**דו"ח תרגיל בית 2**

מודלי שפה – n-grams

**שלב 1**

**שלב 1.1:**

הפונקציה ' calculate\_prob\_of\_sentence' מומשה. היא מקבלת כקלט משפט, ומחזירה את לוג ההסתברות של המשפט. הפונקציה מחשבת את לוג ההסתברות בעזרת נוסחת ה- Trigram מההרצאה, כאשר ההסתברויות מחושבות באמצעות MLE עם החלקת לפלס כנדרש.

בהינתן המשפט , הפונקציה קודם כל מוסיפה למשפט שני טוקני דמה , כך שהמשפט הופך להיות , ומחשבת את ההסתברות שלה באמצעות הנוסחה הבאה:

כאשר:

כך ש-

המקדמים נבחרו להיות 0.9, 0.0999, 0.0001 בהתאמה. המקדמים נבחרו על סמך ניסוי שהשווה בין שלשות שונות של מקדמים באחוז הצלחתם בניבוי טוקן חסר במשפט. הניסוי כלל 500 משפטים שנבחרו באופן אקראי מקורפוס הוועדות, ובכל אחד מהם 10% מהטוקנים מוסכו. הניבוי של הטוקנים הממוסכים התבצע ע"י הפונקציה 'generate\_next\_token' שמסתמכת בניבוי שלה על הסתברות המשפט שמחושב ע"י הקריאה ל- ' calculate\_prob\_of\_sentence'. הצעד הזה של בחירת המקדמים נעשה בסוף (בהתחלה השתמשנו במקדמים שרירותיים), ובחירת המקדמים שישתתפו בניסוי נעשתה על סמך אבחנות שהתקבלו במהלך כתיבת הקוד, בנוסף ל- "מיני" ניסויים שנעשו במטרה ללמוד את התרומה של המקדמים ואיך הגדלת\הקטנת כל אחד מהם משפיעה על התנהגות פונקציית הניבוי. שני הדברים הנ"ל צמצמו את טווח האופציות של כל מקדם. ניתן לסכם את האבחנות בכך שהמקדם של ה- unigrams צריך להיות קטן מ- 0.05, ושהמקדמים (החל ממקדם ה- trigrams) צריכים להיות גדולים אחד מהשני בהפרש משמעותי. האבחנה הראשונה התקבלה בשל כך ששמנו לב שמקדם גבוה עבור ה- unigrams נתן משקל יתר לטוקנים נפוצים כמו "," ו- "אני", גם כשהמיקום שלהם לא תואם מבחינת תחביר ושפה, ולא מבחינת הסתברות של משפט (למשל, האלגוריתם ניבא את הטוקן "," אחרי הטוקן "עמיר" למרות שהופעת הטוקן "פרץ" אחרי "עמיר" יותר שכיחה בקורפוס). לגבי האבחנה השנייה, היה אפשר לשים לב לכך שיכולת הניבוי של ה- bigrams יותר טובה ביחס ל- unigrams ושל ה- trigrams יותר טובה משניהם. המשקל הגבוה של ה- bigrams יחסית ל- unigram ניתן על סמך כך שהרבה צירופי טוקנים באים דווקא כזוגות, דוגמת: "חוות דעת", "אמר :", ושמות משפחות של אנשים מפורסמים שבאים בעקבות השמות הפרטיים שלהם. מקדם גבוה ל- bigrams נתן לאלגוריתם להצליח לנבא את המילה השנייה בצירופים מסוג זה. מצד שני, מקדם גדול מדי ל- bigrams פגע בצירופים הפחות נפוצים, במיוחד בצירופים בהם הטוקן השני פחות מזווג עם הטוקן הראשון מאשר טוקן אחר. למשל, בשל שכיחות הצירוף "במגזר הערבי" בקורפוס, האלגוריתם נכשל בלנבא את המילה "הפרטי" כהמשך למשפט "והן עם העובדים במגזר [\*]", מכאן קיבלנו מסקנה שקונטקסט יותר גדול נדרש על מנת להבדיל בין שני המקרים ("במגזר הערבי" ו- "במגזר הפרטי"). בנוסף לכך, ובשל העובדה שמספר המופעים של trigrams קטן משמעותית ממספר ה- bigrams (למשל, "העובדים במגזר הפרטי" לעומת "במגזר הפרטי"), הגענו למסקנה שהמשקל של ה- trigrams חשוב שיהיה גבוה משמעותית מה- bigrams.

