**דו"ח מנוע אחזור – חלק ב'**

Retrieve Documents For Query

Documents Summary

מחלקות נוספות שהוספנו לטובת דירוג השאילתות:

**מחלקת Searcher:**

מחלקה זו מקבלת שאילתה ומבצעת חיפוש של מסמכים פוטנציאלים עבור השאילתה שאותם היא שולחת לדירוג על ידי הRanker.

קלט: השאילתה שעבורה יש לאחזר את המסמכים.

פלט: רשימת 50 המסמכים המדורגים הכי גבוה.

משתני מחלקה מרכזיים:

* משתנה\_ controller המהווה את עמוד התווך של המנוע ומחזיק את כל המידע של המנוע.

שיטות:

שיטות הקשורות Documents Reteival Functionality :

1. public Dictionary<Document, double> Search(string query)

שיטה זו מקבלת שאילתא ומחזירה עבורה את 50 המסמכים הרלוונטים ביותר עבורה. השיטה נעזרת במציאת המסמכים הפוטנציאליים ע"י שיטות 2,3 ומחשבת את דירוג המסמכים ע"י מחלקת Ranker שתפורט בהמשך. השיטה מחזירה מילון של מסמכים ועבור כל מסמך את הדירוג שלו.

1. private Dictionary<string, TermPostingList> GetQueryPostingList(List<string> termsInQuery)

שיטה זו מקבלת רשימה של terms שנמצאים בשאילתא ומחזירה עבורם את רשימות הפוסטינג שלהן, נעזר בשיטה 4. השיטה מחזירה מילון של מילה עם רשימת הפוסטינג שלה.

1. private Dictionary<int, Document> GetPotentialDocs(Dictionary<string, TermPostingList> queryPostingList)

שיטה זו מקבלת מילון של term והפוסטינג, השיטה עוברת על כל הרשימות פוסטינג ועבור כל מסמך ברשימה מצרפת אותו לרשימת המסמכים הפוטנציאלים. השיטה מחזירה מילון של מזהה מסמך והמסמך.

1. private TermPostingList GetPostingLineFromPostingFile(string term, int pointerLine)

שיטה זו מקבלת מילון של term ואת מספר השורה בקובץ הפוסטינג, השיטה טוענת את שורת הפוסטינג לפי השורה שניתנה לה מתוך קובץ הפוסטינג שנמצא בתיקיית הPosting ומתחיל בתו שבו מתחילה המילה ( או special במידה ומדובר בתו מיוחד( . השיטה מחזירה רשומת פוסטינג.

שיטות הקשורות לDocument Summary Functionality :

1. public Dictionary<int, Tuple<string, float>> Find5SignificantSentencesInDocument(string docName)

שיטה זו מקבלת שם של מסמך, מאתרת את המסמך ברשימת המסמכים שב\_controller ובעזרת שדה הFoldarName טוענת את הקובץ לזיכרון. לאחר טעינת הקובץ נפצל את המסמכים בדומה לפיצול שעשינו בחלק הראשון בReadFile ולפי שדה הLocationInFolder נאתר את המסמך שאנו צריכים. לאחר איתור הטקסט הרלוונטי השיטה תשמש בשיטה 6 על מנת לבצע פיצול של הטקסט למשפטים. ולאחר מכן תשמש בשיטה 7 לטובת אינדוקס המשפטים וקבלת הדירוג עבור כל מסמך. נחזיר את חמשת המשפטים בעלי הדירוג הגבוהה ביותר במסמך.

1. public Dictionary<int, string> SplitTextToSentences(string searchedDocumentText)

השיטה מקבלת מסמך ומפצלת אותו למשפטים על ידי מעבר על האותיות בטקסט ושרשור עד אשר מגיעים לתו שהוא נקודה ושלאחריו מופיע רווח, ירידת שורה או טאב, במצב זה נוסיף את השורה למילון יחד עם מיקומה בטקסט, נאפס את המשפט הנוכחי ונמשיך עד סיום הטקסט. השיטה תחזיר מילון של המשפטים הנמצאים בטקסט כך שמשפט מוגדר מנקודה לנקודה ועבור כל משפט המפתח יהיה המיקום של המשפט בטקסט.

