מנוע חלק ב' – דוח

מגישות: נגה אגמון 203573647, אלינור שלום 204326557

1. אופן פעולת המנוע:

השלמה אוטומטית – כדי לבצע את ההשלמה האוטומטית הוספנו ל Indexer את המילון autocompleteDic. המילון שומר עבור כל מילה במילון רשימה של כל המילים שהופיעו אחריה במאגר ואת כמות ההופעות שלהן. בסוף תהליך האינדוקס שמרנו בdictionary עבור כל מילה את חמשת המילים שהופיעו בצמוד לה בתדירות הגבוהה ביותר. חמשת המילים האלו הן המילים שהמנוע שלנו מציע למשתמש כאשר מוקלדת מילה בחיפוש.  
בנוסף שינינו את הstemmer על המילים וכעת הוא מתבצע ב Index במקום ב Parse, כך יכולנו לשמור את המילים המוצעות לפני שהן עוברות stemmer. גם אם המשתמש בחר להריץ את השאילתה עם stemmer אנחנו מניחות שהוא כותב שאילתה באופן אינטואיטיבי וללא stemmer, לכן רצינו שהמילים המוצעות לו בהשלמה האוטומטית יופיעו לפני שהן עוברות stemmer.

במידה והיה לנו מידע על הקונספטים השונים הקיימים במאגר ואת השיוך של כל מילה במאגר לקונספט, היינו יכולות להציע למשתמש השלמת מילים לפי מילים השייכות לאותו הקונספט.   
בנוסף אם היו לנו נתונים עבור כל משתמש, היינו יכולות להשתמש בנתונים אלו כדי להציע לכל משתמש השלמת מילים המתאימה לו. אפשר לבנות פרופיל לכל משתמש על סמך החיפושים הקודמים שלו במאגר וכך לדעת באילו תחומים הוא מתעניין, כך אפשר להציע למשתמש מילים מתחום מסוים שהמשתמש מתעניין בו.

טיפול סמנטי – לשם הטיפול הסמנטי השתמשנו ב thesaurus. לשם כך הורדנו database והוספנו קובץ dll לפרויקט. (דרך הקישור הבא: https://www.codeproject.com/Articles/43495/Spell-Check-Hyphenation-and-Thesaurus-for-NET-with)  
השתמשנו בפונקציונליות של thesaurus למילים נרדפות. עבור כל מילה בשאילתה שקיבלנו חיפשנו בעזרת הDB של thesaurus מילים נרדפות הדומות למילה המקורית. עבור כל מילה נרדפת בדקנו את הדמיון שלה בעזרת N-Gram (עם n=2) שמימשנו בעצמנו. רק אם המילה הנרדפת דומה למילה המקורית, החלפנו אותה במילה המקורית ויצרנו שאילתה חדשה. את השאילתה החדשה אנו שולחים ל Searcher באותו אופן כמו את השאילתה המקורית.

הנתונים בקבצים (posting, dictionary, docs) – כדי לחשב עבור כל מילה את ה proximity השתמשנו במיקומים ששמרנו עבור כל מילה ומסמך בו הופיעה בקבצי הposting. בנוסף לקחנו מקבצי הposting את המזהים של כל המסמכים בהם היא הופיעה כדי להפעיל את פונקציית הדירוג על מסמכים אלו.  
את הקבצים dictionary ו docs טענו לזיכרון. המידע השמור בקובץ docs שימש אותנו במקרים הבאים:  
- שפת המסמך שימשה אותנו כדי לדעת את שפת המסמך במקרה בו המשתמש בחר לקבל מסמכים בשפה מסוימת  
- כמות המילים במסמך שימשה אותנו בפונקציית ה cosSim.  
- כמו כן גם הווקטור של כל מסמך שנשמר בקובץ הdocs שימש אותנו לחישוב פונקציית הדירוג לפי הנוסחה של cosSim.  
המידע מקובץ הdictionary שימש אותנו:  
- המצביע של כל מילה לקובץ הposting שמור ב dictionary ושימש אותנו כאשר הצטרכנו לקחת מידע של מילה ספציפית השמור בPosting.  
- בנוסף השתמשנו בערך הidf של כל מילה בפונקציית הדירוג שלנו.

