יסודות מדעי המחשב - תרגיל בית מספר 4

מתרגל אחראי: אופיר יעיש

הנחיות כלליות:

- מועד אחרון להגשה: כמפורסם בתיבת ההגשה ב-Moodle או בלוח ההודעות.
- מטרת התרגיל הינה לתרגל כתיבה, ולרכוש מיומנות בכלים שנלמדו עד כה בכיתה.
- קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה. ודאו שאתם מבינים את כל המשימות.
- רמת הקושי של המשימות אינה אחידה.
- את התרגיל יש לפתור לבד!
- בתיבת ההגשה במערכת ה-VPL ישנו קובץ שלד לכתיבת הקודם שלכם. עבור כל משימה כתבו הפתרון שלכם במקום המתאים למשימה.
- אין למחוק שום קטע קוד הנמצא בשלד. עליכם רק להוסיף את הפתרון שלכם בתוכו.
- אין לשנות את שם/שמות ה-Requested Files.
- השאלות יבדקו באופן אוטומטי. הפלט שעליכם להחזיר בכל תרגיל צריך להיות בדיוק כפי שנדרש. כמו כן, באופן אקראי יבדקו גם עבודות באופן ידני.
- כאשר תבוצע בדיקה ידנית, תתבצע גם בדיקת Readability - שימו לב שאתם משתמשים בשמות משתנים אינפורמטיביים וכותבים הערות בכל סעיף.
- בדיקה עצמית: כדי לוודא את נכונותן ואת עמידותן של הפונקציות לקלטים שונים, בכל שאלה הריצו אותן עם מגוון קלטים: אלה שמופיעים בדוגמאות וקלטים נוספים עליהם חשבתם. וודאו כי הפלט נכון. הבדיקה תתבצע על מגוון דוגמאות ולא בהכרח אלה שיינתנו פה.
- ניתן להשתמש בכל החומר שנלמד בקורס.
- במידה שלא צוין אחרת, יש להניח את נכונות הקלט על פי תיאור המשימה.
- במידה שלא עניתם על שאלה מסוימת, אנא השאירו את מילת הקוד pass על מנת שהקוד שלכם יצליח לעבוד.

הוראות חשובות למשימה זו:

- חל איסור על שימוש במשתנים גלובליים. כמו כן, חל איסור על כתיבת פונקציות פנימיות.
- יש צורך בייבוא חבילת NumPy.
- שימו לב שישנם חלקים בהם יש איסור על שימוש בלולאות (כולל list comprehension). המטרה היא לאלץ אתכם לבחון חלופות שחבילת NumPy נותנת.
- ניתן להשתמש בכל המתודות והכלים ש-NumPy מספקת, לרבות אלו שלא נלמדו בכיתה.
- צוות הקורס שומר לעצמו את הזכות לבדוק חלק מהנחיות בעבודה בבדיקות הראשוניות, ורק בבדיקה הסופית לבדוק את כל ההנחיות. לדוגמה, ייתכן שבבדיקה הראשונית לא ייבדק אי שימוש בלולאות.

בלימוד מכונה, נהוג להמיר נתונים מייצוגים שלנו כבני אנוש קל להבין לייצוגים בעלי משמעות מתמטית או/וגם לייצוגים שלמכונה "קל להבין". לאחר המרה כזאת, ניתן לבצע פעולות הנדרשות לניתוח הנתונים.

לתהליך ההמרה מייצוג "אנושי" לייצוג אחר נקרא קידוד (**encoding**) ולתהליך ההפוך נקרא פיענוח (**decoding**). בתרגיל זה, נכיר, בין היתר, כמה דרכים להמרת משפט, ויהיה עליכם לממש את פונקציות ה-encode ו-ה-decode של אחד מהם.

חלק הכרחי בתהליך ההמרה הוא יצירת אוצר מילים (**vocabulary**) שרק ממנו המשפטים יכולים להיות מורכבים.

לדוגמה, vocabulary יכול להיות הרשימה הבאה: ["the", "cat", "cats", "sat", "on", "mat", "alexa"]. לכן, משפטים יכולים להיות מורכבים אך ורק מה-vocabulary (ללא אבחנה ב-letter case) וסימני פיסוק. בתרגיל שלנו סימני הפיסוק אינם חלק מאוצר המילים. לכן, עבור אוצר המילים הנ"ל, משפט תקין לדוגמה יהיה:

"The cat, Alexa, sat on the mat."

