דו"ח פרויקט:

יואב זלינגר	208752667	yoavzel@post.bgu.ac.il
יצחק אלבזיס	207181603	elbazis@post.bgu.ac.il

GitHub	GCP Bucket	Index Files

תיאור הליך העבודה:

בתחילת העבודה הקדשנו זמן רב בניסיון לתכנן את הליך העבודה בצורה יעילה. החלטנו לחלק את העבודה למספר חלקים:

- תכנון בניית המחלקות.
- בחירת פונקציות משקל ודמיון.
- בניית מבני Inverted Index שהתחלק לשלבים:
  - בחירת תוכן כל ערך ב posting (בכל posting list של טוקן מסוים).
  - ניסוי המבנים תחת חלק קטן מהקורפוס.
  - בניית האינדקסים ואחסונם (תחת ה Bucket ב GCP). עבור כלל הקורפוס.
- בדיקת המחלקות ו"שאיבת" מידע מה GCP תוך הרצת שאילתות התחלתיות לדוגמה.
- שיפור זמני הריצה והפיכת הקוד לקוד מקבילי.
- בניית ה Frontend ב Google Colab.
- בדיקת תוצאות מול שאילתות לדוגמה ושיפור פרמטרים ומשקלים של פונקציות הדירוג וכן הרצת בדיקות חוזרות על השאילתות לדוגמה. ובניית Frontend ב GCP.

<u>תכנון בניית המחלקות:</u>

חשבנו כיצד יהיה חכם לחלק את הקוד, כך שכל תחזוקה של הקוד תצריך שינוי קטן ונקודתי (ללא שכפולי קוד וכדומה). הגענו לחלוקה של כל העבודה למספר מחלקות. הבנו שאת רוב המחלקות נצטרך לשני שלבי הפרויקט – לחלק העיבוד של הקורפוס ולחלק הרצת השאילתות. לכן הכנו את המחלקות בקובץ עזר, כך שנוכל לשאוב את המחלקות המתאימות בכל חלק משלבי הפרויקט.

<u>בחירת פונקציות דמיון ומשקל:</u>

עברנו על החומר הלימודי וחיפשנו אילו פונקציות משקל ודמיון עומדות לראשותנו. לבסוף החלטנו להשתמש ב Cosine Similarity תוך שימוש ב TF-IDF. השימוש בפונקציית משקל זו נותן יתרון לטוקנים יותר נדירים אול מול יותר שכיחים, וכן מנורמל ביחס לגודל המסמך. בנוסף ה Cosine Similarity חסום וכן לא מצריך זמן רב של חישוב (יחסי) עבור כל שאילתא.

<u>בניית Inverted Index:</u>

תחילה ניסינו אילו מבני Inverted Index אנו צריכים לשמור. מעתה ועד סוף המסמך נשתמש במושג Segment עבור אחד משלושת חלקי המסמך – Title, Body ו Anchor Text.

החלטנו לשמור Inverted Index עבור כל סגמנט. כדי לחסוך זמן בעת קבלת שאילתות, בזמן העיבוד (Preprocessing) החלטנו כי אפשר לשמור עבור טוקן i במסמך j בשדה ה " tf_{ij} " של ה Inverted Index את החלק הרלוונטי אליו בפונקציית הדמיון. כלומר : $\frac{w_{ij}}{|d_j|} = \frac{tf_{ij} * idf_i}{|d_j|} = \frac{\frac{f_{ij}}{\#terms\ in\ doc\ j} * \log_{10}\left(\frac{N}{df_j}\right)}{|d_j|}$, כאשר N זה גודל הקורפוס. בנוסף, שמנו לב כי אפשר לחסוך זמן וליעיל את חישוב הדמיון בין מסמכים לשאילתות, בכך שלא נחלק את החישוב בגודל השאילתא, משום שהוא אחיד לכלל ה"דמיונות" והתעלמות ממנו תשמור על סדרי הגודל. לכן, מה שישאר לעשות בעת קבלת שאילתא זה הכפלה w_{iq} עבור כל טוקן בשאילתה (לא צריך לחלק בגודל השאילתה כי זה יהיה קבוע לכל המסמכים ולא ישפיע על יחסי הגודל).התחלנו בבנייה. כדי שנוכל לעקוב ולדבג באופן רציף התחלנו בבנייה על קובץ parquet יחיד. שמנו לב כי אפשר להשתמש בצורה נוחה בדברים שכתבנו בעבודות הקודמות ולכן עבדנו בצמוד אליהן.