מחלקת Document-  
לשם חלק זה של המנוע הוספנו למחלקה את **השדות הבאים**:  
מחרוזת header ששומרת את הכותרת של המסמך.  
מילון queryFreq ששומר עבור כל מילה במסמך את כמות המופעים שלה במסמך.  
שדה docVector שהוא ווקטור המסמך המחושב לפי הווקטור בנוסחה של Cos Similarity בה השתמשנו לדירוג המסמכים.

מחלקת PreTextProcess-  
בחלק זה של המנוע השתמשנו בנוסחת הדירוג של cosSim, לשם כך היה צורך לחשב את הווקטור של כל מסמך במאגר ובנוסף להשתמש בנתונים השמורים בdictionary של המנוע. לכן החלטנו לבצע חישוב זה לאחר האינדוקס של כל המאגר.  
- השיטה IndexingDocuments נוצרה לשם שמירה מחודשת של קבצי הdocuments כאשר עכשיו יישמרו כל המסמכים בקובץ עם הווקטור שחושב להם.  
- השיטה CalcCosineSimilarity מקבלת נתיב לתיקייה בה שמורים כל המסמכים במאגר, וה stop\_words. השיטה מבצעת מעבר על כל אחד מהמסמכים, מוצאת את הטקסט של מסמך, מבצעת עליו פירסור ולאחר מכן קוראת לפונקציה CalcDocVector של Indexer שעוברת על כל המילים בטקסט ומחשבת את ווקטור המסמך לפי הנוסחה של cosSim.  
- השיטה SaveDocs בדומה לחלק א תפקידה לשמור את המסמכים לקובץ כאשר כעת היא תשמור גם את הווקטור של כל מסמך.

מחלקת Search-  
**שדות המחלקה:**משתנה סטטי מאותחל ב100 ששומר את מזהה השאילתה שהורצה דרך החלון, המזהה גדל עם כל שאילתה שמריצים.  
שדה של מחלקת ranker בעזרתו נפעיל את פונקציית הדירוג על כל המסמכים הרלוונטיים לשאילתה.  
השאילתה עצמה המיוצגת כרשימה של מילים List<string>.  
מילון שהוא הdictionary של המאגר.  
מילון נוסף השומר את כל המסמכים לפי המזהה שלהם.  
מילון השומר את כל השפות ואת כמות המסמכים הקיימים בכל שפה.  
ומילון filteredDocs המחזיק את כל המסמכים שקיימת בהם לפחות אחת מהמילים בשאילתה (מחושב בהמשך).  
hashset של כל המילים החשובות שהן מילים שאין להן שום מילה נרדפת.

**שיטות המחלקה:**- השיטה FilterDocuments אחראית על סינון המסמכים הרלוונטיים לשאילתה הנוכחית. מסמך יוגדר רלוונטי אם אחת מהמילים בשאילתה נמצאת בו. בנוסף אם המשתמש בחר שפות מסוימות מתוך רשימת השפות הפונקציה תסנן רק מסמכים שהן בשפה שנבחרה ע"י המשתמש.  
- השיטה GetQueryPostingInfo אחראית על הבאת השורה של כל מילה בשאילתה מקובץ הפוסטינג. עבור כל מילה בשאילתה הפונקציה פותחת קובץ פוסטינג רלוונטי (לפי האות הראשונה של המילה) ולפי המצביע השמור במילון, קוראת את שורת הפוסטינג של המילה. הפונקציה מחזירה רשימה של מחרוזות כאשר כל איבר ברשימה הוא שורת הפוסטינג של מילה בשאילתה.  
- השיטה GetAllDocsFromPosting מקבלת מחרוזת שהיא שורה שנקראה מקובץ פוסטינג ומחלצת מתוך מחרוזת זו את כל מזהי המסמכים. בנוסף היא מעדכנת את המילון docsPerTerm של המחלקה ranker כך שישמור עבור כל מילה בשאילתה את כמות המסמכים שהיא מופיעה בהם. בנוסף עבור כל מסמך במחרוזת הפונקציה מעדכנת את השדה queryFreq כך שעבור כל מילה בשאילתה תצוין כמות המופעים שלה במסמך זה.  
- השיטה GetWordsPositions מקבלת רשימת של מחרוזות כאשר כל מחרוזות היא שורה מתוך קובץ פוסטינג. השיטה מחלצת מתוך כל שורה מילה, מזהה מסמך ואת כל המיקומים של המילה במסמך. לבסוף השיטה מעדכנת את המשתנה postingPositionsDic של המחלקה ranker. postingPositionsDic הינו מילון השומר עבור כל מסמך מילון נוסף ששומר מילה ואת רשימת המיקומים של המילה במסמך זה. שדה זה של המחלקה ranker ישמש אותנו לדירוג מסמכים לפי דירוג של proximitySearch.