1. public Dictionary<int, Tuple<string, float>> IndexSentencesInDocument(Dictionary<int, string> sentencesInText, string docHeadLine)

שיטה זו מקבלת מילון של משפטים בטקסט ואת כותרת המסמך. השיטה מתייחסת למסמך כאל מאגר של משפטים – במקביל למאגר המסמכים כולו, ואל כל משפט במסמך כאל מסמך בפני עצמו – במקביל למסמך במאגר. השיטה תאנדקס את המשפטים בצורה הבאה:

ע"י מעבר על כל משפט ופרסור כל משפטים למילים על ידי הparser:

* נשמור רשימה של כל המילים במסמך ומספר ההופעות הכולל שלהם במסמך כולו.
* נשמור עבור כל משפט את רשימת המילים שלו ומספר המופיעים של כל מילה במשפט.

לאחר מכן, עבור כל משפט ונחשב עבורו את הTF-IDF בצורה הבאה:

* הidf של כל מילה במשפט יהיה שווה ל: .

ה TF של כל מילה במשפט יהיה שווה למספר המופיעים שלה.

* נסכם לכל משפט את הtf – idf של כל המילים.

בנוסף, נסכם עבור כל משפט גם את מספר המילים שהופיעו בכותרת המסמך ומופיעים גם במשפט.

דירוג של משפט יהיה שווה ל 0.7 הtfidf שלו ו- 0.3 מספר המילים מתוכו שמופיעות בכותרת המסמך.

השיטה תחזיר מילון של מיקום המסמך, המשפט והדירוג שלו.

**מחלקת Ranker:**

מחלקה זו מקבלת רשימת מסמכים פוטנציאלים לאחזור ומחשבת עבור כל מסמך את הדירוג שלו ביחס לשאילתה הנתונה.

קלט: רשימה של מסמכים פוטנציאליים.

פלט: רשימה של מסמכים עם הדירוג שלהם.

משתני מחלקה מרכזיים:

* Controller, עמוד התווך של המנוע, מרכז את כל המידע של המנוע.

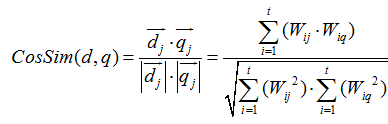
שיטות:

1. public Dictionary<Document, double> rankDocumentsForQuery(Dictionary<int, Document> potentialDocs, Dictionary<string, TermPostingList> queryPostingsList, string query

שיטה זו מקבלת מילון של מזהה מסמך ומסמך, מילון של שאילתה ורשימת הפוסטינג שלה והשאילתה עצמה. השיטה עוברת על כל מסמך ומחשבת עבורו את הדירוג על ידי שיטות 2,3 ו-4 לפי הנוסחה הבאה:

1. public double CosSim(int docId, Document potentialDoc, Dictionary<string, TermPostingList> queryPostingsList, string query)

שיטה זו מקבלת מסמך אחד פוטנציאלי ומילון של מילים שלכל מילה רשומת פוסטינג ואת השאילתה עצמה. השיטה מחשבת את ערך הcosine similarity לפי הנוסחה:



את הנתונים אנו שולפים מהמידע שהפקנו בחלק הראשון של המנוע בתהליך האינדוקס:

– המשקל של המילה במסמך. יחושב ע"פ .

* נשלוף מרשומת הפוסטינג של המילה, נשתמש בשיטה שנמצאת במבנה הנתונים של הרשומה. את ה TF ננרמל לפי אורך המסמך ששמור לנו כשדה בDocument.
* חישבנו גם כן בחלק הראשון של המנוע ושמרנו אותו עבור כל term ב בתור שדה ב.

– המשקל של המילה בשאילתה.