אופן הבנייה עצמו כלל מספר שלבים :

- טוקניזינג וסטמינג, תוך הורדת stop words. בשלב זה השגנו עבור כל מסמך בכל סגמנט את "הכמויות" של כל טוקן שהשארנו. בשלב זה כבר שמנו בשדה ה"כמות" את החישוב שהצגנו מעלה, פרט ל idf שאנו עוד לא יכולים לדעת בשלב זה.
- יצירת posting list ממוין לכל טוקן.
- חישוב df של כל מסמך.
- כעת היה בידנו את כל המידע הרלוונטי לחישוב ה idf של כל token. לכן הכפלנו כל ערך ב postings ב idf.
- כתיבת כל Inverted Index לזיכרון. בשלב זה נתקלנו בבעיה – הערך של כל posting שאנו מעוניינים לשמור הפך ממספר שלם למספר עשרוני. כדי להתגבר על זה הוספנו שלב בכתיבה ובקריאה לזיכרון – בעת הכנסה לזיכרון הכפלנו ב10,000 ועיגלנו (כך שישאר בגבולות ה 16 ביטים), בהוצאה עשינו את התהליך ההפוך (חילקנו ב 10,000).

<u>בדיקת המחלקות :</u>

כעת היה ברשותנו את כל המידע כדי להתחיל לבצע שאילתות. לפני שבדקנו את כלל המנוע עם ה frontend רצינו לבדוק התכנות שאיבת המידע ששמרנו. בנינו את התהליך בשלבים – קריאת posting list מתאים מהבאקט, לאחר מכן החזרת כל המסמכים הרלוונטיים לשאילתא מסוימת. לאחר מכן התקדמנו לבניית פונקציית חיפוש של ממש – מקבלת שאילתא – מחפשת את ה posting lists המתאימים, מדרגת לפי רלוונטיות ומחזירה את הרלוונטיים ביותר.

בשלב זה נתקלנו במספר בעיות, שמנו לב שהבדיקות הראשוניות שלנו על הקורפוס הקטן לא היו מקיפות. שמנו לב כי מסמכים רלוונטיים רבים לא חוזרים – לכן חשבנו על הוספת page rank ו page views לדירוג המסמך. בין הבעיות היה גם stop words נוספות שהיינו צריכים להוסיף. בנוסף שמנו לב כי ביצענו Tokenizing בשלבים לא נכונים – תחילה ביצענו stemming ולאחר מכן בדקנו התאמה ל stop-words, מה שגרם לפספוס מילים רבות. בנוסף שמנו לב כי ה Regex בו השתמשנו מתעלם מהרבה מילים שלא התכוונו להתעלם מהם. לכן חזרנו לעבודה על שלב Tokenizing ובנינו Inverted Index-ים חדשים. בעיה נוספת הייתה שקיבלנו מסמכים רבים שלא רלוונטיים. הצלחנו לאתר כי סינון מסמכים אשר ב Body

המסמך לא מופיעות כלל מילי השאילתא הניב תוצאות טובות (במידה ולא היו מספיק מסמכים כאלו אז כן לקחנו בחשבון מסמכים עם פחות מכיסוי מלא). לשלב זה לא היינו צריכים לבנות Inverted-Index חדש משום שהמידע הנ"ל קיים באובייקט שבנינו – מסמכים אלו הם המסמכים שמופיעים בכל ה posting lists של מילות השאילתה.
לאחר מכן הרצנו שוב וקיבלנו תוצאות יותר טובות אז המשכנו לשלב הבא.

