2667	0.2	0.4375	0.14	encryption equipment export	373
4.447	0.0196	0.0333	0.0667	0	0.0196	0.08	Nobel prize winners	374
1.782	0.2222	0.2333	0.0667	0	0.3056	0.22	cigar smoking	377
2.101	0.4129	0.1333	0.2667	0	0.7143	0.1	obesity medical treatment	380
4.64	0.2549	0.3	0.2667	0.4	0.2549	0.26	space station moon	384
5.679	0.2	0.2	0.0667	0	0.2	0.34	hybrid fuel cars	385
7.866	0.1918	0.3333	0.2	0.2	0.1918	0.28	radioactive waste	387
2.41	0.22	0.2667	0.2667	0.4	0.22	0.22	organic soil enhancement	388
10.806	0.1066	0.3333	0.4	0.4	0.1066	0.26	orphan drugs	390

MAP - Average precision over all rel docs 0.0788

Stemming ללא Semantic:

זמן ריצה	Precision50	Precision30	Precision15	Precision5	Recall	Precision	מילות השאילתה	מספר שאילתה
6.796	0.3125	0.2333	0.0667	0	0.3542	0.34	Falkland petroleum exploration	351
8.594	0.1341	0.7333	0.8	0.8	0.1341	0.66	British Chunnel impact	352

8.275	0.4118	0.4	0.3333	0.2	0.4118	0.42	blood-alcohol fatalities	358
6.782	0.1429	0.1667	0.2	0	0.2143	0.12	mutual fund predictors	359
1.903	0.1026	0.1333	0.1333	0	0.1282	0.1	human smuggling	362
8.707	0.0811	0.2667	0.2	0.6	0.0973	0.36	piracy	367
8.352	0.25	0.2333	0.2	0.4	0.4375	0.14	encryption equipment export	373
6.917	0.0196	0.033	0.0667	0.2	0.0196	0.08	Nobel prize winners	374
4.204	0.2778	0.2667	0.2	0	0.3889	0.28	cigar smoking	377
3.3	0.1429	0.1333	0.2	0	0.5714	0.08	obesity medical treatment	380
7.294	0.2157	0.2	0.2667	0.4	0.2157	0.22	space station moon	384
8.128	0.2235	0.2	0	0	0.2235	0.38	hybrid fuel cars	385
11.344	0.1644	0.2333	0.1333	0.2	0. 1644	0.24	radioactive waste	387
5.497	0.4	0.5	0.4667	0.2	0.4	0.4	organic soil enhancement	388
14.242	0.1393	0.4333	0.3333	0.4	0.1393	0.34	orphan drugs	390

MAP - Average precision over all rel docs 0.0792

בשאילתה מספר 352- השאילתה הינה באנגלית בריטית, כאשר המילון שלנו הוא באנגלית אמריקאית. לכן הוספנו אופציה לשינוי המילה "Chunnel" להיות tunnel וגם channel – הדבר הזה מראה בעיקר איך שימוש בטיפול סמנטי נכון יכול להביא אחזור של מסמכים בצורה משמעותית הרבה יותר טובה.

(כאשר לא היה את ההמרה הנ"ל ההחזרה שלנו עבור אותה שאילתה הייתה אפס).

<u>טיפול סמנטי:</u>

בטיפול סמנטי החזרת המסמכים ב Unstemmed עם Semantic העלתה לנו את הכמות מ200 ל 207, הסיבה לכך היא מכיוון שהשימוש הסמנטי עוזר לנו למצוא מילים הדומות למילות השאילתה וכך לדרג את המסמכים בצורה טובה יותר.

כנ"ל לגביי stemming , ההחזרה היא 225 במקום 208 ללא semantic.

3) <u>סיכום</u>

התמודדנו עם בעיות רבות במהלך הפרויקט כאשר חלקן תכנותיות וחלקן קשורות לחומר התיאורטי.