מחלקת Ranker-  
**שדות המחלקה:**מילון corpusDoc השומר את כל המסמכים שעברו את הסינון ע"י searcher.  
אורך מסמך ממוצע במאגר.  
N שהוא כמות המסמכים במאגר.  
מילון docsPerTerm השומר עבור כל מילה בשאילתה את כמות המסמכים בהם היא מופיעה.  
מילון queryTermIdf ששומר עבור כל מילה בשאילתה את ערך הidf שלה.  
מילון postingPositionsDic ששומר עבור כל מסמך מילון נוסף ששומר מילה ואת רשימת המיקומים של מילה זו במסמך.  
query השאילתה המיוצגת כרשימת מחרוזות.  
hashSet של מילים חשובות (יפורט בהמשך).

**שיטות המחלקה:**- השיטה RankerRelevantDocs עוברת על כל המסמכים לאחר סינון ומדרגת אותם לפי המדדים שהגדרנו: תחילה לפי BM25, לאחר מכן לפי cosSim ולבסוף לפי proximitySearch (בתנאי שהשאילתה כוללת יותר ממילה אחת). לאחר מכן השיטה מחזירה מילון ששומר דירוג ואת כל המסמכים שקיבלו את הדירוג הזה.  
- השיטה ScoreBM25 מקבלת מסמך ועבור כל מילה בשאילתה מוצאת את השדות המתאימים (תדירות המילה במסמך, תדירות המילה במאגר) ומחשבת את הדירוג של המסמך הזה לפי הנוסחה של BM25.  
- השיטה ProximitySearch מקבלת מסמך ומבצעת מכפלה קרטזית (בעזרת פונקציית העזר CartesianProduct) עבור כל מילות השאילתה. לאחר מכן מתבצע חישוב ודירוג המסמך לפי הנוסחה של ProximitySearch כפי שמתואר בהמשך.  
- השיטה CosSim מקבלת מזהה מסמך ומחשבת את הדירוג של המסמך לפי הנוסחה של CosSim. השיטה נעזרת בפונקציה InnerProduct שמחשבת את ווקטור השאילתה. ווקטור השאילתה מחושב כך שכל מילה בשאילתה מקבלת משקל 1 ומילים חשובות מקבלות את המשקל 2.

המחלקה MainWindow-  
**שדות המחלקה:**שדה בוליאני שמציין האם המשתמש בחר הרצה עם או בלי סטמר.  
שדה loadPath של הנתיב ממנו בחר המשתמש לטעון קבצים ושדה savePath של הנתיב אליו המשתמש בחר לשמור את הקבצים.  
מופע של המחלקה PreTextProccess שדרכו מתבצע תהליך האינדוקס או טעינת הקבצים.  
מילון selectedLan ששומר את שפות המסמכים הקיימים במאגר.  
שדה בוליאני SaveToFile שמציין האם המשתמש בחר לשמור את תוצאות השאילתה/ות לקובץ.  
שדה saveToFilePath ששומר את הנתיב של התיקייה בה בחר המשתמש לשמור את קובץ התוצאות.  
שדה postingFolderPath ששומר את הנתיב של קובץ הפוסטינג אותו טען המשתמש.  
שדה dataFolderPath ששומר את הנתיב של הקבצים dictionary, Docs, stop\_words אותם טען המשתמש.  
בנוסף הגדרנו enum למחלקה שנקרא Status שמציין את מצב המנוע. כלומר האם המנוע ביצע אינדוקס של מאגר, או ביצע טעינה קבצים המכילים מידע על מאגר או לא ביצע את אחד מהנ"ל ולכן לא ניתן להריץ שאילתות.