<u>שיפור זמני הריצה:</u>

כדי לשפר את זמני הריצה רצינו להפוך את הקוד למקבילי. שמנו לב כי סריקות בכל אחד ממבני הנתונים אינו תלוי באחרים, ולכן ביצענו את הסריקות במקביל. בנוסף רצינו לעבוד על מיון המסמכים הרלוונטיים אשר מוחזרים, ולכך השתמשנו בערימה. ביצוע פעולות אלו שיפר משמעותית את זמני הריצה.

<u>בניית Frontend ב Colab:</u>

כעת שכבר היינו יותר בטוחים במודל האחזור שלנו אז רצינו להשלים את שלב ה frontend. התחלנו בבניית ה frontend של ה Colab כדי להבין אם אנו מסוגלים להחזיר תשובות לשאילתות מרחוק. לאחר שראינו שזה עובד הרצנו בדיקות נוספות על השאילתות לדוגמה שקיבלנו.

<u>בדיקות נוספות ובניית Frontend ב GCP:</u>

פה ראינו כי אנו מקבלים תוצאות נמוכות ורצינו לשפר אותן. כדי לעשות זאת הלכנו חשבנו לבדוק את משקל כל סגמנט בדירוג הסופי שהמסמך מקבל – את כלל הדירוגים שמסמך קיבל בכל הסגמנטים סיכמנו כממוצע משוקלל שלהם. כל סגמנט קיבל משקל ורצינו לבדוק איזו קומבינציה של משקלים תניב את הדירוגים הטובים ביותר. דבר נוסף שרצינו לבדוק זה את גורם ה PageRank ו pageviews. חששנו שהם מקבלים משקל רב מדי ולכן בדקנו את נרמולם ע"י לוג בבסיסים משתנים. כדי לעשות זאת הרצנו לולאה שתנסה כל קומבינציה ותחזיר את התוצאה האופטימלית. כשהגענו לתוצאה שנראתה לנו מספיק טוב (טובה אך לא טובה מדי כדי להיזהר מ Overfitting). המשכנו איתה לשלב הבא. התחלנו בבניית VM ב GCP שיריץ את מנוע החיפוש שלנו. כשסיימנו אז שלחנו לו שאילתות ובדקנו את התשובות שהוא מחזיר.

ביצועים מיטביים והלוקים בחסר:

- <u>ביצועים מיטביים:</u>
  השאילתה שהביאה את הביצועים המיטביים היא:

  Neuroscience

  התוצאות הרלוונטיות ביותר שהמודל החזיר הן:

```
Neuroscience
Cognitive neuroscience
Computational neuroscience
Behavioral neuroscience
Neuroscience of free will
Neuroscience of religion
Affective neuroscience
The Journal of Neuroscience
Society for Neuroscience
Neuroscience of music
```

מבחינת הזמן, זו שאילתה קצרה ולכן לא צריך לחשב להסתכל על הרבה posting lists וכן זו מילה בעלת נישה ספציפית כך שכמות המסמכים החוזרים ממנה נמוכה. בנוסף בחירת בסיס השאילתות כגוף הטקסט עובד כאן טוב משום שהשאילתה היא בעלת מילה אחת. משום שהמילה קשורה לתחום ספציפי אז זה מקל על המודל לאחזר תוצאות שאכן רלוונטיות.

- <u>ביצועים הנמוכים ביותר</u>:

השאילתה שהביאה את הביצועים המיטביים היא:

Who is considered the "Father of the United States"?