**שלב 1.2:**

הפונקציה ' generate\_next\_token' מומשה. היא מקבלת כקלט מחרוזת (רצף של טוקנים), ומחזירה tuple בעל שני אלמנטים: הטוקן בעל ההסתברות הכי גבוהה שיהיה ההמשך של המחרוזת, ולוג ההסתברות שלו. לשם כך, הפונקציה עוברת על כל הטוקנים השונים (או באופן יותר מדויק, על ה- unigrams, מכיוון ש- unigrams וטוקנים שקולים במקרה שלנו), מוסיפה את הטוקן למחרוזת ושולחת את התוצאה של החיבור לפונקציה 'calculate\_prob\_of\_sentence' שבתורה מחזירה את לוג ההסתברות של המשפט. הפונקציה תחזיר את הטוקן שהתקבל עבורו לוג ההסתברות הגבוה ביותר ביחד עם לוג ההסתברות.

למרות שהקפדנו על מימוש שתי הפונקציות 'generate\_next\_token' ו- 'calculate\_prob\_of\_sentence' באופן האופטימלי, בנוסף למאמץ שהושקע בבחירת המקדמים, הפונקציה כשלה הרבה פעמים בניבוי הטוקן בצורה נכונה. מקרים בולטים בהם הפונקציה נכשלת בניבוי הם בטוקנים שהם חלק מקולקציות פחות נפוצות, במיוחד כאשר יש טוקנים יותר שכיחים כהמשך של הקולקציה. דוגמה לכך הוא כישלון האלגוריתם בניבוי הטוקן "סעד" כחסר במשפט "בבקשה , חבר הכנסת אחמד [\*] . ", בשל כך שהקולקציה "אחמד טיבי" יותר שכיחה. מכיוון שאחמד טיבי גם כן חבר כנסת, גם קונטקסט יותר גדול כמו ה- trigram (במקרה הזה הוא "הכנסת אחמד [\*]") לא עזר. סוג מקרים אחר הוא כאשר הטוקנים שלפני הטוקן הנדרש להשלים לא נותנים קונטקסט כלשהו, בחלק מהמקרים הללו, דווקא הטוקנים שבאים אחרי הם אלה שכן היו יכולים לעזור בניבוי הטוקן החסר, דבר שנתקלנו בו כאשר השתמשנו בפונקציה לניבוי טוקן ממוסך. דוגמה לכך היא בניבוי המילה "השר" במשפט "... אני לא [\*] לביטחון הפנים ... ", שבו הטוקנים אחרי הטוקן הממוסך הם כן הרלוונטיים שיכולים לתת הקשר.

בחלק מהמקרים בהם הפונקציה נכשלה בניבוי היה ניתן לשים לב לכך שהניבוי של הפונקציה היה טוקן שכן נתן משמעות למשפט למרות היותו לא הנכון. למשל, בניבוי הטוקן "מנגנון" במקום "עוד" במשפט " כדי להתגבר על הפער צריך לייצר [\*] , אבל ..."

מצד שני, הפונקציה כן הצליחה להתמודד עם השלמת הטוקן החסר בקולקציות נפוצות כמו הטוקן "דעת" כהמשך למשפט "לקבל חוות [\*]", וגם בהתמודדות עם טוקנים פחות שכיחים כהמשך של קולקציות מסוימות מאורך 2 אם ההקשר הרחב יותר של ה- trigram עזר בכך. דוגמה לכך שהוזכרה קודם היא בהצלחת הפונקציה בניבוי הטוקן "הפרטי" כהמשך למשפט "העובדים במגזר [\*]" למרות היות שני הצירופים "במגזר הערבי" ו- "במגזר החרדי" יותר נפוצים בקורפוס בהשוואה ל- "במגזר הפרטי".

הצעות שלדעתנו יכולות לשפר את ביצועי הפונקציה: שימוש ב- n-grams יותר גבוהים כגון Four-grams, Five-grams, שימוש בקורפוס יותר גדול שיכיל יותר קולקציות ובכללי יהווה מדגם גדול יותר, בנוסף להתחשבות בטוקנים שאחרי הטוקן החסר ולא רק באלה שלפניו במקרים שטוקנים כאלה כן נמצאים (למשל, כאשר מתבקשים לחזות את טוקן ממוסך שלא בסוף המשפט).