**שיטות המחלקה:**- השיטה GetQueryResults מקבלת מחרוזת queryInput שהיא השאילתה שיש להריץ ומזהה מסוג int? שהוא מזהה השאילתה. השאילתה עוברת תהליך של parse ובנוסף מורדות ממנה מילים כפולות כך שכל מילה תופיע בשאילתה פעם אחת בלבד.

בעזרת השיטה GetAllSynonyms עבור כל מילה בשאילתה אנו מוצאות מילים נוספות שמתאימות לה, וכל פעם מחליפות את אחת המילים האלו במילה אחרת שנמצאה, וכך יוצרות שאילתה חדשה. בנוסף, מחלקה זו גם מסננת מילים חשובות בשאילתה ושמה אותן בHashSet (פירוט בתיאור אלגוריתם החיפוש).  
כל שאילתה חדשה שיצרנו נשלחת לפונקציה SetNewQuery של המחלקה searcher שמוצאת את כל המסמכים הרלוונטים עבור השאילתה ולאחר מכן קוראת לפונקציה RankerRelevantDocs שמדרגת אותם בעזרת ה ranker. לבסוף גם עבור כל מילה חשובה שנמצאה נוצרת שאילתה חדשה המורכבת ממילה זו בלבד וגם עבורה מתבצע חיפוש ודירוג באותו האופן.  
השיטה מחזירה את 50 המסמכים עם הדירוג הגבוה ביותר וגם תכתוב את התוצאות לקובץ במידת הצורך.  
- השיטה GetFinalListResults מקבלת מילון dicResultsFinal השומר עבור כל דירוג את כל המסמכים שקיבלו את הדירוג הזה (המפתח במילון הוא ערך דירוג כלשהו). השיטה שומרת את מפתחות המילון במערך ממוין ולאחר מכן עוברת על מערך זה החל מהדירוג הגבוה ביותר ומוצאת את 50 המסמכים שקיבלו את הדירוג הגבוה ביותר אותם לבסוף היא תחזיר.  
- השיטה NGramComparison שמקבלת 2 מחרוזות של מילים ומספר המייצג את סף הקבלה שיקבע האם המילים דומות או לא. השיטה משתמשת בפונקציית העזר GetNGram שמקבלת מילה ומחזירה hashSet של כל המופעים שנמצאו לה כאשר n=2. לאחר מכן מתבצע חיתוך ואיחוד של ה hashSet שקיבלנו עבור שתי המילים להשוואה, אם החיתוך חלקי האיחוד גדול או שווה לערך הסף שנקבע אז הפונקציה תחזיר true.  
- השיטה QueryInput\_TextChanged נקראת כאשר מתבצע שינוי בתיבת הטקסט בה מזינים שאילתה להרצה. בכל פעם שהמשתמש מקליד טקסט השיטה מפצלת את הטקסט לפי רווחים ואם הטקסט כולל מילה אחת ורווח אחד אז יופיעו על המסך חמשת המילים שהופיעו במאגר הכי הרבה לאחר המילה שהוקלדה.

2. תהליך חיפוש המסמכים:

ראשית, המשתמש רושם שאילתה עליה מתבצעים כמה פעולות לפני שליחתה לSearch:

הפיכת כל השאילתה לאותיות קטנות, הורדת מילים כפולות על ידי פונקציית העזר RemoveDupWords, פרסור השאילתה (עם או בלי stemmer לפי סימון המשתמש) ואז שליחה לפונקציית עזר GetAllSynonyms.   
פונקציה זו יוצרת מילון, synonymsDic, בו כל key הוא מילה בשאילתה וה value זו רשימה של כל המילים הנרדפות למילה שנמצאו דומות לה לפי N-gram.  
עבור כל מילה בשאילתה בודקים את המילים הנרדפות שלה (על ידי שימוש בThesaurus, כפי שהוסבר למעלה), ואז שולחים את שתיהן לפונקציית עזר NGramComparison שמקבלת 2 מחרוזות וערך סף (0.75) שלפיו מחזירים true או false המציין האם המילים דומות לפי N-Gram כאשר N=2. אם הפונקציה החזירה true, אנו מוסיפים את המילה הנרדפת למילון synonymsDic (דוגמא למקרה: לprogram ישנה מילה נרדפת programme שמתווספת למילון).  
בנוסף, אם המשתמש בחר בלי סטמר:  
ממיינים את המילים במילון הראשי, מוצאים את האינדקס של המילה הנוכחית במילון ואז לוקחים את 2 המילים שלפניה ו2 המילים שאחריה. נשווה כל מילה סמוכה עם המילה המקורית בעזרת NGramComparison, וכל מילה סמוכה שההשוואה החזירה עבורה true נוסיף למילון synonymsDic.  
כמו כן, נבצע סטמר על כל מילה סמוכה, ואם היא קיימת במילון הראשי (כלומר הסטמר החזיר מילה אמיתית) נוסיף גם אותה ל synonymsDic.  
הערה: הפונקציה מחשבת תוך כדי את כל ה"מילים החשובות" ומעדכנת את הHashset שנשלח אליה כרפרנס (מילה חשובה הינה מילה שלא מצאנו לה מילים נרדפות). הנחנו שמילים כאלה כנראה מייצגות שם של מקום או שם של אישיות, ולכן זו מילה שהימצאותה במסמכים מייצגת טוב יותר את מה שהמשתמש חיפש.  
לבסוף מחזירים את synonymsDic.

לאחר מכן עבור כל מילה בשאילתה שהיא key במילון synonymsDic, ניצור שאילתות חדשות כאשר כל שאילתה חדשה מורכבת מהמילים בשאילתה המקורית מלבד המילה הkey אותה החלפנו בכל אחת מהמילים הנרדפות מרשימת הvalue שלה. כל שאילתה חדשה כזאת שולחים לSearch.  
הSearch מקבל את השאילתה, מוצא את המסמכים הרלוונטיים כפי שהוסבר למעלה ושולח לRanker לדירוג. באופן כללי כל שאילתה שנשלחת אל הSearch, גם אם המילים בה שונות ממה שהמשתמש בחר, אנו נותנים לה את אותו המשקל כמו השאילתה המקורית.

בRanker קיימים 3 שלבים לדירוג מסמכים, אותם הוא מבצע לכל מסמך שנשלח אליו.

עבור כל מסמך מאתחלים משתנה rank=0, ושולחים לפונקציות הבאות:

**BM25** – ראשית המסמך נשלח אל הפונקציה ScoreBM25, בה מכניסים את הפרמטרים לפי הנוסחה כפי שלמדנו בכיתה, (לאחר ניסוי וטעייה הצבנו את הערכים K1=1.2, b=0.75, k2=1000). כמו כן qfi=1 (משום שהורדנו מילים כפולות), ואת ni, fi הSearch כבר חישב.

הדירוג שמתקבל מBM25 מתווסף למשתנה rank.

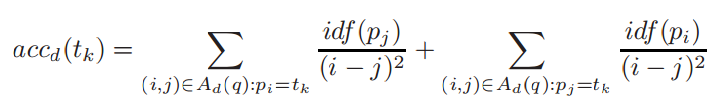
**Cosin Similarty** – לאחר מכן המסמך נשלח לCosin Similarty. לכל מסמך חושב מראש ווקטור המסמך, ולכן נשאר לתת משקל לכל מילה בשאילתה ולהציב בנוסחה. החלטנו לתת משקל של 2 לכל מילה "חשובה" (שנקבעת כפי שצוין למעלה) בשאילתה ומשקל של 1 לכל מילה אחרת.

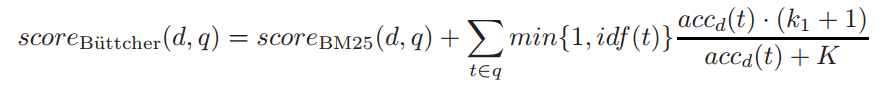
אם המשתמש בחר להריץ שאילתה בלי סטמר אז הכפלנו את הדירוג המתקבל מנוסחה זו ב0.5, אבל אם המשתמש בחר להריץ עם סטמר אז הדירוג מנוסחה זו מוכפל ב1 (כלומר מתקבל ללא שינוי) לבסוף הדירוג מתווסף למשתנה rank. ההכפלה בחצי היא מכיוון שלאחר בדיקה, נוסחת הBM25 מחזירה מסמכים יותר מדויקים כאשר החיפוש מתבצע על מילון וקבצי פוסטינג שעברו סטמר. לכן החלטנו להקטין את ההשפעה של הCosin Similarty במקרה זה.

**Proximity Search** – זהו אלגוריתם שמימשנו לפי מאמר שמצאנו (http://people.mpi-inf.mpg.de/~mtb/pub/proximity-spire07.pdf). האלגוריתם מחשב את ווקטור המיקום של המילים במסמך ונותן דירוג גבוה יותר ככל שהמילים מופיעות קרוב יותר אחת לשנייה. שאילתה תישלח לפונקציה זו רק אם מספר המילים שלה גדול מ1.

האלגוריתם: עבור כל זוג מילים בשאילתה, תתבצע מכפלת קרטזית של המיקומים שלהם וכל זוג שנוצר יתווסף למילון megaDic (Qd) שהkey שלו הוא זוג המילים האלה והvalue הוא רשימת תוצאות ההכפלה.  
בנוסף, יתווסף לHashSet שנקרא locationsSet כל מיקום שהיה מאחת מהמילים.  
ממיינים את הlocationSet ואז עבור כל זוג מיקומים עוקב, בודקים האם קיים זוג כזה בmegaDic, ואם כן מוסיפים אותו למילון חדש finalDic (Ad(q)).

  
לאחר מכן עבור כל מילה בשאילתה, מחשבים acc לפי הנוסחה הבאה:



כאשר tk זו השאילתה, i,j הם המיקומים וidf זו כמות ההופעות של המילה במסמך.  
כפי שניתן לראות מנרמלים את הacc של כל מילה בשאילתה ומחברים לBM25, ולכן הrank מוחזר ומתווסף לrank הקיים ללא נרמול נוסף.

משקלי דירוג ונוסחת הדירוג הסופית-

משקלי הדירוג נקבעו לפי הBM25. ראינו שזוהי נוסחה מאוד חזקה שהחזירה לבדה תוצאות יחסית גבוהות, לכן עם כל נוסחת דירוג חדשה שהוספנו הכפלנו בקבועים שונים כך שהתוצאות מהנוסחה BM25 לא ייפגעו.

3. סיכום:

מעבר למתואר למעלה, ניסינו נוסחאות נוספות:

התחשבות בheader המסמך- מצאנו שההתחשבות במילים המופיעות בכותרת המסמך לא מספיק מייצגות את המסמכים הרלוונטיים לשאילתות. השימוש בנוסחה הזו פגע בתוצאותינו ולכן לא השתמשנו בה.

BM25F- תחילה, עקב טעות בנוסחה לחישוב BM25 קיבלנו תוצאות מאוד נמוכות וחיפשנו חלופה לנוסחה הזו. הגענו לBM25F (אותו חישבנו לפי נוסחה שמצאנו במאמר https://km.aifb.kit.edu/ws/semsearch10/Files/bm25f.pdf) שמתחשב גם בheader המסמך ובעזרתו קיבלנו תוצאות טובות יותר. לאחר גילוי הטעות, מצאנו שהBM25 המקורי עדיף מBM25F ולכן לא השתמשנו בו.

טיפול סמנטי בשאילתה- הטיפול הסמנטי שלנו מתבצע ב GetAllSynonyms, שם לכל מילה אנו מוצאים מילים נוספות שאולי יתאימו לה. תחילה החלפנו כל מילה בכל המילים הנרדפות לה, אך עקב חוסר יכולת לסנן מה רלוונטי ומה לא, התוצאות שלנו נפגעו. לכן הוספנו רק מילים שהן מילים נרדפות ומאותו השורש של המילה שהמשתמש רשם (בעזרת N-Gram).