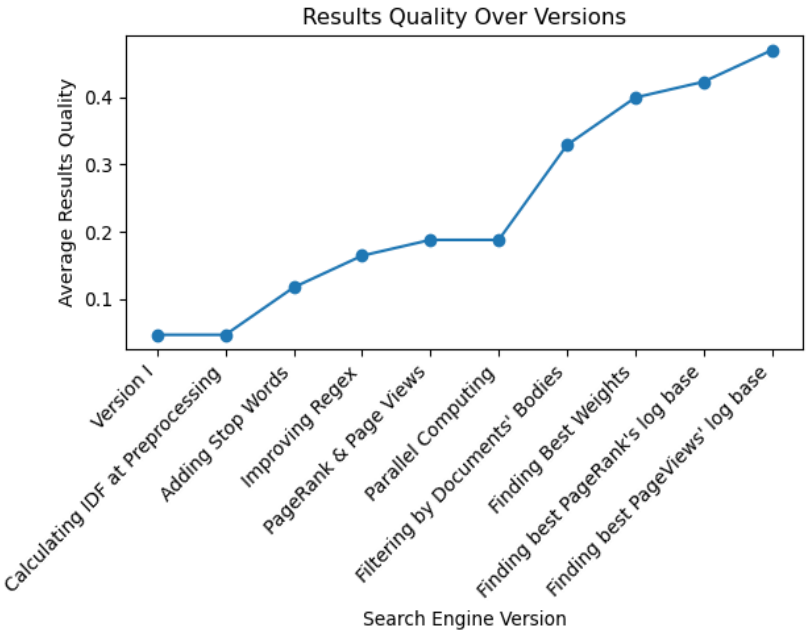
התוצאות הרלוונטיות ביותר שהמודל החזיר הן:

```
Father
Church Fathers
God the Father
Larry Page
President of the United States
United States Navy
Father of the Nation
Father Ted
United States Congress
Founding Fathers of the United States
```

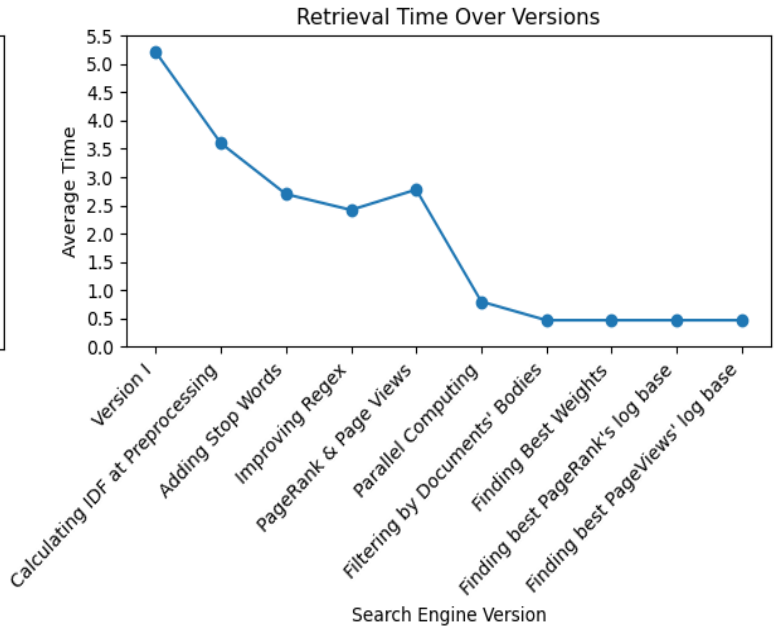
מבחינת זמן, השאילתה מאוד ארוכה ובה טוקנים רבים, דבר המאריך את זמן הביצוע. מבחינת איכות, המילה Father מקשה על מציאת תוצאות טובות. המילה היא בעלת סמנטיקה הקרובה למילים הקשורות לתוצאות הרלוונטיות אבל מנוע האחזור שלנו לא מבצע LSI או שיטות אחרות להתאמה סמנטית, כך ששאילתות כאלו פוגעות באיכות האחזור. בנוסף, במסמך מילים אשר קיימות במסמכים הכי פופולריים – United States. למסמכים הקשורים לארצות הברית הרבה קישוריות (pagerank גבוה) והרבה צפיות (pageviews גבוה). מה שמקשה על המודל להחזיר תוצאות ממוקדות.