**שלב 2**

**שלב 2.1:**

הפונקציה 'get\_k\_n\_t\_collocations' מומשה. הפונקציה מחזירה אוסף של קולקציות העונות על הדרישות הבאות שמתקבלות כקלט: מספרם הוא k, אורך כל אחת מהן הוא n, ומספר המופעים של כל אחת לא פחות מ- t בקורפוס מסוים, שהוא pandas dataframe. כמו כן, הפונקציה מקבלת כקלט את סוג המדד שלפיו מחליטים איזה k מבין כל הקולקציות העונות על הדרישות צריך לבחור.

הפונקציה מתחילה בכך שהיא מחלצת את כל הקולקציות מאורך n הנמצאות בקורפוס. הדבר מתבצע ע"י כך שהיא עוברת על כל המשפטים בקורפוס, ועבור כל אחת שאורכה לפחות n, היא מחלצת את כל הקולקציות באורך n השונות שבתוכה (כל המחרוזות של טוקנים רצופים בגודל n). תוך כדי כך, היא סופרת את כמות הופעות כל קולקציה בפרוטוקולים השונים, ואת הקולקציות המופיעות בכל פרוטוקול וכמות הופעתם. הספירה מאוחסנת בעזרת שני מלונים, אחד לפי הקולקציה והשני לפי הפרוטוקול. הפונקציה, אח"כ, מסננת את הקולקציות ומשאירה רק את אלה שמספר הופעתם בכלל הפרוטוקולים הוא לפחות t. אחרי הסינון, הפונקציה מסדרת את הקולקציות מהכי נפוצה לפחות לפי המדד המועבר לפונקציה, שני המדדים הם "Frequency" ו- "TF-IDF", כאשר הראשון מסדר אותם לפי כמות ההופעות הכללי, והשני לפי הנוסחה הנתונה, שמתחשבת במספר הקולקציות השונות בפרוטוקול ומחשבת את השכיחות לא רק לפי כמות המופעים, אלא גם לפי מספר הפרוטוקולים השונים שבהם הקולקציה הופיעה. אם לפונקציה מועבר מדד אחר מהשניים הנ"ל היא מצגיה Value Error ומדפיסה הודעה בהתאם. בעקבות שלב הסידור, הפונקציה מחזירה את ה- k קולקציות הראשונות בסדר. במקרה שמספר הקולקציות הוא פחות מ- k, היא מחזירה את כולם.

**שלבים 2.2 – 2.4:**

הקוד שמדפיס את הדרוש נמצא ב- main ומשתמש בפונקציה מהשלב 2.1 לשם כך. הפלט מצורף כקובץ txt כנדרש.

**שלב 3**

**שלב 3.1:**

הפונקציה mask\_tokens\_in\_sentences מומשה. היא מקבלת כקלט רשימה של משפטים ומספר בטווח (0,100). הפונקציה ממסכת x% מהטוקנים במשפט (מחליפה אותם ב- [\*]). ומחזירה משפט חדש עם הטוקנים הממוסכים. במקרה ש- x% לא מספר שלם היא מקרבת אותו לשלם הכי קרוב, ובשל כך, משפט בעל פחות מ- 5 טוקנים (במקרה של המשפטים שלנו, משפט בעל 4 טוקנים) לא ימוסך אף טוקין בו. את הטוקנים שהיא ממסכת הפונקציה בוחרת באקראי.

**שלב 3.2:**

הקוד מומש ב- main, והפלט נמצא בשני קבצי txt מצורפים כנדרש.

**שלב 3.3:**

הקוד מומש ב- main, והפלט נמצא בקובץ txt המצורף כנדרש.

**שלב 3.4:**

הקוד מומש ב- main. להלן התוצאות שקיבלנו, שנמצאות גם כן בקובץ txt המצורף כנדרש.

הנוסחה שהשתמשנו בה לחישוב ה- Perplexity היא אותה נוסחה שהוזכרה בהרצאה עם השינוי שהשתמשנו רק בטוקנים שהחליפו טוקנים ממוסכים. מתמטית, נסמן ב- s את המשפט, וב- את הטוקנים שהחליפו טוקנים ממוסכים. כמו כן, נסמן ב- s[:i] את החלק במשפט עד הטוקן ה- i (לא כולל). אז הנוסחה היא: