Improving Semantic Textual Similarity by Dividing Ambiguous Words Meanings

Artificial Intelligence Project 236502
Faculty of Computer Science
Technion Institute of Technology

Written By:

Binat Ziser	206078040
Nathaniel Joseph	302305396
Tal Swisa	204594923

Supervised By:
Prof. Shaul Markovitch
Leon Anevy

2020

הקדמה:

מטרת הפרויקט:

נושא ה-word embedding תופס תאוצה כיום ביישומים רבים ב-NLP. Word embedding הינו יצוג מילים בעזרת וקטורים עליהם ניתן לבצע חישובים ולנסות למצוא משמעויות של מילים ואף משפטים או טקסטים שלמים ע"י פריסת מרחב וקטורי וקירוב\הרחקה של הוקטורים אחד מהשני לפי המילים שנמצאות בסביבת כל מילה. ניתן בעזרת word embedding למצוא דימיון בין מילים, אלימות ורבלית בטקסט, רמת שיכנוע של הכותב ואף ציניות. ישנם אלגוריתמים רבים המשתמשים בעיקרון זה, אנו נתמקד ב-word2vec ו-doc2vec, כאשר המטרה של הפרויקט הינה לשפר את הביצועים של אלגוריתמים אלו במציאת דימיון בין משפטים.
הרעיון של הפרויקט נבנה בעקבות כך ששמנו לב כי באלגוריתם של Word2Vec (שעל בסיסו בנוי Doc2Vec), אין התייחסות מיוחדת למילים בעלות מספר משמעויות, כלומר מילים אלו, ככל המילים, מקבלות ייצוג וקטורי יחיד. הבעיה בכך היא שהייצוג הוקטורי של מילה בעלת מספר משמעויות ינסה לתפוס את כל המשמעויות של המילה, וכנראה לא ייצג אף משמעות בצורה טובה. למשל, הוקטור שיחושב על ידי האלגוריתם עבור המילה עכבר יערבב בתוכו את המשמעות של החיה עכבר יחד עם המשמעות של עכבר של מחשב.

רעיון כללי לפתרון הבעיה:

האלגוריתמי word embedding שונים מתבססים על הרעיון שאת המשמעות של מילה ניתן ללמוד לפי ההקשר שבו היא מופיעה, כאשר המילה "הקשר" מתייחסת למילים שמופיעות בסביבת המילה. למשל, ניתן ללמוד את המשמעות של המילה מלפפון לפי כך שהיא מופיעה ליד המילים עגבנייה, סלט וכו'.
האלגוריתם יקבל טקסט כקלט ויבצע עליו word embedding, כך נוכל לקבל עבור כל משפט את הוקטור המתאים לו. בעת נסתכל על מילה w, שיש לה שתי משמעויות- m_1, m_2. ההופעות של w שמתאימות למשמעות m_1 מופיעות בכל הנראה בהקשר שונה מההקשר שבו מופיעות ההופעות של w שמתאימות למשמעות m_2 ולכן גם הוקטורים המתאימים v_1, v_2 המייצגים מילים המופיעות בהקשר של המשמעויות השונות m_1, m_2 יקבלו ייצוג שונה וימצאו במקומות שונים במרחב הנפרס ע"י כל הוקטורים שהמרחב הוקטורי יצר.
הפתרון שאנחנו מציעים הוא לקבוע כי המילה w היא בעצם שתי מילים – w_1 ו- w_2 כאשר w_1 מתאימה למשמעות m_1 ו- w_2 מתאימה למשמעות m_2. לשם כך אנחנו צריכים להגדיר מהן המשמעויות של w על פי הוקטורים v_i כאשר כל וקטור v_i מייצג משפט שבו w מופיעה, ולדעת לזהות אותן בטקסט. אם את כל הופעה של w בטקסט נחליף ב- w_i עבור m_i (המשמעות המתאימה), אז מהרצת אלגוריתם word embedding מחדש נוכל לקבל וקטור עבור כל מילה נוכל לקבל תוצאות מדוייקות יותר מכיוון שהמילים כעת מופרדות גם לפי המשמעויות שלהם.

מאגרי המידע:

בשביל לפתור את הבעיה הזדקקנו למאגר מילים ומשפטים רחב ומקיף ככל הניתן.

