

תוכנה 1 – חורף תשע"ו

תרגיל מספר 5

הנחיות כלליות:

- קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.
- הגשת התרגיל תעשה במערכת ה-moodle בלבד (<http://moodle.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש stav יקרא הקובץ stav_hw5.zip). קובץ ה-zip יכול:
 - קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
 - קבצי ה-java של התוכניות אותם התבקשתם לממש.

הגשת מחלקה עם חבילות: יש לכווץ בתוך קובץ ה-zip שאתם מגישים את כל היררכיית התיקיות מתחת ל-src באקליפס. למשל, כדי להגיש את המחלקה sw1.pac.MyClass העתיקו את התיקיה sw1 שמתחת ל-src כולל כל מה שבתוכה לתוך קובץ ה-zip.

בתרגיל זה נבנה מודל בסיסי של שפה (השפה האנגלית) מתוך טקסט נתון, תוך שימוש בסטטיסטיקה פשוטה של הופעת צמדי מילים בטקסט. לדוגמא, אם הטקסט שאנו לומדים עליו הוא "here comes the sun", הרי שבשפה שלנו יש 4 מילים, והצמדים <the,sum>, <comes,the>, <here,comes>, מופיעים כל אחד פעם אחת. הצמד <here,sun> לא מופיע בקובץ שלנו, ולכן סה"כ ראינו אותו 0 פעמים. מודל השפה שלנו למעשה יכול את אוצר המילים ואת מספרי המופעים של כל צמד מילים.

שימו לב, בתוכנית זאת עליכם להשתמש **במערכים בלבד**, ולא במבני נתונים גנריים כמו Lists/Maps/Sets וכו'. שימוש במבני נתונים גנריים יכול לגרום להורדת ניקוד משמעותית עד כדי ציון נכשל.

בתרגיל זה אתם יכולים להניח שכל שמות הקבצים הם חוקיים ואין צורך לממש טיפול נפרד למקרה שקריאה/כתיבה לקובץ נכשלת.

בניית מודל שפה

המחלקה BigramModel אשר נמצאת בחבילה il.ac.tau.cs.sw1.ex5 מייצגת את מודל השפה. מחלקה זו מכילה שני שדות:

שדה	משמעות
<code>String[] vocabulary</code>	מערך שמכיל את כל המילים באוצר המילים
<code>int[][] bigramCounts</code>	מערך דו מימדי המכיל את מספר הפעמים שבו ניראה כל צמד מילים בקובץ ממנו למדנו את מודל השפה. האיבר במקום ה j, i יהיה מספר הפעמים שראינו את המילה ה j מופיעה אחרי המילה ה i

(1) [20 נקודות] ממשו את המתודה `buildModelFromFile` המקבלת שם של קובץ ולומדת ממנו את מודל השפה. המתודה מאתחלת את שני השדות של המחלקה `BigramModel` (במידה והיו מאותחלים קודם, הערכים הקודמים נדרסים). **הפונק' תחזיר את גודל אוצר המילים.** השדה `vocabulary`:

- יכיל עד 15,000 מילים חוקיות מתוך הקובץ.
- מילה חוקית היא מילה העונה לאחת משתי ההגדרות:
 - i. מילה שמכילה לפחות אות אחת בשפה האנגלית.
 - ii. מספר שלם (למשל, 13 או 1984)
- כל מילה חוקית שמכילה לפחות אות אחת באנגלית יש להמיר ל `lowercase`.
- כל מספר שלם יומר למחרוזת "some_num" (השתמשו בקבוע שהוגדר בתחילת המחלקה)
- יש להתעלם מכל המילים ה"לא חוקיות".

נכניס את המילים החוקיות לתוך `vocabulary` על פי סדר הופעתן בקובץ (המילה הראשונה תיכנס לאינדקס 0, וכן הלאה). במידה ומילה מופיעה פעמיים, היא תישמר רק פעם אחת, תחת האינדקס שבו נשמרה בפעם הראשונה שנראתה (למשל, אם המילה `the` היא המילה השלישית בקובץ ונכנסה ל `vocabulary` תחת האינדקס 2, בפעם הבאה שנראתה אותה ה `vocabulary` לא ישתנה).

במידה והקובץ מכיל יותר מ 15,000 מילים שונות, נשמור רק 15,000 מילים ב `vocabulary`. אם הקובץ מכיל פחות מ 15,000 מילים שונות, גודלו של המערך `vocabulary` יותאם למספר המילים השונות בקובץ.