ומשפט שאינו תקין יהיה:

"The cat, Alexa, sit on the mat."

שכן המילה sit אינה חלק מה-vocabulary שלנו.

שימו לב שכל מחרוזת מופיעה פעם אחת באוצר המילים בצורת lower case שלה (ז"א אותיות קטנות). כלומר, עבורנו, "the" ו-"The" הן מחרוזות זהות.

על מנת להמחיש את פעולת ה-encode וה-decode, נגדיר את **הייצוג הקטגורי** (אותו לא תצטרכו לממש, אך נעזר בו לצורך הסברים לאורך כל השאלה):

בהינתן רשימת vocabulary ומשפט (כנ"ל), ה-encoding הקטגורי ממיר כל מילה למיקום (ולא האינדקס) של אותה מילה ברשימת ה-vocabulary.

לדוגמה, עבור ה-vocabulary שראינו לעיל (["the", "cat", "cats", "sat", "on", "mat", "alexa"]). הייצוג של המילה "The" יהיה 1, של "cat" יהיה 2 וכן הלאה. לכן הייצוג של המשפט:

"The cat, Alexa, sat on the mat."

יהיה:

[1, 2, 7, 4, 5, 1, 6]

כלומר, עבור הייצוג הקטגורי, פעולת ה-encode תקבל את המשפט ואת ה-vocabulary ותחזיר:

[1, 2, 7, 4, 5, 1, 6]

פעולת ה-decode תקבל את הייצוג [1, 2, 4, 5, 1, 6] ואת ה-vocabulary, ותחזיר את המשפט:

"the cat alexa sat on the mat"

שימו לב שבפעולת ה-decode איבדנו את סימני הפיסוק והאבחנה בין lower case ו- upper case, וזאת בשל הצורה שבה הגדרנו את המילון שלנו.

<u>הבהרה חשובה:</u> בתרגיל זה ניתן להניח שמשפטים מורכבים מהאלמנטים הבאים:

1. מילים שהם רצף של אותיות באנגלית או רצף של ספרות. <u>לא</u> ניתן להניח שאין מילים המורכבות מאותיות וספרות כגון "CR7".
2. רווחים וסימני פיסוק המפרידים בין המילים או נמצאים בתחילת/סוף המשפט.

שימו לב: בשל הגדרות התרגיל, מילים כמו check-in ו-cat's יחשבו באוצר המילים כ-2 מילים כל אחד בשל סימני הפיסוק שבתוכם. check-in מורכב מ-check ו-in ו-cat's מורכב מ-cat ו-s.

<u>הגדרת Tokenization (טוקניזציה – פירוק לטוקנים)</u>

טוקניזציה היא תהליך בו מפרקים טקסט למילים בודדות או יחידות לשוניות בסיסיות שנקראות טוקנים. באופן כללי, טוקן יכול להיות מילה, מספר, סימן פיסוק, או כל רצף תווים בעל משמעות. בתהליך זה, משפטים או טקסטים ארוכים מתפצלים לרשימה של טוקנים כדי להקל על ניתוח הטקסט ועיבודו ע"י המכונה.

כפי שהובהר לעיל, בתרגיל שלנו, המילון יורכב אך ורק מרצף אותיות באנגלית או רצף ספרות. כלומר הטוקנים יהיו מסוג זה בלבד. בתהליך הטוקניזציה שלנו, אנו מסירים את כל האלמנטים שהם לא מסוג זה כך שהם לא יהפכו לטוקנים. בנוסף, הטוקנים יהיו כולם בצורת ה-lower case שלהם.

לדוגמה, אם ניקח את המשפט הבא:

"The cat's mat was dirty! So, the cat did not sit on it."

אז לאחר תהליך הטוקניזציה נקבל את רשימת הטוקנים הבאה:

["the", "cat", "s", "mat", "was", "dirty", "so", "the", "cat", "did", "not", "sit", "on", "it"]

<u>שימו לב:</u> הסדר של האלמנטים (רצפי האותיות או הספרות) בטקסט נשמר לאחר הטוקניזציה.