- *Brown*:
  השתמשנו ב-*Brown Corpus* הנמצא בחבילה של *NLTK*. מאגר בראון הינו המאגר הראשון באנגלית המכיל יותר ממיליון מילים. המאגר מכיל טקסטים מ-500 מקורות שונים אשר מויינו על פי ג'אנר.
  מאגר זה מגיע בצורת רשימה של משפטים, כאשר כל משפט מיוצג באמצעות רשימה מקושרת של מילים.
- *Wikipedia*:
  ויקיפדיה הינו מאגר המידע הגדול והממויין ביותר שקיים למין האנושי. חלק גדול מהפרויקט היה התאמת האלגוריתמים לרוץ עם מאגר בסדר גודל של ויקיפדיה.

במאגר זה יש כ-4,700,000 מאמרים ומיליארדים של מילים. התבסנו על כך שכל מאמר בויקיפדיה ערוך ובדוק ולכן למילים הנמצאות תחת כל מאמר יש קשר לשם המאמר ולכן התייחסנו לכל מאמר בויקיפדיה כמשפט אחד המתאר את כותרת המאמר.

<u>אופן פעולה כללי:</u>

מימשנו ארבעה מודלים, ולכולם אותם שלבים, ומימוש שונה לכל שלב. ניתן לתאר את אופן פעולת המודלים באופן כללי:

<u>שלב האימון:</u>

מקבלים corpus. מבצעים preprocessing (הסרת stop words כמו for ו- and, ולפעמים גם הבאת המילים לצורת השורש שלהם) ומגדירים בדרך כלשהי, ייחודית לאלגוריתם, את המשמעויות של המילים ב- corpus.

<u>שלב תיקון ה- corpus:</u>

לכל מילה w שהוגדרה עבורה יותר ממשמעות אחת, נשנה כל הופעה של המילה ב- corpus ל- w_i עבור i – האינדקס של המשמעות המתאימה, לפי ההקשר שבו מופיעה המילה.

<u>שלב אימון Doc2Vec:</u>

נאמן מודל Doc2Vec על ה- corpus לאחר השינוי.

<u>אופן החזרת ווקטור עבור טקסט:</u>

בקבלת טקסט, לכל מילה w בטקסט שהוגדרה עבורה יותר ממשמעות אחת, נשנה את ההופעה של המילה בטקסט ל- w_i עבור i – האינדקס של המשמעות המתאימה, כאשר המשמעות נקבעת לפי ההקשר שבו המילה נמצאת בטקסט.
לאחר מכן נעביר את הטקסט המתוקן למודל Doc2Vec שאימנו על ה- corpus המתוקן, ונחזיר את הוקטור שהוא מחזיר.

<u>מבחן:</u>

כדי לבחון את הביצועים של המודלים השתמשנו ב- dataset שמפורט עליו בהמשך.
בקצרה, ב- dataset מופיעים זוגות של משפטים, וציון לרמת הדמיון בניהם על פי בני-אדם (ציון בין 0 ל- 5), נסמן ב- sim_i את הציון לרמת הדמיון בין זוג המשפטים ה- i, ונגדיר את המרחק בין זוג המשפטים ה- i להיות: $dist_i = 5 - sim_i$. נסמן ב- d_{human} ווקטור שבכניסה ה i מכיל את $dist_i$.
כפי שתיארנו, כל מודל יכול לקבל משפט s ולהחזיר ווקטור v שמייצג אותו סמנטית (בעזרת מודל Doc2Vec שאומן על הקורפוס שלאחר התיקון). לפיכך ניתן לקחת שני משפטים s_a, s_b ולהחזיר את המרחק בניהם על ידי חישוב המרחק בין הווקטורים v_a, v_b שמתקבלים עבור המשפטים - $d_{minkowski_2}(v_a, v_b) = \sqrt{\sum_{j=0}^{n}\left(v_{a_j} - v_{b_j}\right)^2}$.
לאחר חישוב המרחק בין כל זוג משפטים ב- dataset באמצעות המודל M שאנחנו רוצים לבחון, נקבל ווקטור מרחקים שנסמן ב- d_M, שהכניסה ה- i שלו מכילה את $d_{minkowski_2}\left(v_{i_a}, v_{i_b}\right)$, כאשר v_{i_a} ו- v_{i_b} הם זוג הווקטורים שהמודל מחשב עבור זוג המשפטים ה- i ב- dataset.
כעת ניתן לחשב את הקורלציה שבין הווקטורים d_{human} ו- d_M. עבור מודל M שמפיק את ווקטור המרחקים בין הזוגות d_M אנחנו מסמנים ב- $CorrScore(M)$ את ערך הקורלציה בין d_M ל- d_{human}.

אם נתייחס אל הציונים ב- dataset כאל האמת, אז ככל שהערך $CorrScore(M)$ גבוה יותר, כך המרחקים בין הווקטורים v_{i_a} ו- v_{i_b} שמייצר המודל הם נכונים יותר, מה שאומר שהווקטורים מייצגים בצורה נכונה יותר את הסמנטיקה של המשפטים, כלומר המודל מבצע את המשימה שלו – ייצוג סמנטיקה של טקסט על ידי וקטור – בצורה טובה יותר.

כדי לבחון האם האלגוריתמים משפרים את הביצועים, אנחנו משווים אותם ל- Doc2Vec רגיל - מודל Doc2Vec שהתאמן על ה- corpus המקורי (לאחר ביצוע preprocessing), ללא שינוי (תיקון) שלו על ידי אחד האלגוריתמים.
המבחן מבוצע על פני כל זוגות המשפטים ב- dataset.
על האלגוריתמים השונים שפיתחנו הפעלנו Doc2Vec עם פרמטרים שונים, שמגיעים לערכי קורלציה שונים, לכן בעת ההשוואה, נציין את ערך הקורלציה שאליו הם מגיעים.

<u>מימושים:</u>
כאמור פיתחנו ארבעה אלגוריתמים שונים, ההבדל העיקרי בניהם הוא בדרך שבה הם מגדירים את המשמעויות שיש למילים. האלגוריתם הראשון מתבסס על ייצוג וקטורי של המילים שמתקבל ממודל Doc2Vec מאומן. שלושת האלגוריתמים הנותרים מתבססים על בניית גרפים שמבטאים את מידת הקשר בין מילים, והסקה של המשמעויות השונות לפי גרפים אלו. אלגוריתמים אלו לא נעזרים במידע נוסף מלבד ל- corpus.

<u>אופן פעולת האלגוריתם:</u>

- <u>עיבוד מקדים:</u>
  האלגוריתם יקבל כקלט קורפוס עליו אנו נבצע עיבוד מקדים בו אנו מורידים ניקוד מהמילים, תווים שאינם באנגלית או מספרים והופכים כל מילה למילה הקנונית שלה. בשלב הבא של העיבוד המקדים אנו יוצרים מילון CanonicToDoc אשר ממפה לנו בין מילים קנוניות למשפטים בהם מילה הממופה לאותה מילה קנונית נמצאת. כעת ניצור מודל DOC2VEC מאומן על כל הכתבות לשם קבלת הוקטורים המתאימים לכל מאמר.

- <u>למידה:</u>
  כעת יש בידינו את כל המידע הדרוש למציאת מילים בעלות דו-משמעות ונתחיל בתהליך הקיבוץ (Clustering). נעבור על כל מילה הנמצאת ב-CanonicToDoc ונאתחל במערך את כל הוקטורים המתאימים לשמות המאמרים המקושרים לאותה מילה. לאחר מכן נשלח לאלגוריתם OPTICS את המערך והוא יבצע לנו קיבוץ של הוקטורים. OPTICS יחזיר לנו את הוקטורים והקבוצות אליהם הם שייכים, במידה ואכן נמצאו קבוצות אנו נחשב את הוקטור הממוצע של כל קבוצה, כל וקטור ממוצע כשזה ישמש לנו לפיצול מילים בעלות דו-משמעות. בהעדר מגבלות זיכרון ניתן לבצע קיבוץ על כלל הוקטורים בקורפוס, קיבוץ זה יאפשר לנו לזהות את המשמעויות השונות הנמצאות בטקסט, נשמור בצורה דומה את הוקטורים הממוצעים. יתרון של שיטה זו הינה מהירות הריצה, בנוסף נמצא כך את כל קבוצות המשמעויות השונות לפי כל המשפטים בקורפוס ולא נסתמך רק על המשפטים השונים אשר מילה מסויימת נמצאת בהם. החיסרון שאיתו יצטרכו להתמודד הוא שלא נדע אילו מהמילים בעלות דו-משמעות לאילו לא (ניתן להשתמש בשילוב השיטות לפתור בעיה זו). חיסרון נוסף הינו צריכת הזיכרון הנדרשת לשם קיבוץ כל הוקטורים.

  לדוגמא: עבור המילה mouse, וחמישה וקטורים השייכים לה - $v_i, 0 \leq i \leq 4$. מכיוון שהמילה עכבר יכולה לקבל משמעות של עכבר מחשב או החיה עכבר, נקבל בתהליך הקיבוץ כי v_0 שייך לקבוצה 0, v_1, v_2 לקבוצה 1 ו-v_3, v_4 שייכים לקבוצה 2. על פי אלגוריתם הקיבוץ, הקבוצה 0 אומרת כי לא נמצאה קבוצה מתאימה לוקטור. עבור

הקבוצות האחרות נחשב את הממוצע של הוקטורים השייכים לכל קבוצה - avg_0, avg_1.
כעת נשמור במילון את המילה mouse עם הוקטורים avg_0, avg_1.
אם בתהליך הקיבוץ לא נמצא פיצול למילה או נמצא פיצול אחד אז היא תשאר כמו שהיא.
במידה ונמצאו 2 או יותר פיצולים נשמור את כמות הוקטורים שנמצאו.

- <u>תהליך הסיווג:</u>
נעבור על הקובץ שקיבלנו כקלט, עבור כל משפט נחפש במילון המתאים את הוקטור המשויך לו ב-DOC2VEC. עבור כל מילה במשפט נבדוק האם מצאנו למילה פיצול בתהליך הקיבוץ. אם קיים פיצול, נבדוק לאיזה מהוקטורים של אותה מילה אנחנו הכי קרובים ונסווג את המילה למילה החדשה המתאימה לפי המיקום של הוקטור הכי קרוב במילה. לאחר תהליך הסיווג ניצור מודל DOC2VEC מהקובץ המסווג שייצרנו ונשתמש בו למציאת דמיון בין מילים ומשפטים.
לדוגמא: אם נמשיך מהדוגמא הקודמת אז כעת כאשר נגיע למילה mouse בטקסט אנו נבדוק האם קיים למילה פיצול, נראה שקיים פיצול. נבדוק לאיזה מהפיצולים שנמצאו הוקטור של המשפט שבו מצאנו את המילה mouse הכי קרוב, אם אנו הכי קרובים לוקטור במיקום 1 במערך אז נשנה את המילה ל-mouse_1. בסוף התהליך כל מופע של המילה mouse ישונה ל-mouse_0 או mouse_1 וכאשר נשלח את הקורפוס המסווג ל-doc2vec נקבל וקטורים שונים עבור המילים הללו.

נציג תרשים זרימה של אופן פעולת האלגוריתם:

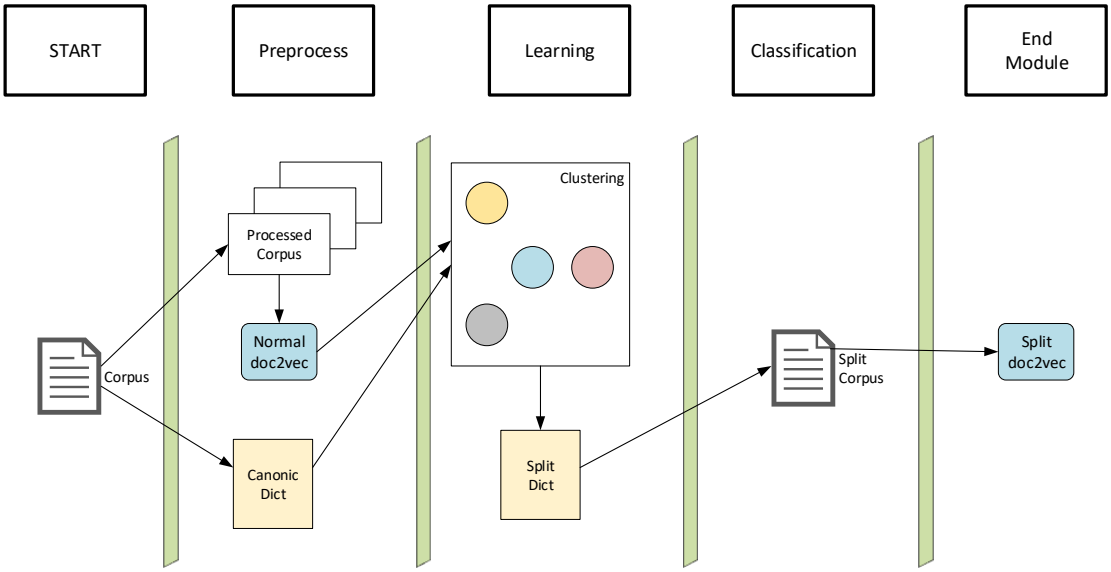
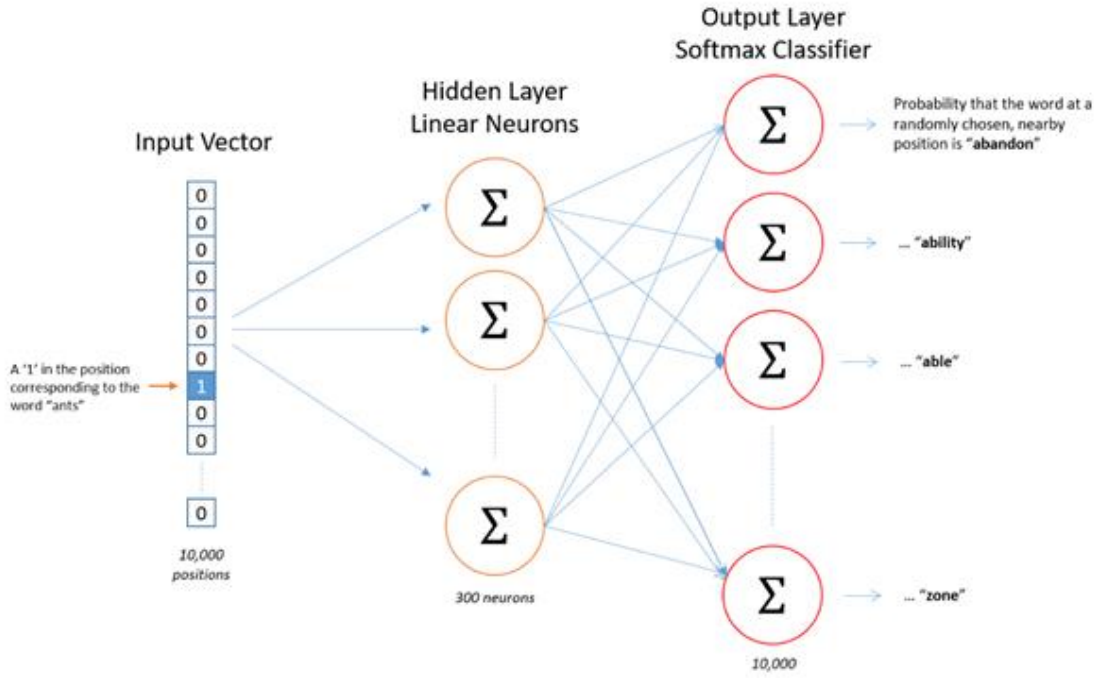


Figure1 - WordSplitter Algorithm Flow Chart

<u>אלגוריתם WORD2VEC:</u>
אלגוריתם זה משמש להטמעת מילים (embedding word) ומשתמש בעיקרון שלמילים שמופיעות באותו קונטקסט תיהיה משמעות דומה – כלומר מילה מקבלת את המשמעות שלה מהמילים שמופיעות לידה.
באופן כללי, האלגוריתם לוקח מאגר (corpus) בתור קלט ומקצה וקטור לכל מונח ייחודי במאגר. נניח והקורפוס שלנו מורכב מרצף המילים $w_0, w_1, \dots, w_{N-1}$. נובע כי המשמעות של המילה w_j נקבעת לפי חלון המילים המופיעות מימינה ומשמאלה $w_{j-M}, \dots, w_j, \dots, w_{j+M}$, כאשר M הוא גודל החלון. כעת לכל מילה w מקצים וקטור v ומגדירים את ההסתברות שמילה w_a היא באותו

הקשר של המילה w_b כ-$P(w_a|w_b) = \frac{e^{v_{w_b} * v_{w_a}^T}}{\sum_{k=0}^{V} e^{v_{w_b} * v_{w_k}^T}}$. מכאן שהמטרה של אלגוריתם Word2Vec הינו למצוא את קבוצת הוקטורים V שתמקסם את הפונקציה $\frac{1}{N}\sum_i^N \sum_j^i \log P(w_j|w_i)$.

