Saji Assi 314831207
Mohammad Arabi 206985533

דוו"ח
תרגיל בית 1

שלב 1

1. שליפת נתונים מתוך שמות קבצי הפרוטוקולים : הסעיף הזה קל לממש אותו השתמשו בפונקציה def extracting_data_protocol_file(file_name) שמקבלת שם הקובץ ואז משתמשים ב-split. לפי ב-'_' ואז מזה נקבל מספר הכנסת וסוג הפרוטוקול.

2. שליפת מספר פרוטוקול : לכל אחד מסוגי הפרוטוקולים השתמשנו במנגנון אחר של מימוש, לסוג ptv השתמשנו בפונקציה def ptv_protocol_number(document) שמקבל מסמך ואז מחפשים בהמסמך את התבנית: פרוטוקול מס' ואז ניקח המספר שאחרי התבנית הזאת ובמקרה שאין מספר נחזיר 1-, ובמקרה שיש יותר ממספר לפרוטוקול לקחנו את כל המספרים של הפרוטוקול הזה.
לפרוטוקולים מסוג ptm השתמשנו בפונקציה def ptm_protocol_number(document) כך נחפש התבנית 'הישיבה ה' ואז החלק מהטקסט אחרי התסנית הזאת עד 'של הכנסת' נשלח לפונקציית עזר def word_to_number(arr_word_num) שמחשבת המספר של מילה (המספר שבמילים) כך בשביל החישוב הגדרנו מילון בשם doc_word_number שהגדרנו בתוכו הסמפרים במילים הבסיסיים כמפתח והערך המתאים הוא המספר.

3. שליפת טקסט בעל תוכן: ממשנו פונקציה def fetching_text_with_content(document)
בהתחלה היה צריך לחפש מאיפה המדברים מתחילים לדבר כלומר איפה המדבר הראשון (בוועדה וגם במליאה) שזה לרוב מתחיל בדיבור של ה-היו"ר אז עבור המסמך התחלנו לחפש הפסקה שיש בה מילת היו"ר או << יור >> עם זה שהמשפט מסתיים ב-: וטבלנו בעוד מקרים שזה לא מתחיל ב-היו"ר או << יור >> כמו באחד הפרוטוקולים מתחיל ב-מר ועוד כמה מקרים כאילו שטבלנו ספיציפי כלומר כתבנו פקודה שתבדוק מתי הפסקה שיופיע בה השם הספיציפי הזה של המדבר הראשון בפרוטוקול, אחרי שנמצא הדובר הראשון נתחיל לקחת הטקסטים שהוא דבר (שהם הפסקאות מתחת לשם שלו) עד הדובר הבא עם להסיר הדברים שבין שני המדברים כמו ההצבעה (נבדוק בכל פעם אם הטקסט בפסקה הוא "הצבעה" ואז מהפסקה הזאת עד הדובר הבא לא ניקח) וגם להסיר הכותרות דרך בדיקה אם הפסקה היא באמצע (paragraph.alignment == 1), כך אחרי שנגיע לפסקה שהיא שם של הדובר (פסקה קטנה בגודל עם ה-: בסוף << יור >> או << אורח >> או << דובר >> או << דובר_המשך >> : או שזה מתחיל באחד מאילו
וגם יש כמה מקרים אחרים שטבלנו ספיציפי) כל מה תחת השם של הדובר עד הדובר הבא נשייך לדובר הנוכחי.
כשנגיע לפסקה של שם דובר נקינו כל מה שהוא לא השם כמו : מה בין הסוגריים (כמו באיזו מפלגה שייך) או אם זה מתחיל בתפקיד כמו שר ה... או מזכיר הכנסת או ראש הממשלה וכל האפשריות של תפקידים שיכול להיות בפרוטוקול הדבר הזה טבלנו גם ספיציפי כלומר שמנו ספיציפי שורות קוד שבודקות אם זה מתחיל בתפקיד מסוים כדי לנקות אותו (נסינו לכלול כל התפקידים שניתן למצוא בפרוטוקול).
בכל פעם נגיע למדבר הבא נכניס לרשימה את שם הדובר הקודם ופסקאות שהוא אמר ואז בסוף הפונקציה תחזיר רשימה כך בכל תא יש שם דובר מה והטקסטים שהוא אמר (לא כולם ביחד אלא מה אמר עד הדובר הבא).