* מספר המופיעים של המילה בשאילתה הנתונה.

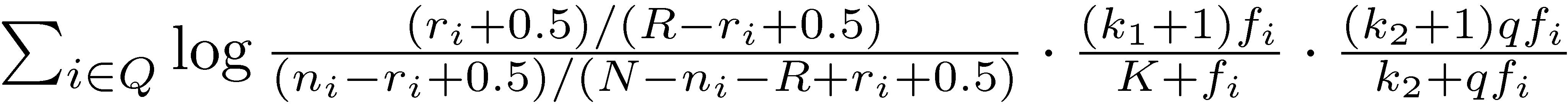
תוך כדי מעבר על המילים בשאילתה נחשב את ה של כל מילה ונבנה את ה. המונה עבור נוסחת ה .

על מנת לחשב את וקטור ה עבור המסמך הוספנו פרמטר נוסף לכל מסמך – TotalTFIDF וחישבנו אותו בריצה נפרדת עבור כל המסמכים.

לבסוף נחלק את המונה ב- TotalTFIDFשל המסמך מנורמל לפי אורך המסמך ונכפיל בשאותו נחשב על ידי סכימת כל ערכי במעבר על השאילתה.

1. public double BM25(int docId, Document potentialDoc, Dictionary<string, TermPostingList> queryPostingsList, string query)

שיטה זו מקבלת מסמך אחד פוטנציאלי ומילון של מילים שלכל מילה רשומת פוסטינג ואת השאילתה עצמה. השיטה מחשבת את ערך ה BM25 לפי הנוסחה:



את הנתונים אנו שולפים מהמידע שהפקנו בחלק הראשון של המנוע בתהליך האינדוקס:

- נשלוף ממילון האינדקס את הdf ששמור עבור המילה.

- נשלוף משורת הפוסטינג של המילה על ידי שימוש בשיטה .

עבור חישוב :

אורך המסמך שמור כשדה לכל מסמך.

את אורך המסמך הממוצע חישבנו בריצה נפרדת על ידי מעבר על כל ארכי המסמכים ולבסוף חישוב הממוצע.

ביצוע ניסויים אמפיריים וחיפוש במאמרים בנושא עלה כי הערכים שהניבו את התוצאות המיטביות הם:

**מחלקת Controller:**

מחלקה זו מהווה את עמוד התווך של המנוע ומפעילה את כל הפונקציונליות הנדרשת ומחזיקה את מבני הנתונים הנדרשים.

משתני מחלקה מרכזיים שהתווספו:

* DocumentSummary המייצג האם המשתמש בחר באופציית תקצור המסמך.
* SavingQueryResultPath השומר את הנתיב אליו המשתמש רוצה לשמור את קובץ התוצאות.
* SavingFileName השומר את שם הקובץ שהמשתמש בחר עבור קובץ תוצאות השאילתה.
* SearchDone , delegateManager המודיע על סיום חיפוש השאילתות ושולח הודעה עם הזמן שלקח החיפוש ומספר המסמכים שחזרו.
* Searcher, מופע של Searcher המכיל את כל השיטות התומכות בחיפוש תוצאות שאילתות.
* QueryFilesNameResults מילון המכיל מס' סידורי של שאילתה ועבור כל שאילתה את תוצאות המסמכים.
* QueryResults מכיל את תוצאות השאילות, מס' שאילתה ורשימה של המסמכים החוזרים ודירוגם.
* QueryID מספר אקראי לשאילתות ידניות.

הוספנו למחלקה זו שיטות המטפלות בחיפושי שאילתות וביצירת תקציר למסמך, להלן השאילתות:

1. public void SearchQueriesFromFile(string path)

שיטה זו מקבלת נתיב לקובץ המכיל שאילתות, מפרסרת את הקובץ ע"פ המבנה הנתון בעבודה ויוצרת מבנה נתונים המכיל את מספר השאילתה והשאילתה כך שכל שאילתה מכילה את הכותרת שמופיעה בקובץ ואת תיאור השאילתה. עבור כל שאילתה שיצרנו נקרא לשיטה search בSearcher המחזירה 50 מסמכים לשאילתא כבר מדורגים. נשמור את הרשימה הזו במילון עבור השאילתה. ונעדכן את ה QueriesResults וה- QueiresFileNames הנוכחים לתוצאות שהוחזרו מהשיטה.

1. CleanDescriptionAndNarrative(string text)

שיטה זו מנקה מילים שזיהינו כ- stopWords ל- description ול narrative ( למרות שלבסוף החלטנו לא לשרשר לשאילתה את הnarrative כי ראינו שפירוט נוסף פוגע בערכי התוצאות ) . המילים שזיהינו הם: document, discussing, relevant, information וכו'..

1. private TermPostingList GetPostingLineFromPostingFile(string term, int pointerLine)

שיטה זו היא שיטת עזר המביאה את רשומת הפוסטינג המלאה עבור הterm הספציפי לטובת חישוב הtotaltfidf שביצענו בעבור כל המסמכים.

1. public void AddTfIdfToDocuments()

שיטה זו עוברת על כל המילים במילון ומעדכנת את שדה הtotaltfidf של כל המסמכים בהם היא מופיעה. עבור כל מילה שלפנו את רשומת הפוסטינג המלאה על ידי שיטה 3 ועל ידי מעבר על הרשומה איתרנו את המסמכים בהם היא מופיעה.

1. internal void SearchReset()

שיטה זו מאפסת את כל מבני הנתונים שהחיפוש עושה בהם שימוש – documents,cache,invertedIndex. ובנוסף מוחקת את הקובץ תוצאות האחרון שנשמר על ידי שימוש בpath האחרון ששמור במערכת.

1. public void Search(string query)

שיטה זו משתמשת במחלקת ה Searcher ובשיטה search שבה ומעדכנת את ה QueriesResults הנוכחי לתוצאות שהוחזרו מהשיטה ואת וה- QueiresFileNames.

1. public void WriteResultsInfo()

שיטה זו כותבת לקובץ את התוצאות השמורות במערכת בהתאם לפורמט המבוקש להשוואה מול treceval.

1. public Dictionary<int, Tuple<string, float>> Search5SignificantSentences(string docName)

שיטה זו משתמשת בSearcher ובשיטה Find5 SignificantSentences המחזירה את המשפטים הרלוונטים ביותר במסמך. כל מופע במילון מייצג משפט באופן הבא: מיקום המשפט במסמך, המשפט עצמו והדירוג שלו.

אלגוריתם הדירוג – פירוט נוסף :

כיוון שבעבודה חויבנו לעבוד לפי מדד ה cosine similarity הכנסנו מדד זה לחישוב הדירוג שלנו. יישמנו את אלגוריתם המדד כפי שמפורט בתוך מחלקת הRanker. לאחר מכן ראינו כי עבור 100 אחוז דירוג של cosSim אנו מקבלים תוצאות פחות טובות והבנו כי יש להוסיף דירוג. הדירוג שבחרנו הוא BM25. מדד זה שיפר לנו את תוצאות האחזור מאוד וניסינו להריץ אותו עם מספר רב של פרמטרים, שכל אחד מהם בדקנו באינטרנט באתרים שונים ובמאמרים. לבסוף, החלטנו להוסיף גם את מדד הכותרת שחישבנו לפי כמות הterms במסמך אשר מופיעים בכותרת. לאחר בדיקות רבות באינטרנט באתרים שונים ובמאמרים כמו:

# Improvements to BM25 and Language Models Examined,

# Advances in Knowledge Discovery and Data Mining

הבנו כי את k2 כדאי לאתחל ל0 ואת k1 ל1 ואת b ל0.75.

כיוון שמדד הbm25 לפי בדיקתנו הוא הגבוה ביותר נתנו לו את המקדם הגבוה ביותר במדד 0.9, הכותרת יהיה 0.099 ומדד הcosine יהיה 0.01 כיוון שראינו שהוא מוריד לנו את הדירוג באופן יחסי.

