**חלק 2 – מנוע חיפוש**

1. **הסבר אופן פעולת המנוע**

פירוט שינויים והוספה של מחלקות, הפרויקט בוצע בארכיטקטורת MVVM לטובת הכמסה בין ה-Model ל-View.

**שכבת ה-View (תוספות):**

**מחלקת MainWindow**

private void LoadSuggestions()

מתודה זו טוענת ההמלצות לשאילתות אשר רושם המשתמש לזכרון המערכת, האלגוריתם יפורט בהמשך.

private bool countResults(string search, object item)

מתודה זו משמשת כפילטר לטעינת התוצאות אשר יוצגו למשתמש בעת כתיבת שאילתה – עד 5 ורלוונטית.

private void btn\_LoadQuery\_Click(object sender, RoutedEventArgs e)

מתודה זו מאפשרת למשתמש לטעון את קובץ השאילתות מהמחשב באמצעות .

private void btn\_Search\_Click(object sender, RoutedEventArgs e)

מתודה זו קוראת לפעולת החיפוש בעת לחיצה על כפתור ה-.

private void queryAutoComplete\_TextChanged

מתודה זו אחראית להצגת ההשלמה האוטומטית שתוצג למשתמש בעת שינוי טקסט ב-.

private void btn\_SaveResults\_Click

מתודה זו מאפשרת למשתמש לשמור את ה-Results של השאילתה במקום שבו יבחר באמצעות .

private void find\_btn\_Click

מתודה זו מציגה למשתמש מילים נרדפות בעת לחיצת כפתור לאחר שרשם אותן ב-.

private enum Languages

זוהי מחלקת אשר מציגה למשתמש את השפות שלפיהן יכול לבצע את החיפוש.

**שכבת ה-View-Model:**

**מחלקת MyViewModel**

internal void Search(System.IO.FileInfo queryFile, bool stemmerChecked, string postingPath, string language)

מתודה זו קוראת לחיפוש ב- לאחר שהתקבלה בקשה מה-View, זהו חיפוש של קובץ שאילתות.

internal void Search(string queryString, bool stemmerChecked, string postingPath, string language)

מתודה זו קוראת לחיפוש ב- לאחר שהתקבלה בקשה מה-, זהו חיפוש של מחרוזת משתמש.

internal string GetResults()

מתודה זו קוראת לקבלת תוצאות של דירוג מסמכים של השאילתה/שאילתות שהתקבלו מה-.

**שכבת ה-Model:**

**מחלקת MyModel**

בדומה למחלקת , מתודות אלו ממשיכות את הקישור בין ה- למחלקת ה- אשר מבצעת את החיפוש.

public void Search(FileInfo queryFile, bool stemmerChecked, string postingPath, string language)

public void Search(string queryString, bool stemmerChecked, string postingPath, string language)

public string GetResults()

**מחלקת Searcher**

internal void Search(List<string[]> queryList, Indexer indexer, bool stemmerChecked, string postingPath, string language)

מתודה זו מקבלת את השאילתה או קובץ השאילתות, מפרקת את השאילתות ושולחת אותן למחלקת ה-Ranker אשר מבצעת דירוג למסמכים בהתאם לרלוונטיות שלהם לכל שאילתה.

internal string GetResults()

מתודה זו מחזירה את תוצאות דירוג המסמכים לפי השאילתות.

**מחלקת Indexer**

internal Tuple<int, int> GetLengths()

מתודה זו מחזירה לנו את כמות המסמכים שקיימת במאגר ואת כמות המילים הכוללת במאגר לטובת חישוב עתידי של דירוג המסמכים.

private void splitCorpus()

מתודה זו מתבצעת בסיום יצירת כלל המילונים ומבצעת פיצול של כל מילון ל-*Partitions* גודל שהוגדר מראש (65 מחיצות) לכל מילון לטובת מזעור פעולות קלט פלט של המערכת על קבצים גדולים.

**מחלקת Ranker**

internal void Rank(string[] query, Indexer indexer, bool stemmerChecked, string postingPath, string language)

מתודה זו הינה המתודה הראשית אשר אחראית לביצוע הדירוג של המסמכים לאחר קבלת שאילתה בודדת בכל הפעלה, קיימת התייחסות לשפות שנבחרו (אם בכלל) והאם המסמכי ה- הם .

private void LoadDocumentDetails()

מתודה זו טוענת את מילון המסמכים בלבד (הוא מפורק לחלקים) על מנת להחזיקו בזכרון ולא לבצע קריאות מיותרות.

private void CreateGlobalList(bool stemmerChecked, string postingPath)

מתודה זו יוצרת רשימה של מסמכים שרלוונטיים לחיפוש השאילתה וכבר מתחילה באיסוף פרמטרים אשר ישמשו לחישוב ה-.

private void RankTheQuery(string postingPath)

מתודה זו עוברת על רשימת המסמכים הרלוונטיים לשאילתה ומדרגת כל מסמך בעזרת אלגוריתם .

private double BM25(string name, int iter)

מתודה זו ממשת את אלגוריתם לדירוג מסמכים.

private string GetPath(string word, bool stemmerChecked, string postingPath)

מתודה זו משמשת לקבלת נתיב גישה לקובץ שמכיל נתונים על מילה מסויימת שחוקרים אותה.

public string GetResults()

מתודה זו מחזירה את תוצאות דירוג המסמכים בסדר יורד מהרלוונטי ביותר (עד 50 לכל שאילתה).

**מחלקת SynonymsFinder**

מחלקה זו מחזירה מילים נרדפות למחרוזת אותה רוצים לבחון.

public static List<string> FindSynonym(string reference)

מתודה זו מבצעת קריאה ל- *אשר מחזירה לנו רשימה של מילים נרדפות למחרוזת ששלחנו.*

**אלגוריתם השלמת משפטים:**

על מנת לבצע השלמה אוטומטית לשאילתת משתמש, התבצעו הפעולות הבאות:

1. בתהליך יצירת ה-Posting שומרים קובץ ובו פרטי ה-Topics של כל מסמך, אלו הן הכותרות של כל מסמך.
2. לאחר שהמשתמש בוחר כתובת לטעינת Posting או שיוצר מילון חדש, המערכת פותחת את קובץ ה-Topics ושולפת משם את רשימת הנושאים.
3. המערכת יוצרת רשימה חדשים של Shingles, מחרוזות של 2 מילם לכל Topic.
4. בעת הקלדת טקסט של המשתמש, בעזרת פונקציונאליות של פקד AutoComplete, המערכת מבצעת חיפוש על רשימת המחרוזות החדשה.
5. לכל אפשרות קיימת ברשימה, המערכת מבצעת פילטור תוצאות על בסיס – האם המשפט מתחיל במילה והאם חרגנו מ-5 תוצאות, במידה והתשובה היא חיובית, מציגים למשתמש את ההמלצה.

הרציונל העומד מאחורי פעולות אלו, הינו לקחת את רשימת הנושאים אשר קיימים כבר במאגר המסמכים ולהמליץ לו לבצע שאילתה אשר מתאימה מבחינה סמנטית במדויק למה שקיים בכותרת של המסמך. פעולה זו עוזרת תורמת ל-2 מאמצים:

1. המלצה אפקטיבית של תוכן למה שהמשתמש מחפש.
2. התאמה לאלגוריתם חיפוש ודירוג המסמכים, מה שיביא לתוצאות אפקטיביות יותר.

במידה והיה לי מידע לגבי מילים נרדפות פופולאריות אשר משתמשים מחפשים ביחד עם המילה שהוקלדה, הייתי מבצע דירוג טוב יותר של ההמלצות אשר יוצגו בפני המשתמש, כך 5 התוצאות או פחות שיוצגו יהיו רלוונטיות ביותר למה שהשמתמש בסבירות גבוהה מחפש.

**אלגוריתם דירוג מסמכים:**

אופן פעולת דירוג המסמכים:

1. קבלת שאילתה.
2. קביעת פרמטרים שכבר ניתן לקבוע שרלוונטיים לחישוב .
3. טעינת המידע על הקבצים לזכרון (לטובת קבלת שם ונושא מסמך).
4. יצירת רשימה כללית של מסמכים אשר רלוונטיים לכל מילות השאילתה שהתקבלה ע"י בדיקה במילונים האם המילה קיימת, האם קיימת בשפה שהתקבלה (אם התקבלה) ושאיבת רשימת המסמכים אשר בהם מופיעה המילה.
5. מעבר על כל מסמך ברשימה שיצרנו וביצוע דירוג של המסמך בעזרת חישוב של סכימת הפעלת אלגוריתם על כל מילה אשר מופיעה בשאילתה.
6. הכנסת המסמך לרשימת מסמכים שעברו דירוג יחד עם הציון הכללי אשר קיבל.
7. ביצוע מיון של הרשימה שהתקבלה.
8. הפיכה של הרשימה לטובת קבלת הדירוגים הגבוהים ראשונים.
9. יצירת רשימה חדשה מודרגת של שם מסמך ודירוג של המסמך בסדר יורד עם הגבלה של עד 50 מסמכים לשאילתה.
10. החזרת רשימה זו – זהו דירוג המסמכים ביחס לשאילתה שהתקבלה.

דירוג המסמכים התבצע על פי אלגוריתם אשר הופעל על כל מסמך ובצורה פרטנית על כל שמופיע בשאילתה ביחס למסך שבודקים.

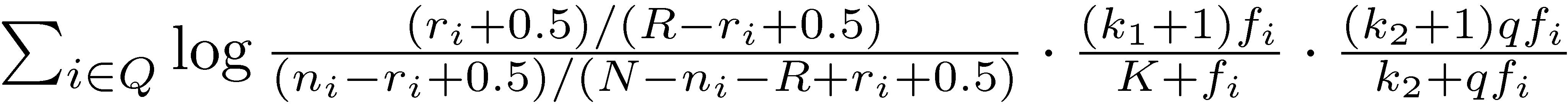
לפני כל איטרציה נאסף המידע הבא:

כמות המסמכים שבהם ה- הופיע, כמות המסמכים הכללית במאגר תדירות ה- בשאילתה.

בכל איטרציה על מהשאילתה ביחס למסמך הנבדק, נאספו הפרטים אודות:

אורך המסמך, אורך מנורמל של המסמך וכמות המופעים של ה- במסמך.

מידע זה שימש להפעלת הנוסחא של האלגוריתם ובסיום בדיקת כל ה-, ביצוע סכימה של הדירוג:



בדקתי אפשרויות נוספות של הפעלת משקל שונה על הדירוג כגון:

1. האם המילה כותרת.
2. ניסיון להוסיף לחישוב משקל של .

ב-2 הבדיקות, קיבלתי התאמה נמוכה יותר של הדירוג שהתקבל ביחס ל-qrels.txt, לכן נשארתי עם אלגוריתם עם המשקולות:

לאחר מספר בדיקות ומשחק עם המדדים (בעיקר של ו-), השארתי את ערכים אלו, כל ערכים אחרים החזירו תוצאות פחות מדויקות.

**אלגוריתם סמנטי:**

*האלגוריתם הסמנטי שבו השתמשתי הוא API אשר מקבלים אותו מ: <http://www.datamuse.com/api>*

*זהו מנוע לחיפוש מילים לפי שאילתות משתמש אשר מחזיר מילים נרדפות או מילים אשר קשורות ב-Context לשאילתה ששלחנו.*

*פעולת קבלת המילים מתבצעת ע"י שליחת בקשת HTTP עם מחרוזת של מילה או מילים וקבלת אובייקט מסוג JSON אשר מכיל רשימה של מילים נרדפות או בעלות הקשר של Context, קרי קשר סמנטי, קיימת הגבלת ברירת מחדל ל-100 תוצאות בעלות הציון הגבוה ביותר.*

*בקבלת התוצאות קיימת התייחסות לפרמטרים הבאים:*

1. *משמעות המילה (סמנטיקה).*
2. *איך המילה נשמעת.*
3. *כיצד מאייתים את המילה.*
4. *מילים שקשורת למילה.*

*כל תוצאה שמתקבלת במערכת, עוברת דירוג לפי רלוונטיות לשאילתה והאם מה שמוחזר הוא:*

1. *מילת הנושא (המרכזית).*
2. *מילה משמאל לשאילתה – לפי ההקשר.*
3. *מילה מימין לשאילתה – לפי ההקשר.*

*מקורות המידע שעליהן מסתמכת המערכת:*

1. *ידע פונטי – מה שנשמע כמו מה שמחפשים לפי מילון: The CMU Pronouncing Dictionary.*
2. *ידע מבוסס Corpus לפי:* 
   1. *The Google Books Ngram Viewer.*
   2. *Word2vec (גם של Google).*
   3. *PPDB: The Paraphrase Database.*
3. *ידע מבוסס Context לפי:* 
   1. *WordNet 3.0.*
   2. *OneLook.*

*ה-API הינו מאד מוצלח ובשימוש של מפתחים רבים, הוא יותר אפקטיבי מביצוע של פעולות LSI על גבי המאגר והצגתן.*

**פירוט ה-Posting:**

*ב-Posting נשמר המידע הבא:*

1. *מילון של כלל המילים וכמות המופעים שלהם.*
2. *מילון של כלל המסמכים שבו מידע אודות כל מסמך:*
   * שם מסמך.
   * נושא (נוסף מעבר למה שהתבקש).
   * תאריך (נוסף מעבר למה שהתבקש).
   * שפה.
   * כמות הופעת Term מקסימאלית (maxTF).
   * כמות ביטויי ייחודיים במסמך.
3. *מילון אשר מכיל את ה-Topics (כותרות) של כל מסמך לטובת ביצוע השלמה אוטומטית לשאילתות משתמש (קיים פירוט מורחב למעלה).*
4. *מילון לפי אות ומספרים אשר מחולק למחיצות ובו מידע אודות: מילים, מספרים, לכל ביטוי קיים:*

**TermInfo**

* + - כמות המסמכים שבהם ה-Term מופיע (df).
    - צמדי DictPair (מפורט למטה).

לכל צמד DictPair קיים:

* כמות הופעת ה-Term במסמך (tf).
* שם המסמך.
* אורך המסמך (התווסף לחישוב מהיר יותר של שאילתות).

1. *מילון (inverted index dictionary) אשר מכיל מידע אודות מילים וכמות המופעים שלהן במאגר.*

שינוי נוסף מעבר להוספת אורך המסמך ל-DictPair והוספת מילון *Topics* שבוצע בחלק זה הוא תהליך חלוקת קבצי ה-Posting למחיצות לפי גדול שהוגדר מראש על למנוע פעולות I/O על קבצים גדולים. איטרציות על מספר קבצים קטנים הראו ביצועים טובים יותר אל מול מעבר על מילון שלם של מילה.

מספר המחיצות המוגדר הוא: 65 מחיצות לכל מילון, כאשר בכל חיפוש עוברים על המילונים וכאשר מוצאים ביטוי מסוים, חודלים את החיפוש.

**שימוש בקוד פתוח:**

*נעשה שימוש ב-2 סוגים של קוד פתוח:*

1. *API של החיפוש הסמנטי (מפורט בהרחבה למעלה).*
2. *שימוש ב-* *WPF Toolkitלטובת הוספת פקד Auto-Complete עם כל הפונקציונאליות שלו אשר מגיע היום יחד עם ה-Visual Studio החדשים כתוסף ליבה ולכן אין צורך להרחיב לגביו מהסיבה שפשוט לא היה מותקן.*

*קישורים:*

[*http://wpftoolkit.codeplex.com/*](http://wpftoolkit.codeplex.com/)

[*https://www.nuget.org/packages/WPFToolkit/*](https://www.nuget.org/packages/WPFToolkit/)

1. **הערכה של המנוע**







1. **סיכום**

העבודה הייתה מאד מעניינת ואני חושב שחד משמעית למדתי המון ממנה, בעיקר:

1. עולם אחזור המידע
2. מימוש אלגוריתמים ונוסחאות מתמטיות לשימוש מעשי
3. התמודדות עם קבצים
4. תכנות

אני חושב שהקשיים העיקריים היו בהתמודדות עם קבצים גדולים, קריאה, שליפת נתונים, אחסון נתונים חכם ועוד. פעם ראשונה שאני מבין שיש משמעות אמיתית לאיך מאחסנים מידע, איך קוראים אותו ואיך שומרים אותו בצורה אופטימלית (גם אם לא תמיד יצא ככה).

קושי נוסף הוא היכולת לבדוק את המנוע, בעיקר מהסיבה שהדבר לוקח המון משאבים מהמחשב ולוקח גם המון זמן, זמן שצריך לנצל אותו בצורה חכמה אך שלא פוגעת בקוד.

כל מה שהיה קשור לחלק הראשון של המנוע – "העבודה השחורה" של לקחת ביטויים, לשנות אותם, לתקן אותם ולשמור אותם בצורה חכמה הייתה מאד מתישה אבל תרמה ליכולת להבין וגם להעריך מערכות שעושות את זה. לכן, באיזשהו מקום אני שמח שזה לא היה איזו מחלקה שאני סתם הורדתי מהאינטרנט והפעלתי.

מה שהייתי עושה אחרת הוא חד משמעית הצורה שבה אני שומר את קבצי ה-Posting, לאחר חלק ב' אני באמת מבין את המשמעות של קריאה וכתיבה לקבצים גדולים או באיטרציות מרובות מאד. הייתי שומר הכל בצורה כמה שיותר מבוזרת אך חכמה ככה שאדע לגשת למילה מסויימת בצורה מדוייקת והקובץ יהיה קטן.

משהו חשוב שלדעתי לא יצא לעבור עליו בתואר הוא עניין שמירת המידע כאובייקטים. בעבודה השתמשתי ב-JSON, לא למדנו על כך בעבר ואולי זו לא דרך המימוש האולטמטיבית, אך לא הכרתי זאת לפני כן ומסתבר שמאד חשוב להכיר את היכולת לשמור קבצים בצורה הזו.

דבר נוסף, אני לא יודע מה הייתי עושה בלי StackOverFlow.