<u>חלק א' (יצירת אוצר מילים):</u>

בחלק זה יהיה עליכם לממש את המחלקה Vocabulary המיועדת לנהל את אוצר המילים שנוצר מרשימת משפטים. היא כוללת מתודות לטוקניזציה של משפטים, יצירה והרחבה של אוצר המילים, ושליפת מילים לפי אינדקס או שליפת אינדקס לפי מילים.

ממשו את המחלקה על פי פירוט המתודות הבאות:

תיאור	מתודה
קלט: רשימת משפטים (אופציונלי). משפט הוא רצף של תווים. ניתן להניח שהמשפטים תקינים על פי ההנחיות לעיל. המתודה מאתחלת את רשימת אוצר המילים. אם סופקו משפטים כארגומנט למתודת האתחול, המתודה יוצרת את רשימת אוצר המילים	__init__(self, sentences=None)

ממשפטים אלה וממיינת אותם על פי סדר התווים בטבלת ה-ASCII. **שימו לב שאוצר המילים נוצר על פי ההנחיות לעיל, ועליו להישמר כמאפיין מסוג רשימה.**	
מחזירה את אורך רשימת אוצר המילים.	__len__(self)
קלט: רשימת משפטים תקינים (ניתן להניח). המתודה מאתחלת את רשימת אוצר המילים להיות רשימה ריקה ויוצרת את אוצר המילים מהמשפטים שנקלטו וממיינת את המילים באוצר המילים על פי האופן הנ״ל.	create_vocabulary(self, sentences)
קלט: רשימת משפטים תקינים (ניתן להניח). המתודה מוסיפה מילים חדשות מהמשפטים לרשימת אוצר המילים וממיינת אותם על פי האופן הנ״ל.	extend_vocabulary(self, new_sentences)
המתודה מחזירה את רשימת אוצר המילים.	get_vocabulary(self)
תמיכה ב-indexer כך שתאפשר שליפת מילים מאוצר המילים על פי אינדקסים או שליפת אינדקסים על פי מילים באוצר המילים. ז״א, אם vec הוא מופע של המחלקה Vocabulary שמכיל את רשימת אוצר המילים הבאה: [“check”, “closed”, “in”, “is”, “now”, “the”] אז הוספת התמיכה ב-indexer תאפשר את הפעולות הבאות לדוגמה: vec[5] which returns the string “the” vec[“is”] which returns the integer 3 vec[[0, 4, 3, 0]] which returns the list: [“check”, “now”, “is”, “check”]	תמיכה ב-indexer (כלומר באופרטור ה-[]).

	vec[["is", "check", "in", "closed"]] which returns the list: [3, 0, 2, 1] vec[["is", 0, "in", 1]] which returns the list: [3, "check", 2, "closed"] במידה שמילה אינה נמצאת באוצר המילים, יש להרים שגיאה מסוג ValueError עם הערה כרצונכם. במידה שאינדקס הוא מחוץ לתחום, יש להרים שגיאה מסוג IndexError עם הערה כרצונכם. **אינדקס שלילי הוא גם מחוץ לתחום.** <u>הערה:</u> חשבו כיצד ניתן לתמוך בכל האפשרויות במינימום חזרה על קוד.
tokenize(self, sentence)	קלט: מחרוזת המייצגת משפט תקין (ניתן להניח). במידה שכל המילים במשפט נמצאים באוצר המילים, המתודה מחזירה את הטוקניזציה של המשפט כפי שהוגדר לעיל. אחרת, המתודה תחזיר None. בקישור הבא ניתן למצוא מתודות שימושיות של מחרוזות שניתן להשתמש בהם על מנת לממש את המתודה כנדרש.

הערה חשובה: זכרו שגם בתוך המחלקה אתם יכולים להשתמש במתודות שהגדרתם. כמו כן, אתם מוזמנים להגדיר מתודות עזר (רגילות או סטטיות) נוספות במחלקה כרצונכם. הדבר יעזור לכם להפוך את הקוד לתמציתי.

דוגמאת הרצה:

עבור הקוד הבא:

```
voc = Vocabulary(["The cat sat on the mat"])
print(voc.get_vocabulary())

voc.create_vocabulary(["The cat, Alexa, sat on the mat.", "The dog, Bob, sat on the log"])
print(voc.get_vocabulary())

voc.extend_vocabulary(["The cat sat on the mat"])
print(voc.get_vocabulary())

print(voc[2])
print(voc[0])
print(voc[[3, 7]])
print(voc["cat"])
print(voc[["cat", "dog"]])
print(voc[["cat", 5]])


print(voc.tokenize("the dog, Alexa, sat on Bob!"))
print(voc.tokenize("the dog, Alex, sat on the log"))
```

פלט ההדפסות הוא:

['cat', 'mat', 'on', 'sat', 'the']

['alexa', 'bob', 'cat', 'dog', 'log', 'mat', 'on', 'sat', 'the']

['alexa', 'bob', 'cat', 'dog', 'log', 'mat', 'on', 'sat', 'the']

cat

alexa

['dog', 'sat']

2

[2, 3]

[2, 'mat']

['the', 'dog', 'alexa', 'sat', 'on', 'bob']

None

חלק ב׳ (ייצוג נקודה-חמה):

ייצוג נקודה-חמה (one-hot)הוא ייצוג כזה שבהינתן רשימת vocabulary באורך n, ומשפט בעל m מילים, כל מילה שהייצוג שלה בייצוג הקטגורי הוא i מומרת לווקטור בגודל n כך שבתא ה-i בווקטור יימצא הערך 1 ובשאר התאים יימצא הערך 0.

לדוגמה, האיור הבא ממחיש את פעולת ה-encode של ייצוג זה על vocabulary ומשפט:

vocabulary = [cat, cats, check, in, mat, on, sat, the]

sentence: "The cat sat on the mat."

the	cat	sat	on	the	mat
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	1	0	0	0
1	0	0	0	1	0

הייצוג באיור לעיל הוא פר מילה. ייצוג של המשפט כולו הוא מטריצה בת m שורות כך שכל שורה i-ית מחזיקה את הייצוג נקודה-חמה של המילה המילה ה-i-ית במשפט (כלומר למטריצה n עמודות).

לכן המטריצה עבור המשפט באיור הקודם תראה כך:

the	0	0	0	0	0	0	0	1
cat	1	0	0	0	0	0	0	0
sat	0	0	0	0	0	0	1	0
on	0	0	0	0	0	1	0	0
the	0	0	0	0	0	0	0	1
mat	0	0	0	0	1	0	0	0

בחלק זה יהיה עליכם לממש את המחלקה OneHotEncoder המיועדת להמיר משפטים לייצוג one-hot ולהחזיר מטריצות מקודדות בחזרה למשפטים (בהתאם לאילוצים של איבוד סימני הפיסוק).

ממשו את המחלקה על פי פירוט המתודות הבאות:

תיאור	מתודה
קלט: אובייקט מסוג Vocabulary המייצג את אוצר המילים שבעזרתו מבוצעת פעולת ה-encoding וה-decoding. ניתן להניח שהקלט תקין. המתודה מאתחלת את המקודד עם אוצר המילים המתקבל.	__init__(self, vocabulary)
קלט: מחרוזת המייצגת משפט. ניתן להניח שהקלט תקין. המתודה מחזירה מטריצת numpy.ndarray שהוא הייצוג נקודה-חמה של המשפט. **<u>שימו לב:</u>** • חל איסור על שימוש בלולאות ו- list comprehension באופן מפורש. • ניתן להניח שהמשפט מורכב אך ורק ממילים הנמצאות באוצר המילים. • יש לדאוג שכל תא במטריצה החוזרת יהיה מסוג np.int8 (ז״א, כל תא יהיה מסוג integer הצורך תא זיכרון של 8 bit). זכרו שמערכי numpy יכולים להכיל גם מחרוזות וניתן לבצע מניפולציות שונות (כגון הפעלת מתודות) על מערכים אלו.	encode(self, sentence)
קלט: מטריצת numpy.ndarray בגודל mXn שהיא הייצוג נקודה-חמה של משפט. המתודה מחזירה מחרוזת שהיא המשפט לאחרת פעולת ה-decode. **<u>שימו לב:</u>** • חל איסור על שימוש בלולאות ו- list comprehension באופן מפורש.	decode(self, encoding)

• ניתן להניח שהמטריצה תקינה ומייצגת משפט שניתן להרכיב בעזרת אוצר המילים. זכרו כי המתודה join יכולה לקבל כארגומנט כל אובייקט שניתן לרוץ עליו (Iterable), ולכן מקבלת בין היתר: רשימות, מחרוזות, ומערכי numpy חד-ממדיים.	

זכרו שאתם יכולים שעל אובייקט מסוג Vocabulary ניתן להפעיל את המתודות שמימשתם בחלק א'.

<u>דוגמאת הרצה:</u>

עבור הקוד הבא:

```
voc = Vocabulary(["The cat sat on the mat!",
                  "The cat, Alexa, sat on the mat.",
                  "The dog, Bob, sat on the log"])
print(voc.get_vocabulary())
one_hot_encoder = OneHotEncoder(voc)
encoded = one_hot_encoder.encode("the dog, Alexa, sat on Bob!")
print(encoded)
decoded = one_hot_encoder.decode(encoded)
print(decoded)
```

פלט ההדפסות הוא:

```
['alexa', 'bob', 'cat', 'dog', 'log', 'mat', 'on', 'sat', 'the']
[[0 0 0 0 0 0 0 0 1]
 [0 0 0 1 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0 0 0]]
the dog alexa sat on bob
```

חלק ג׳ (ייצוג סוכם):

ייצוג סוכם הוא ייצוג כזה שבהינתן רשימת vocabulary באורך n, ומשפט בעל m מילים, כל מילה שהייצוג שלה בייצוג הקטגורי הוא i מומרת לווקטור בגודל n כך שבתא ה-i בווקטור וכל התאים בעלי אינדקס נמוך יותר יימצא הערך 1 ובשאר יימצא התאים הערך 0.

לדוגמה, האיור הבא ממחיש את פעולת ה-encode של ייצוג זה על vocabulary ומשפט:

vocabulary = [cat, cats, check, in, mat, on, sat, the]

sentence: "The cat sat on the mat."

the	cat	sat	on	the	mat
1	1	1	1	1	1
1	0	1	1	1	1
1	0	1	1	1	1
1	0	1	1	1	1
1	0	1	1	1	1
1	0	1	1	1	0
1	0	1	0	1	0
1	0	0	0	1	0

הייצוג באיור לעיל הוא פר מילה. ייצוג של המשפט כולו הוא מטריצה בת m שורות כך שכל שורה i-ית מחזיקה את הייצוג סוכם של המילה המילה ה-i-ית במשפט (כלומר למטריצה n עמודות).

לכן המטריצה עבור המשפט באיור הקודם תראה כך:

the	1	1	1	1	1	1	1
cat	1	0	0	0	0	0	0
sat	1	1	1	1	1	1	0
on	1	1	1	1	1	0	0
the	1	1	1	1	1	1	1
mat	1	1	1	1	0	0	0

בחלק זה יהיה עליכם לממש את המחלקה SumEncoder המיועדת להמיר משפטים לייצוג סוכם ולהחזיר מטריצות מקודדות בחזרה למשפטים (בהתאם לאילוצים של איבוד סימני הפיסוק).

ממשו את המחלקה על פי פירוט הדרישות למחלקה OneHotEncoder בחלק ב׳ של התרגיל.

דוגמאת הרצה:

עבור הקוד הבא:

```
voc = Vocabulary(["The cat sat on the mat!",
                  "The cat, Alexa, sat on the mat.",
                  "The dog, Bob, sat on the log"])
print(voc.get_vocabulary())
sum_encoder = SumEncoder(voc)
encoded = sum_encoder.encode("the dog, Alexa, sat on Bob!")
print(encoded)
decoded = sum_encoder.decode(encoded)
print(decoded)
```

פלט ההדפסות הוא:

```
['alexa', 'bob', 'cat', 'dog', 'log', 'mat', 'on', 'sat', 'the']
[[1 1 1 1 1 1 1 1 1]
 [1 1 1 1 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 0]
 [1 1 1 1 1 1 1 0 0]
 [1 1 0 0 0 0 0 0 0]]
the dog alexa sat on bob
```

חלק ד׳ (Bag of words):

Bag of words הוא מודל פשוט לייצוג טקסט כווקטור מספריים, בו כל משפט מיוצג כווקטור של מספרים. כל מימד בווקטור מתאים למילה מסוימת מאוצר המילים הכולל את כל המילים שמופיעות בכל המסמכים או המשפטים. הערך של כל מימד מציין את מספר הפעמים שהמילה המתאימה לו מופיעה במשפט או במסמך הנתון.

לכן שלבי יצירת ה-Bag of word הם שניים:

1. יצירת אוצר המילים – אוסף כל המילים הייחודיות המופיעות בכל המשפטים.
2. בניית ווקטור – לכל משפט או מסמך יוצרים ווקטור באורך אוצר המילים, כך שבכל תא בווקטור מכניסים את מספר הפעמים שהמילה המתאימה מופיעה במשפט או במסמך.

לדוגמה:
עבור אוצר המילים הבא:

[“cat”, “dog”, “log”, “mat”, “on”, “sat”, “the”]

ה-bag of words של המשפט:

״The dog sat on the log״

הוא הווקטור:

[0, 1, 1, 0, 1, 1, 2]

וה-bag of words של המשפט:

״The cat sat on the log״

הוא הווקטור:

[1, 0, 1, 0, 1, 1, 2]

בחלק זה יהיה עליכם לממש את המחלקה BagOfWordsEncoder המיועדת להמרת משפטים לייצוג bag of words ומחשבת שכיחות מילים על פני משפטים רבים.

ממשו את המחלקה על פי פירוט המתודות הבאות:

תיאור	מתודה
קלט: אובייקט מסוג Vocabulary המייצג את אוצר המילים שבעזרתו מייצרים ייצוג ה- bag of words. ניתן להניח שהקלט תקין. המתודה מאתחלת עם המילון המתקבל.	__init__(self, vocabulary)

sentence_to_bag_of_words(self, sentence)	קלט: מחרוזת המייצגת משפט. ניתן להניח שהקלט תקין (ז״א מייצג משפט כלשהו). המתודה מחזירה ווקטור numpy.ndarray שהוא הייצוג bag of words של המשפט. **<u>שימו לב:</u>** • במידה שאחד הטוקנים של המשפט אינם חלק מהמילון יש להרים שגיאה מסוג ValueError עם הערה כרצונכם. • יש לדאוג שכל תא במטריצה החוזרת יהיה מסוג np.int32 (ז״א, כל תא יהיה מסוג integer הצורך תא זיכרון של 32 bit).
calculate_word_frequency(self, sentences)	קלט: רשימת משפטים תקינים (ניתן להניח). המתודה מחזירה ווקטור numpy.ndarray המכיל עבור כל מילה באוצר המילים (נניח אוצר מילים באורך n) את אחוז ההופעות שלה בכלל המשפטים. התא במקום ה-i ($1 \leq i \leq n$) בווקטור יכיל את אחוז ההופעות של המילה ה-i-ית באוצר המילים. **<u>שימו לב:</u>** • את הערכים בווקטור יש לעגל עד ל-4 ספרות אחרי הנקודה העשרונית (ניתן לעגל בעזרת np.round). ניתן להניח שזהו ייצוג מספיק טוב כך שלא ישפיע על יכולת השוואה בין התאים. • במידה שאחד הטוקנים של המשפט אינם חלק מהמילון יש להרים שגיאה מסוג ValueError עם הערה כרצונכם.

דוגמאת הרצה:

עבור הקוד הבא:

```
voc = Vocabulary(["The cat sat on the mat!",
                  "The cat, Alexa, sat on the mat.",
                  "The dog, Bob, sat on the log",])
print(voc.get_vocabulary())
bow = BagOfWordsEncoder(voc)
encoded = bow.sentence_to_bag_of_words("the dog, Alexa, sat on the log!")
print(encoded)
freq = bow.calculate_word_frequency(["the dog, Alexa, sat on the log!", "the cat Alexa", "Bob sat
on log"])
print(freq)
```

פלט ההדפסות הוא:

['alexa', 'bob', 'cat', 'dog', 'log', 'mat', 'on', 'sat', 'the']

[1 0 0 1 1 0 1 1 2]

[14.2857 7.1429 7.1429 7.1429 14.2857 0.   14.2857 14.2857 21.4286]

חלק ה׳ (שימוש במחלקות):

בחלק זה תממשו פונקציות שבהן תצטרכו לעשות שימוש במחלקות שמומשו בחלקים א׳ ו-ד׳.

ה׳-1 (מציאת המילים בעלות השכיחות הגבוה ביותר במשפטים):

ממשו את הפונקציה most_frequent_words(sentences) המקבלת רשימה של משפטים תקינים (ניתן להניח), ומחזירה את רשימה עם המילים בעלות השכיחות הגבוה ביותר במשפטים. על הרשימה המוחזרת להיות ממוינת כפי שהוגדר שאוצר מילים ממוין.

הבהרה: מילה שכיחה בעלת השכיחות הגבוה ביותר היא מילה המופיע הכי הרבה פעמים במשפטים. אנו רושמים המילים בעלות השכיחות הגבוה ביותר שכן ייתכנו כמה מילים שהן בעלות מספר זהה של הופעות.

שימו לב:

- חל איסור על שימוש בלולאות ו-list comprehension באופן מפורש.
- הגדרת משפט היא כפי שהוגדר בתחילת העבודה.

דוגמה:

הקריאה:

most_frequent_words(["This is an example",
"The example is just study proposes.",
"Example is the best thing for practice!"])

תחזיר:

['example', 'is']

ה׳-2 (Cosine similarity):

Cosine similarity הוא מדד למידת הדמיון בין שני ווקטורים במרחב. מדד זה משמש במיוחד בניתוח טקסטים ובתחומים אחרים של למידת מכונה.

עבור 2 וקטורים A ו-B באורך זהה, הנוסחה לחישוב ה-cosine similarity היא:

$$Cosine\ similarity = \frac{A \cdot B}{\|A\|\|B\|}$$

כאשר מכפלת הווקטרים היא מכפלה סקלרית ו-|| || היא נורמת L2 של הווקטור.

ממשו את הפונקציה cosine_similarity(sentence1, sentence2, vocabulary) המקבלת 2 מחרוזות משפטים תקינים (ניתן להניח), sentence1 ו-sentence2, ואוצר מילים vocabulary (מסוג Vocabulary). הפונקציה מחזירה את ה-cosine similarity של ווקטרי ה-bag of words של sentence1 ו-sentence2.

שימו לב:

- חל איסור על שימוש בלולאות ו-list comprehension באופן מפורש.
- ניתן להניח שהמשפטים מורכבים אך ורק ממילים הנמצאות באוצר המילים.
- ייצוג ה-bag of words של המשפטים מבוסס על אוצר המילים המתקבל כארגומנט לפונקציה.
- את ערך ה-cosine similarity יש לעגל עד ל-4 ספרות אחרי הנקודה העשרונית (ניתן לעגל בעזרת np.round). ניתן להניח שזהו ייצוג מספיק טוב כך שלא ישפיע על יכולת השוואה ערכי cosine similarity.

דוגמה:

עבור הגדרת המילון:

```
voc = Vocabulary(["This is an example",
                  "The example is just study proposes.",
                  "Example is the best thing for practice!"])
```

הקריאה הבאה:

```
cosine_similarity("This is an example",
                  "The example is just study proposes.",
                  voc)
```

תחזיר:

```
0.4082
```

ה׳-3 (מציאת המשפטים הדומים ביותר)

ממשו את הפונקציה closest_sentences(sentences, vocabulary) המקבלת רשימה של משפטים תקינים (ניתן להניח), ואוצר מילים vocabulary (מסוג Vocabulary). הפונקציה מחזירה רשימה עם 2 האינדקסים (בסדר עולה) של זוג המשפטים ברשימה שהכי דומים אחד לשני על פי מדד cosine similarity שהוגדר בסעיף ה׳-3 על ייצוג ה-bag of words.

שימו לב:

- חל איסור על שימוש בלולאות ו-list comprehension באופן מפורש.
- ניתן להניח שהמשפטים מורכבים אך ורק ממילים הנמצאות באוצר המילים.
- ניתן להניח שקיימים זוג משפטים אחד שהוא הדומה ביותר.

הדרכה (לא מחייבת): למדו על פונקציות np.vactorize ו-np.fromfunction.

דוגמאת הרצה:

עבור הקוד הבא:

```
sentences = ["This is an example",
             "The example is just study proposes.",
             "Example is the best thing for practice!"]
voc = Vocabulary(sentences)

print(voc.get_vocabulary())
print(cosine_similarity(sentences[0], sentences[1], voc))
print(cosine_similarity(sentences[0], sentences[2], voc))
print(cosine_similarity(sentences[1], sentences[2], voc))
print(closest_sentences(sentences, voc))
```

פלט ההדפסות הוא:

['an', 'best', 'example', 'for', 'is', 'just', 'practice', 'proposes', 'study', 'the', 'thing', 'this']

0.4082

0.378

0.4629

[1, 2]