הערכה:

בלי סטמר

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| precision@30 | precision@15 | precision@5 | Recall | Precision | Query | # |
| 0.2667 | 0.3333 | 0.4000 | 0.1333 | 12/50=0.24 | Space Program | 11 |
| 0.3667 | 0.5333 | 0.4000 | 0.0871 | 20/50=0.4 | Water Pollution | 12 |
| 0.4333 | 0.2000 | 0.2000 | 0.4 | 22/50=0.44 | Genetic Engineering | 82 |
| 0.1667 | 0.1333 | 0.2000 | 0.04 | 13/50=0.26 | International Terrorists | 118 |
| 0.3667 | 0.3333 | 0.4000 | 0.0284 | 23/50=0.46 | Impact of Government Regulated Grain Farming on International | 142 |
| 0.4333 | 0.2000 | 0.2000 | 0.0325 | 19/50=0.38 | Real Motives for Murder | 189 |
| 0.1333 | 0.0667 | 0 | 0.476 | 10/50=0.2 | Airport Security | 341 |
| 0.7333 | 0.7333 | 1.0000 | 0.4933 | 37/50=0.74 | Wildlife Extinction | 347 |
| 0.1000 | 0.1333 | 0.2000 | 0.1382 | 13/50=0.26 | piracy | 367 |
| 0.1333 | 0.1333 | 0 | 0.3181 | 7/50=0.14 | Nobel prize winners | 374 |
| 0.2667 | 0.2667 | 0 | 0.2333 | 14/50=0.28 | oceanographic vessels | 399 |
| 0.2333 | 0.2000 | 0.2000 | 0.8947 | 17/50=0.34 | Schengen agreement | 410 |
| 0.4333 | 0.4000 | 0.6000 | 0.7142 | 20/50=0.4 | Three Gorges Project | 416 |
| 0.5667 | 0.6000 | 1.0000 | 0.6667 | 30/50=0.6 | robotic technology | 431 |
| 0.4000 | 0.2000 | 0 | 0.1639 | 30/50=0.6 | King Hussein, peace | 450 |

Map: 0.1366

עם סטמר

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| precision@30 | precision@15 | precision@5 | Recall | Precision | Query | # |
| 0.3000 | 0.3333 | 0.4000 | 0.1304 | 12/50=0.24 | Space Program | 11 |
| 0.5333 | 0.6667 | 0.2000 | 0.1052 | 24/50=0.48 | Water Pollution | 12 |
| 0.4667 | 0.2000 | 0.2000 | 0.4 | 22/50=0.44 | Genetic Engineering | 82 |
| 0.4333 | 0.4667 | 0.6000 | 0.0771 | 25/50=0.5 | International Terrorists | 118 |
| 0.6667 | 0.6667 | 1.0000 | 0.0457 | 37/50=0.74 | Impact of Government Regulated Grain Farming on International | 142 |
| 0.4000 | 0.2000 | 0.2000 | 0.0273 | 16/50=0.32 | Real Motives for Murder | 189 |
| 0.1333 | 0.0000 | 0.0000 | 0.4761 | 10/50=0.2 | Airport Security | 341 |
| 0.6667 | 0.7333 | 1.0000 | 0.48 | 36/50=0.72 | Wildlife Extinction | 347 |
| 0.1000 | 0.1333 | 0.2000 | 0.1382 | 13/50=0.26 | piracy | 367 |
| 0.1333 | 0.1333 | 0.0000 | 0.3181 | 7/50=0.14 | Nobel prize winners | 374 |
| 0.2000 | 0.2667 | 0.0000 | 0.2 | 12/50=0.24 | oceanographic vessels | 399 |
| 0.3000 | 0.2667 | 0.2000 | 0.9473 | 18/50=0.36 | Schengen agreement | 410 |
| 0.4333 | 0.4000 | 0.4000 | 0.6785 | 19/50=0.38 | Three Gorges Project | 416 |
| 0.5333 | 0.6000 | 0.8000 | 0.6444 | 29/50=0.58 | robotic technology | 431 |
| 0.1667 | 0.0667 | 0.0000 | 0.0710 | 13/50=0.26 | King Hussein, peace | 450 |

Map=0.1294