ביצועי המודל לאורך הפרויקט:

<u>איכות תוצאות האחזור</u>



<u>זמני האחזור</u>



נספח א' : קבצים

כל קבצי הפיקל העדכניים נמצאים בתיקיית global_pickles וקבצי ה bin תחת newWrites

קבצים עיקריים :

- Inverted Index לכל הסגמנטים :
  - Inverted Index – title_inverted_index לכותרות.
  - Inverted Index – body_inverted_index לגוף המסמכים.
  - Inverted Index – anchor_inverted_index לגוף ל anchor text של המסמכים.
- stop_words – סט של מילות ה stop_words אותן אנו מסננים.
- מילונים גלובליים עם מיפויים בין doc_id לערכים מסוים :
  - title_dict – מיפוי לכותרות.
  - page_rank_dict – מיפוי לערך ה pagerank.
  - page_views_dict – מיפוי לערך ה pageviews.

נספח ב׳: משתנים ומחלקות

כלל המחלקות נמצאות תחת הקובץ העזר backend_helper.py

משתנים גלובליים עיקריים:

- PICKLES_FOLDER – תיקיית הפיקלים בבאקט.
- SEGMENTS_WEIGHTS_DICT – מילון משקלים של כל סגמנט.
- PAGE_RANK_LOG_BASE – בסיס ללוג של ערך ה pagerank.
- PAGE_VIEWS_LOG_BASE – בסיס ללוג של ערך ה pageviews.

מחלקות עיקריות:

- BucketManager – אחראי על כלל התקשורת מול הבאקט. בעל שלוש פונקציות עיקריות:
  1. get_bucket: משיג instance של הבאקט.
  2. store_pickle: מקבל אובייקט ושומר אותו בתיקיית פיקלים בבאקט.
  3. load_pickle: מקבל שם של פיקל ומחזיר את האובייקט התואם בתיקיית הפיקלים בבאקט.
- InvertedIndexManager – אחראי על תקשורת עם מבני ה InvertedIndex השונים. בעל שתי פונקציות עיקריות:
  1. retrieve_inverted_indicies – טוען את כלל מבני ה InvertedIndex.
  2. get_posting_list – מקבל שם של סגמנט וטוקן ומחזיר את ה posting list של הטוקן תחת הסגמנט הזה.
- PageManager – אחראי על תקשורת עם המילונים הגלובליים. בעלת שתי פונקציות עיקריות:
  1. get_number_of_pages – מחזיר את גודל הקורפוס (משמש לחישוב idf של כל טוקן בשאילתה).
  2. get_page_item – מקבל doc_id ושם של ערך אותו אנו רוצים לקבל.
- Tokeizer – אחראי על ביצוע token לטקסט (לשאילתה ולסגמנטים השונים של המסמכים). למחלקה שלוש פונקציות עיקריות:
  1. store_stop_words – מקבל רשימת stop words אותם נרצה להוסיף לסט ה stop_words שלנו.
  2. load_stop_words – טוען את כל ה stop_words ששמורים בבאקט.
  3. Tokenize – מקבל טקסט ומבצע עליו tokenize – מוריד stop_words, מבצע stemming ומחזיר מונים של כל טוקן ייחודי.
- BackendSearch – אחראי על קבלת שאילתות והחזרת מסמכים רלוונטיים אליהן. למחלקה שלוש פונקציות עיקריות:
  1. process_query – מקבל שאילתה ומחזיר אותה לאחר ביצוע tokenizing.
  2. segment_search – מקבל טוקני שאילתה ומחזיר מילון מסמכים רלוונטיים לסגמנט.
  3. search – מאגד את הכל: מקבל שאילתה ומחזיר את המסמכים הכי רלוונטיים אליה בצורה ממוינת.