==> השימוש בשמות כמו שיופיעו בפרוטוקול בלי ניקיון יכול לעשות בעיות כמו בעיה לזהות שזה אותו דובר כי בלי ניקוי השם של מדבר יכול להופיע בצורות שונות כמו משל פעם יופיע השם שלו עם התפקיד ופעם אחרת יופיע השם שלו עם המפלגה שהוא שייך לה, ועוד מופיעות שונות של השם בצורות שונות ואז בלי ניקיון לא ניתן לזהות שזה אותו בן אדם, כלומר למרות שהוא אותו בן אדם נחשב כבן אדם אחר, ואז גם לא ניתן ולדעת כל המשפטים שדובר אמר אם רצינו את זה. לכן ניקוי כל התוספות להישאר רק עם השם של הדובר שפתור הבעיה הזאת ויעור לנו לעשות יותר בדיקות אם נרצה.

4. חלוקה למשפטים: ממשנו פונקציה def division_into_sentences(sentences_list)
הפונקציה מקבלת הרשימה מהסעיף הקודם ואז נעבור על הרשימה ועבור כל תא נעבור על הטקסטים שבתא הזה, החלוקה הייתה דרך עבירה על הטקסט מההתחלה עד שנגיע לנקודה או ; או הפסקה תסתיים ואז זה נחשב משפט, ואז ישר נכניס המשפט הזה עם שם הדובר לתא חדש ברשימה. כלומר אחרי שנגיע ל נקודה או ; בפעם הבאה נתחיל לחפש המשפט הבא אחרי הנקודה או ה-; הזאת, אם הפסקה הסתיימה כל מה קראנו יהיה משפט. היה צריך לטפל גם במקרים מסוימים כמו משל : הנקודה היא נקודה מספרית כמו 4.5 או הנקודה בטקסט בין סוגריים או גרשיים שלא נחשב סיום משפט.

5. נקיון המשפטים: ממשנו פונקציה def clean_sentences(sentences_list)
הפונקציה מקבלת הרשימה של המשפטים מהסעיף הקודם ואז עוברים על כל משפט ובודקים אם יש בתוך המשפט "- - -" או "- -" או שיש מילים באנגלית או אין מילים בעברית במקרים אילו לא נכניס המשפט בחזרה לרשימה חדשה שרוצים להחזיר, כלומר במקרים האילו נזרוק המשפט ולא ניקח, גם מחקנו מהמשפטים את המופע של - בין שתי מילים במקרה יש רווח לפני ואחרי ה - , בשביל לעשות זה חפשנו בכל פעם אם יש מופע של - ואז נמזוג הטקסט לפני ואחרי המופע הזה.