כמו כן, לפי ניסויים אמפיריים הגענו לפרמטרים הנבחרים.

אלגוריתם הדירוג עובד בצורה הבאה:

1. ה searcher מקבל שאילתה למתודה search:
   1. פרסור השאילתה וקבלת מילון המילים המוכלות בשאילתא ומספר המופיעים שלהן.
   2. איסוף של רשומות הפוסטינג של מילות השאילתה
   3. הוצאת המסמכים מתוך רשומות הפוסטינג ב1.2
   4. שליחת השאילתה המקורית והמפורסרת ורשימת המסמכים הפוטנציאליים לRanker
      1. ב:Ranker
         1. עבור כל מילות השאילתה נעדכן את במילון המאחד את כל המשקלים של מילות השאילתות ובנוסף מעדכן את ה sumOfWiq.
         2. עבור כל מסמך נעדכן את הדירוג משתמש בשיטות של שלושת הדירוגים לחישוב הדירוג הסופי לפי המשקלים שציינו.

נתונים התומכים בתהליך האחזור:

1. TF עבור כל מסמך ברשומת הפוסטינג של המילה, עבור החישובים במדדי הcosSim וה-BM25.
2. IDF עבור כל מילה במילון הinvertedIndex עבור החישובים במדדי הcosSim וה-BM25.
3. DF עבור כל מילה במילון הinvertedIndex עבור מדד ה BM25.
4. TotalTFIDF עבור כל מסמך ברשימת ה DocumentsDataList, עבור מדד הcosSim.

השתמשנו בנתונים אשר שמרנו בחלק הראשון במילון האינדוקס, ברשימת המסמכים וברשומת הפוסטינג.

אלגוריתם דירוג המשפטים במסמך:

על מנת לדרג את המשפטים במסמך הקבלנו את תהליך הדירוג לתהליך דירוג של מסמכים במאגר רק לא על פי שאילתה נבחרת אלא לפי שאר המשפטים במסמך.

משפט - > מסמך

מסמך שהוא אוסף של מסמכים -> מאגר

אלגוריתם דירוג המשפטים עובד בצורה הבאה:

1. ה searcher מקבל מסמך לחיפוש:
   1. פרסור המסמך למשפטים לפי נקודה.
   2. אתחול מילון אשר יכיל את מילות המסמך ומספר המשפטים בהם היא מופיעה.
   3. אתחול מילון אשר יכיל עבור כל משפט את המילים מהם המשפט מורכב וערכי הtf שלהם ביחס למשפט.
   4. עבור כל משפט במסך:
      1. הכנת מילוני פוסטינג עבור כל משפט.
   5. עבור כל מילון פוסטינג של משפט:
      1. חישוב totaltfidf ( פירוט במתודה עצמה במחלקת הSearch) – שקול לסכום כל הtfidf של המילים במשפט. כאשר tf זה מספר המופיעים של המילה במשפט וidf זה היחס בין מספר המשפטים הכולל לבין מספר המשפטים בהם המילה מופיעה. (לפי נוסחת הlog)
      2. חישוב headline rank – שקול למספר המילים במשפט שמופיעות גם בכותרת.
2. החזרת 5 המשפטים עם הדירוג הגבוה ביותר.

הנוסחה הסופית בה השתמשנו היא:

ניסינו להכניס פרמטרים נוספים לחשיבות המשפטים כמו מיקום המילים, מיקום המשפט אך לדעתנו נוסחה זו היא האופטימלית ביותר.

קוד פתוח:

לא ביצענו שימוש בקוד פתוח.

הערכת המנוע:

**ללא stemming:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Query#** | **Query Title** | **Percision** | **Recall** | [**Percision@5**](mailto:Percision@5) | [**Percision@15**](mailto:Percision@15) | [**Percision@30**](mailto:Percision@30) | [**Percision@50**](mailto:Percision@50) |
| 351 | Falkland petroleum exploration | 0.42 | 0.4375 | 0.2 | 0.2 | 0.4 | 0.42 |
| 352 | British Chunnel impact | 0.22 | 0.04471545 | 0.4 | 0.4667 | 0.3667 | 0.22 |
| 358 | blood-alcohol fatalities | 0.42 | 0.41176471 | 0.2 | 0.4667 | 0.4333 | 0.42 |
| 359 | mutual fund predictors | 0.16 | 0.28571429 | 0 | 0.1333 | 0.1667 | 0.16 |
| 362 | human smuggling | 0.12 | 0.15384615 | 0 | 0.0667 | 0.1333 | 0.12 |
| 367 | piracy | 0.26 | 0.07027027 | 0 | 0.2667 | 0.3 | 0.26 |
| 373 | encryption equipment export | 0.14 | 0.4375 | 0.4 | 0.4667 | 0.2333 | 0.14 |
| 374 | Nobel prize winners | 0.52 | 0.12745098 | 0.8 | 0.4667 | 0.5667 | 0.52 |
| 377 | cigar smoking | 0.22 | 0.30555556 | 0 | 0.2 | 0.2667 | 0.22 |
| 380 | obesity medical treatment | 0.06 | 0.17647059 | 0 | 0.1333 | 0.1 | 0.06 |
| 384 | space station moon | 0.26 | 0.25490196 | 0.6 | 0.3333 | 0.2333 | 0.26 |
| 385 | hybrid fuel cars | 0.28 | 0.16470588 | 0 | 0.0667 | 0.2333 | 0.28 |
| 387 | radioactive waste | 0.26 | 0.17808219 | 0.2 | 0.2 | 0.3333 | 0.26 |
| 388 | organic soil enhancement | 0.26 | 0.26 | 0.4 | 0.4 | 0.4 | 0.26 |
| 390 | orphan drugs | 0.26 | 0.10655738 | 0.4 | 0.4667 | 0.3667 | 0.26 |
| **TOTAL** |  | 0.257333333 | 0.15551974 | 0.24 | 0.2889 | 0.3022 | 0.257333333 |
|  |  |  |  |  |  | **Map** | 0.0787 |
|  |  |  |  |  |  |  |  |

**עם stemming:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Query#** | **Query Title** | **Percision** | **Recall** | [**Percision@5**](mailto:Percision@5) | [**Percision@15**](mailto:Percision@15) | [**Percision@30**](mailto:Percision@30) | [**Percision@50**](mailto:Percision@50) |
| 351 | Falkland petroleum exploration | 0.46 | 0.47916667 | 0.2 | 0.4 | 0.4333 | 0.46 |
| 352 | British Chunnel impact | 0.22 | 0.04471545 | 0.4 | 0.4667 | 0.3667 | 0.22 |
| 358 | blood-alcohol fatalities | 0 | 0 | 0 | 0 | 0 | 0 |
| 359 | mutual fund predictors | 0.08 | 0.14285714 | 0 | 0.1333 | 0.0667 | 0.08 |
| 362 | human smuggling | 0.12 | 0.15384615 | 0 | 0.0667 | 0.1333 | 0.12 |
| 367 | piracy | 0.16 | 0.04324324 | 0 | 0 | 0.1333 | 0.16 |
| 373 | encryption equipment export | 0.12 | 0.375 | 0.2 | 0.4 | 0.2 | 0.12 |
| 374 | Nobel prize winners | 0.32 | 0.07843137 | 0.6 | 0.3333 | 0.3667 | 0.32 |
| 377 | cigar smoking | 0.32 | 0.44444444 | 0 | 0.2 | 0.3 | 0.32 |
| 380 | obesity medical treatment | 0.04 | 0.28571429 | 0 | 0.0667 | 0.0667 | 0.04 |
| 384 | space station moon | 0.22 | 0.21568627 | 0.6 | 0.3333 | 0.2333 | 0.22 |
| 385 | hybrid fuel cars | 0.28 | 0.16470588 | 0 | 0.0667 | 0.2 | 0.28 |
| 387 | radioactive waste | 0.18 | 0.12328767 | 0 | 0 | 0.1667 | 0.18 |
| 388 | organic soil enhancement | 0.48 | 0.48 | 0.4 | 0.6667 | 0.5667 | 0.48 |
| 390 | orphan drugs | 0.3 | 0.12295082 | 0.6 | 0.6667 | 0.4333 | 0.3 |
| **TOTAL** |  | 0.22 | 0.13295729 | 0.2 | 0.2533 | 0.2444 | 0.22 |
|  |  |  |  |  |  | **Map** | 0.0717 |
|  |  |  |  |  |  |  |  |