במהלך פיתוח המנוע, כמו בכל פיתוח מערכת בסדר גודל כזה, התגלו באגים וקשיי מימוש. המכשול הגדול ביותר שהיינו צריכים לעבור בכל הפרויקט היה כתיבת מחלקת הParse אשר אחראית לפירסור המסמכים. במהלך ביצוע המטלה הרגשנו פעמים רבות כי איננו לומדים כלום מכתיבתה, אלא רק כותבים קוד פשוט שיכול היה להיכתב בכל קורס מבוא לתכנות. כתיבת מחלקה זו, לדעתנו, תרמה מעט מאוד לידע שלנו לגבי אחזור מידע. התמודדנו עם בעיות התכנות בעזרת ניסיון העבר שלנו והרבה מאוד עבודה קשה.

קושי משמעותי נוסף שנתקלנו בו הוא בהבנת המבנה הנדרש מהמערכת והזרימה של התוכנית והשלבים בה. נאבקנו מאוד להבין איך הארכיטקטורה של התוכנה צריכה להיראות והיה לנו קשה מאוד להבין את היתרונות והחסרונות של כל בחירה. עקב העובדה שהמערכת גדולה, בחירה תכנונית לא מתאימה תגרור שינויים גדולים בהמשך הדרך. נקלענו למצב זה מספר פעמים במהלך הפיתוח. מצב בו אנו נדרשים לשנות חלקים מאוד מאוד משמעותיים בקוד מכיוון שלא עמדו בדרישות הפרויקט (מבחינת זמני ריצה, מבני נתונים וכו'..). אנו הרגשנו שה"חופש" שניתן במימוש המנוע לא עזר לנו להגיש מוצר טוב יותר, אלא מנע מאיתנו להבין וללמוד **דרכים נכונות** לטיפול ועיבוד מסמכים. בעוד שדרכי עיבוד לומדו ברמה גבוהה בהרצאות, **לא הוצגו מימושים כלל** ולכן נאלצנו להתמודד עם קשיים אלו לבד.

התמודדנו עם קושי זה ע"י תכנון מסודר וברור לפני כל שינוי או מימוש, אך שלב זה היה קשה מאוד עקב חוסר ההבנה שלנו במערכות מסוג זה וחוסר התמיכה שחווינו מצד הצוות לטובת עיצוב ותכנון המערכת שלנו.

אחת הבעיות המרכזיות היא שלא קיבלנו פידבק על חלק א' של הפרויקט. פרויקט זה מאוד גדול וכאשר חלק ב' תלוי באופן ישיר על מימוש חלק א', חוסר במשוב מינימלי בקשה עלינו מאוד. גם עד לרגע ההגשה איננו יודעים האם הארכיטקטורה והמבנה של מנוע האחזור שלנו נכון ופועל כשורה.

הצעות לשיפור:

מגבלת הזמן עקב העומס הגדול וההתעסקות בפתרון בעיות מימוש מנעו מאתנו לממש מספר אלמנטים שלדעתנו היו משפרים את תוצאות האחזור. הנקודה המרכזית אותה היינו משפרים היא דירוג המסמכים אל מול השאילתה. בעוד שהדירוג איננו גרוע כלל, יש מקום לשיפור. שקלנו ליצור מטריצת Tf-idf לטובת שימוש בדמיון cosSim אך לא הספקנו לממש זאת. בנוסף, היינו שמחים להעמיק יותר בשימוש בניתוח סמנטי של השאילתה כדי להשתמש בכלים נוספים שעולם זה מציע.

זאת ועוד, הדרישה בחלק א של הפרויקט, לעמוד בזמני ריצה לאינדוקס המאגר מנעה מאיתנו לייצר אינדקס איכותי יותר. ללא מגבלה זו (או הקלה בה) יכולנו לשמור מידע נוסף על המסמכים. התוספת המשמעותית ששקלנו לממש הייתה positional indexing , דבר אשר היה יכול לעזור לנו מאוד באחזור המסמכים. בנוסף גם כתיבת פרסר איכותי יותר.