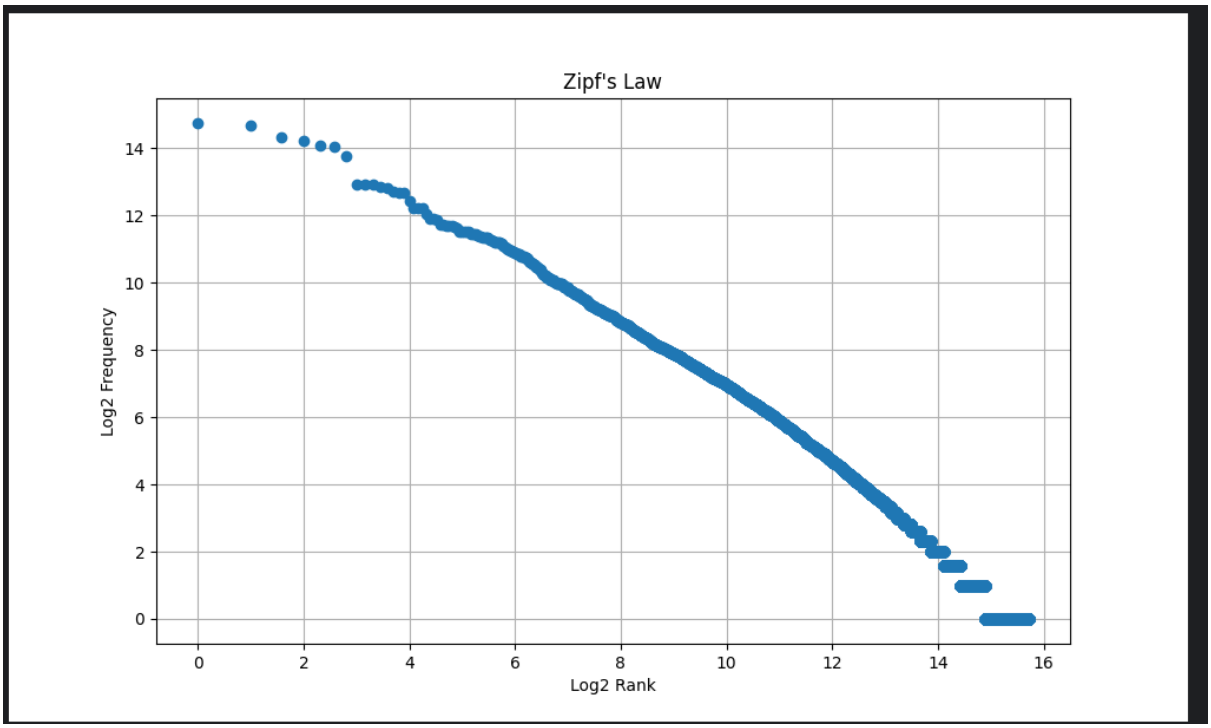
6. טוקניזציה: ממשנו הפונקציה def tokenization(sentences_list)
הפונקציה מקבלת הרשימה של המשפטים אחרי הנקיון מהסעיף הקודם, עבור כל משפט כדי לחלק לטוקנים היינו עוברים על המשפט ואז מחפשים רווח ואז זה יהיה טוקן ואז נמשיך לחפש הרווח הבא מהמקום אחרי הטוקן הקודם וככה, כלומר לחלק לפי הרווחים, בנוסף לבדיקת הרווח בדקנו גם אם הטוקן הזה מתחיל או מסתיים בתו שהוא לא אות כמו משל : "ביחד" יש בהתחלה " שלקחנו כתוקן לבד וה-" שבסוף לבד ומילת ביחד כטוקן לבד גם טבלנו במקרים שיש יותר מתו בסוף או בהתחלה כמו משל הדוגמא הזאת "ביחד". בסוף יש יותר מתו לא אות אז לקחנו את ה-" שבהתחלה כטוקן וביחד כטוקן וב-" בסוף טוקן והנקודה בסוף כטוקן, כלומר גם כל תו שלא אות כמו : . , ? (ועוד לקחנו כטוקן. הפונקציה תחזיר רשימה של טוקנים כך בכל תא יש טוקנים של משפט.

7. שליפת משפטים איתם ניתן לעבוד:ממשנו הפונקציה def sentences_atleast4_tokens(tokens_list,sentences_list)
הפונקציה תקבל רשימת הטוקנים מסעיף הקודם ורשימת המשפטים מהסעיף 5 כך נעבור על רשימת הטוקנים במקרה שיש בתוכה לפחות 4 טוקנים נכניס הטוקן הזה לרשימת טוקן חדשה (טוקנים של המשפטים שאיתם ניתן לעבוד) וגם במקביל ניקח המשפט המתאים לטוקנים האילו (כלומר נכניס המשפט לרשימה חדשה של משפטים שאיתם ניתן לעבוד), כך נשים לב הטוקנים בתא ה-i ברשימת הטוקנים tokens_list שהפונקציה תקבל מתאים (זה הטוקנים שלו) להמשפט בתא ה-i ברשימת המשפטים sentences_list שהפונקציה תקבל. ואז הפונקציה תחזיר רשימת המשפטים שאיתם ניתן לעבוד עם רשימת הטוקנים שלהם.

שלב 2

1. הקובץ מצורף
2. הגרף מייצג יחס בין ה RANK ו FREQUENCY כאשר RANK הוא דירוג המילה והFREQUANCY הוא תדירות המילה. ואפשר לראות שהגרף מתנהג כמו שלמדנו לגבי חוק ZIPF
3. לא באמת אנחנו יודעים שצריך להיות כמו שלמדנו לינארי באמצע ובקצוות לא. אבל קיבלנו משהו אחר בגלל שגיאה שלא הצלחנו לטפל בה.
4. לפי החוק הוא לא אמור להשתנות, וזה בגלל המילים הלא נפוצות ישארו לא נפוצות והמילים הנפוצות ישארו כך.

.5




```
most common words: ['תא', 'אל', 'ינא', 'לש', 'לע', 'הד', 'תסנכה', 'יש', 'ונחנא', 'אוה']
least common words: ['וטלקוי', 'ויגיצנמ', 'רדמ', 'הארנכשו', 'ילגרהל', 'תודומ', 'תוחתפתהו', 'עצובו', 'ברמהו', 'ותקולחו']
```

.6