השדה `bigramCounts`:

- יכיל את מספר המופעים של צמדי במילים:
- אם `vocabulary[x] = word1` ו `vocabulary[y] = word2`, אזי `bigramCounts[x][y]` יכיל את מספר הפעמים שבהן המילה `word2` הופיעה מיד אחרי `word1`.
- הסטטיסטיקות יאספו רק עבור מילים שנמצאות באוצר המילים. אם הקובץ מכיל יותר מ 15,000 מילים, צמדים שמכילים מילה שלא נכנסה ל `vocabulary` לא יספרו.
- אם המילה ה `y` מעולם לא ניראתה אחרי המילה ה `x`, אזי `bigramCounts[x][y]` יכיל את הערך 0.

חתימת המתודה:

```
public int buildModelFromFile(String fileName) throws IOException
```

(2) [10 נקודות] ממשו את המתודה `buildModelFromFile` **saveModelToFile** המקבלת שם של קובץ ושומרת לתוכו את המודל. אם המודל טרם נטען למחלקה, המתודה לא תבצע שמירה לקובץ ותחזיר את הערך `false`. אחרת, תחזיר את הערך `true`. השמירה לקובץ תיעשה בפורמט הבא:
השורה הראשונה תכיל את מספר המילים שנמצאות ב `vocabulary`.
בשאר השורות נכתוב את מספר המופעים של כל צמד: הצמד `<w1,w2>` יכתב לפני הצמד `<w3,w4>` אם האינדקס של `w1` הוא נמוך יותר מזה של `w3` ב `vocabulary`, או ש `w1` ו `w3` הן אותה המילה, ואז ל `w2` יש אינקס נמוך יותר מ `w4`.
מכיוון שעבור רוב צירופי המילים האפשריים מספר המופעים שווה ל 0 (כלומר, רוב הצמדים האפשריים לא הופיעו בטקסט), נשמור רק את הצמדים שכן ניראו (כלומר, ערכים גדולים מ 0).
עבור הקובץ `all_you_need.txt` אשר מכיל את הטקסט הבא:

```
All you need is love  
All you need is love  
All you need is love love  
Love is all you need
```

נקבל קובץ פלט שניראה כך:

```
5 words
all,you:4
you,need:4
need,is:3
is,all:1
is,love:3
love,all:2
love,is:1
love,love:2
```

(שימו לב לכך שאנחנו מתייחסים לירידות שורה כמו לרווחים ומבחינתנו מדובר בטקסט רצוף).

הערה: שמירה בפורמט זה נחשבת לבזבזנית מבחינת גודל הקובץ. במקום לכתוב בכל פעם את המילים יכולנו לכתוב רק את האינדקסים שלהן, ולשמור בקובץ נוסף מיפוי בין אינדקס למילה האמיתית. מכיוון שבתרגיל זה אנו עובדים עם קבצים יחסית קטנים, בחרנו בפורמט בזבזני זה על מנת להקל על הקריאות של קבצי הפלט.

חתימת המתודה:

```
public boolean saveModelToFile(String fileName) throws IOException
```

(3) [15 נק'] השלימו את המימוש של המתודה loadModelFromFile המקבלת שם של קובץ וטוענת ממנו את מודל השפה לתוך השדות vocabulary ו bigramCounts. במידה ושדות אלה מאותחלים לערכים אחרים, ערכים אלה נדרסים.

שימו לב: למרות שהקובץ שומר כביכול רק את הנתונים של bigramCounts, למעשה, ניתן לשחזר ממנו בקלות גם את ה vocabulary. רמז: חשבו על הסדר שבו הנתונים נשמרים.

חתימת המתודה:

```
public int loadModelFromFile(String fileName) throws IOException
```

(4) [5 נק'] ממשו את המתודה getWordIndex המקבלת מחרוזת ומחזירה את האינדקס שלה ב vocabulary. במידה ומחרוזת זו לא מופיעה, הפונק' תחזיר 1- (הערה: בקובץ שהוגדר בראש המחלקה הקבוע אינו מוגדר במחלקה, כדאי, אך לא חובה, להגדיר קבוע כזה בעצמכם באופן דומה לקבועים האחרים המוגדרים במחלקה).

חתימת המתודה:

```
public int getWordIndex(String word)
```

(5) [5 נקודות] ממשו את המתודה getBigramCount המקבלת שתי מחרוזות word1 ו word2 ומחזירה את מספר הפעמים ש word2 הופיעה אחרי word1. אם אחת מהמילים לא נמצאת באוצר המילים (vocabulary) הפונק' תחזיר 0.

חתימת המתודה:

```
public int getBigramCount(String word1, String word2)
```

(6) [10 נקודות] ממשו את המתודה `getMostFrequentProceeding` המקבלת מילה `word` ומחזירה את המילה שהופיע אחריה הכי הרבה פעמים. במידה ויש כמה מילים עם אותו מספר מופעים, תוחזר המילה בעלת האינדקס הנמוך יותר ב `vocabulary`. במידה ואחרי המילה `word` לא ניראתה אף מילה אחרת (חשבו באיזה מצב זה יכול לקרות) המתודה תחזיר `null`.

חתימת המתודה:

`public String getMostFrequentProceeding(String word)`

(7) [10 נקודות] ממשו את המתודה `buildSentence` המקבלת מילה `word` ומספר שלם `n` גדול מ 1. המתודה תבנה משפט באורך עד `n` באופן הבא:

- המילה הראשונה במשפט תהיה `word`. המילה השניה תהיה המילה שמופיעה הכי הרבה פעמים אחרי `word` במודל השפה שלנו. במידה ויש כמה מילים כאלה, תיבחר המילה עם האינדקס הכי נמוך (בדומה לסעיף הקודם). המילה השלישית תהיה המילה הכי שכיחה אחרי המילה השניה וכן הלאה עד שנקבל משפט בעל `n` מילים.
- במידה ונגיע למילה שאף מילה לא הופיעה אחריה במודל השפה שלנו, המשפט יסתיים במילה זו ויתכן ויכיל פחות מ `n` מילים.
- אם `word` אינה קיימת באוצר המילים שלנו, הפונק' תחזיר מחרוזת ריקה.
- אותה המילה יכולה לחזור על עצמה מספר פעמים במהלך המשפט.

לדוגמא, עבור מודל השפה שמופיע בסעיף 2: אם נרצה לבנות משפט בן 4 מילים שיתחיל במילה `love`, נקבל את המשפט `love all you need`.

- המילה השניה תהיה `all` – היא נספרה פעמיים אחרי `love`. גם המילה `love` הופיעה פעמיים אחרי `love`, אבל מכיוון שהאינדקס שלה גבוה יותר, נבחר ב `all`.
- אחרי `all` תופיע המילה `you` שזו למעשה המילה היחידה שהופיעה אחריה במודל השפה.
- אחרי `you` תופיע המילה `need` שהיא גם המילה היחידה שהופיעה אחרי `you` במודל השפה.

חתימת המתודה:

`public String buildSentence(String word, int n)`

(8) [10 נקודות] נרצה לבדוק אם באמצעות מבנה הנתונים שלנו ניתן ללמוד משהו על התנהגות השפה והיחסים בין מילים. לצורך בדיקה זו, נשתמש בקובץ טקסט גדול, וננסה לראות אם יש קשר בין מילים שמופיעות בקובץ הטקסט בהקשר דומה: כלומר, בסמיכות לאותן המילים. לצורך כך, ממשו את המתודה הסטטית `calcCosineSim` המקבלת שני מערכים של מספרים שלמים שאורכם זהה, ומחשבת את דמיון הקוסינוסים ביניהם על פי הנוסחה הבאה:

$$\frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

כאשר A ו B הם וקטורים באורך n הממומשים ע"י מערכים של int -ים. A_i הוא האיבר ה i במערך A , ו B_i הוא האיבר ה i במערך B .

ככל שהוקטורים הם דומים יותר, הערך שנקבל יהיה גדול יותר.
 לדוגמא: עבור שני הוקטורים [1,0,5], [1,2,3] נבצע את החישוב הבא:

$$\frac{1 * 1 + 0 * 2 + 5 * 3}{\sqrt{1^2 + 0^2 + 5^2} * \sqrt{1^2 + 2^2 + 3^2}}$$

אם אחד הוקטורים מכיל רק אפסים, הפונקציה תחזיר 0.

חתימת המתודה הסטטית:

```
public static double calcCosineSim(int[] arr1, int[] arr2)
```

(9) [15 נקודות] כעת, נשתמש בפונקציה שמימשנו בסעיף הקודם בשביל למצוא את המילה הכי דומה

למילה word על פי ההקשר שלהן בטקסט.

נגדיר את המושג "וקטור מייצג עבור מילה word".

- עבור מילה word הקיימת באוצר המילים שלנו, הוקטור המייצג הוא וקטור הנגזר מתוך מודל השפה ואורכו הוא כמספר המילים באוצר המילים (vocabulary).
- האיבר במקום ה x יכיל את מספר הפעמים שבהם הצירוף word, vocabulary[x] נראה במודל השפה (כלומר, כמה פעמים הצמד word, vocabulary[x] נספר במודל).
- לדוגמא: עבור מודל השפה שמתואר בסעיף (2), עבור המילה love נקבל את הוקטור הבא: [2, 0, 0, 1, 4]

- סידור המילים באוצר המילים הוא: love, is, need, you, all, ולכן האיבר הראשון בוקטור ייצג את המופע של all אחרי love, האיבר השני ייצג את המופעים של you אחרי love, וכן הלאה.
- הצירוף love, all מופיע פעמיים על פי מודל השפה.
- הצירופים love, you ו love, need מופיעים 0 פעמים.
- הצירופים love, is ו love, love מופיעים כל אחד פעם אחת (הצירוף love, is מופיע פעם אחת, ו love, love מופיע פעמיים).

- את הוקטורים המייצגים עבור כל מילה נייצג ע"י מערכים חד מימדיים של מספרים שלמים שאורכם הוא כגודל אוצר המילים.

ממשו את הפונקציה getClosestWord אשר מקבלת מילה word ומחזירה את המילה הדומה לה ביותר על פי השוואת הוקטורים המייצגים של כל המילים לזה של word.

עליכם לחלץ את הוקטור המייצג את word מתוך השדה bigramCounts, ולאחר מכן, למצוא מילה w כך שה cosineSim בין הוקטור המייצג של w לבין זה של word הוא הגבוה ביותר מבין שאר המילים באוצר המילים. הציפיה שלנו היא שמילים בעלות וקטורים "דומים" יותר יהיו דומות במשמעות או באופן השימוש בהן בטקסט.

חתימת המתודה:

```
public String getClosestWord(String word)
```

הערה לידע כללי:

על מנת לקבל תוצאות משמעותיות בשיטה המתוארת בסעיף 9 עלינו להשתמש בכמות מאוד גדולה של טקסט, עם אוצר מילים רחב ודוגמאות שימוש רבות לכל מילה. כמו כן, עלינו להשתמש במידע נוסף: לא להשוות רק את המילים שמופיעות אחרי המילה, אלא גם את המילים הקודמות לה, ולהשתמש ביותר ממילה עוקבת/קודמת אחת. יחד עם זאת, גם בשיטה המאוד בסיסית שמתוארת בסעיף 9 ניתן להגיע לתוצאות מעניינות עבור הקובץ emma.txt (הספר "אמה" של ג'יין אוסטין, מתוך האתר <https://www.gutenberg.org>). כך לדוגמא, נראה ש great ו good הם בעלי וקטורים דומים יחסית, וכן "she", "he" ו "trick", "scheme".

מה שיכול לעזור לנו לשפר את התוצאות הוא עיבוד לשוני על הטקסט: למשל, להסב את כל הטיות הפעלים ושמות העצם לצורת היחיד (goes -> go, books -> book).

כאמור, המימוש שלכם הוא בסיסי ביותר, אבל מהווה בסיס לטכניקות מתקדמות יותר בהן משתמשים בשביל ללמוד מידע על שפה.

שימוש בחוזים:

בשלב התרגיל מופיעים חוזים עבור חלק מהמתודות. השתמשו בחוזים על מנת להבין אילו הנחות אפשר להניח על הפלט, ובאיזה מקרי קצה צריך לטפל.

טסטר

לתרגיל זה מצורפת מחלקת טסטר. על מנת להריץ את הטסטר עליהם לוודא שהנתיבים לקבצים מהם קוראים/כותבים תואמים למיקומם על המחשב שלכם, ולעדכן את כתובות הקבצים במידת הצורך. מומלץ להוסיף בדיקות משלכם תוך שימוש בקבצי טקסט חדשים. ברוב המקרים כדאי להשתמש בקבצי קלט קצרים עליהם קל לחשב את התשובה אותה תרצו לקבל מפונקציות שלכם.

בהצלחה!