סיכום:

בעיות בהן נתקלנו:

* הבעיה העיקרית לדעתנו היא גודל המאגר שעמד לרשותנו לעומת יכולות המחשבים שברשותנו ויכולות סביבות העבודה איתם פיתחנו את המנוע. נתבקשנו לאנדקס מאגר בגודל של כחצי מיליון מסמכים ומאגר מילים ייחודיות של כמילון מילים כך שכל רשומת פוסטינג למילה במילון יוצאת בגודל מאוד גודל. על מנת לבצע חיפוש שאילתה בזמן סביר אשר לא עולה על מספר דקות בודדות נאלצנו לחשוב על פתרונות של קריאת חלק מהחוליות ברשימה, הרלוונטיות ביותר עבורנו ורק עבור לבצע את החיפוש. אך מצב זה גורם לאיבוד נתונים אשר יכולים להגדיל את איכות התוצאות והשקענו זמן רב במציאת הערך שיהווה איזון בין איכות תוצאות לזמן ריצה סביר.
* בנוסף ובהמשך לסעיף הקודם השקענו זמן רב במחשב כיצד לשלב את הCache בתהליך חיפוש השאילתה. בחלק הקודם הגענו למסקנה כי כל מילה שהTFIDF שלה עולה על ערך סף מסוים תיכנס לCache, אך יישמרו עבורה רק 50 חוליות של מסמכים. כעת, שהבנו כי קריאה מרובה של מסמכים תגדיל את איכות התוצאות והערך של החוליות האופטימלי שהגענו אליו היה 7000 מסמכים מצאנו כי לCache אין משמעות כי אין לזיכרון התוכנית מקום לאחסן כמות כזו של נתונים, בטח שתעמוד בדרישה כי על מבנה הנתונים להיות עד 30 אחוז מגודל המילון. במצב זה מצאנו כי הcache מיותר כיוון שמתוך מיליון מילים על מנת לעמוד בכל התנאים נוכל להחזיק פחות מ50 מילים בcache ומכיוון שזהו אינו שיפור וכן השקענו זמן רב בחיפוש ערך הסף בחרנו לא להשתמש בcache בחלק זה של העבודה.
* בעיה נוספת שעמדה בפנינו היא קביעת פרמטרים המדדים השונים שהשקענו זמן רב בניסויים אמפיריים ושינויים של פרמטרים אלו והבנה מהם הפרמטרים האידיאליים ע"י בדיקה במקורות מידע ומאמרים. כמו כן, ניסינו להבין מה מאפיין תוצאות שניתנו כרלוונטיות בדוגמה וע"י כך להבין אילו מין המדדים הם הרלוונטיים ביותר ואיך לבנות את השאילתה בצורה הטובה ביותר.

שיפורים אפשריים שנוכל להציע יהיו ריצה על רשומת פוסטינג מלאה ובצורה כזו מקסום של התוצאות ושימוש בcache על מנת להקטין את זמני הריצה, שינויים אלו יתאפשרו בתנאי שנשנה את סביבת העבודה ונריץ את המערכת על גבי שרת חיצוני בעל אחסון רב עם מעבד חזק.