בסיום ריצת האלגוריתם נקבל ייצוג מטריצה אשר כל שורה בה מייצגת וקטור של מילה המופיעה בקורפוס.



דוגמה קלסית לשימוש באלגוריתם היא ביצוע פעולות מתמטיות על וקטורים של מילים שמראה ש"מלך" פחות "גבר" ועוד "אישה" שווה "מלכה".

<u>אלגוריתם DOC2VEC:</u>

מכיוון שאנו צריכים למצוא שיוויון בין משפטים ולא בין מילים בודדות נשתמש בהכללה של Word2Vec למשפטים – Doc2Vec.

אלגוריתם זה פועל בדומה ל-Word2Vec אך הוא משתמש בממוצע משוכלל של המילים הנמצאות בכל משפט לשם יצירת וקטור אשר ייצג את משפט זה. באופן זה, נוכל למדל את המשפטים במאגר באמצעות וקטורים.

וקטורים אלו נחוצים למציאת משפטים בעלי אותם משמעויות. מילה מסויימת המופיעה במשפטים שונים בעלי משמעויות שונות יכולה להיות מילה בעלת דו-משמעות – כלומר אנו נשתמש בדימיון בין המשפטים במאגר לשם מציאת מילים בעלות דו-משמעות.

פיצול המילים לפי המשמעות שלהם אמורה בתיאוריה לשפר את דיוק הוקטורים על ידי שיפור הממוצע המשוכלל של המילים המוכלות במשפט.

אלגוריתם DBSCAN:

אלגוריתם ה-DBSCAN, קיצור של Density Based Spatial Clustering of Applications with Noise, הינו אלגוריתם קיבוץ המשתמש בלמידה לא מפוקחת אשר מחלקת את נקודות המידע שהיא מקבלת כקלט לקבוצות, כאשר היא שומרת על כך של התכונות המאפיינות כל קבוצה יהיו ייחודיות.

האלגוריתם מקבל כקלט את נקודות המידע אותם רוצים לקבץ, לאחר מכן הוא עובר על כל הנקודות ומחשב עבור כל נקודה את התכונות שלה (מרחקים מהנקודות האחרות) ולאחר מכן מנסה לחלק את הנקודות לקבוצות. הרעיון העיקרי מאחורי האלגוריתם הינו שמכל נקודה בתוך קבוצה שאנו מנסים ליצור, יהיו מספר מינימלי של נקודות מידע השייכות לקבוצה ברדיוס מינימלי כלשהו ממנה. בניגוד לאלגוריתמים אחרים אשר מנסים להתאים קבוצות לפי צורות מוגדרות מראש, אלגוריתם ה-DBSCAN מנסה למצוא את הקבוצות לפי המרחקים והתכונות של הנקודות עצמן. בצורה זו ניתן ליצור קבוצות שאינן בהכרח קמורות ולהתמודד יותר טוב עם מידע רועש.

אלגוריתם OPTICS:

אלגוריתם ה-OPTICS, קיצור של Ordering Points to Identify Cluster Structure, הינו אלגוריתם קיבוץ המתבסס על DBSCAN.

האלגוריתם מקבל כקלט את נקודות המידע אותם רוצים לקבץ וראשית מבצע - K-Nearest Neighbors על כל הנקודות למציאת מקומות בעלי צפיפות גבוהה של נקודות. לאחר מכן האלגוריתם משתמש במקומות הצפופים שהוא מצא ומהם מרחיב את הקבוצות עד ליצירת הקבוצות הסופיות. בנוסף, בניגוד ל- DBSCAN, אלגוריתם זה שומר את תצורות הקבוצות למרחקי רדיוסים שונים מכל נקודה ובכך מתאים את עצמו טוב יותר לקלטים גדולים יותר.

בחרנו להשתמש באלגוריתם זה מכיוון שהיינו זקוקים לאלגוריתם המבצע קיבוץ ללא הגדרה מוקדמת של צורת הקבוצות ומספר האיברים בכל קבוצה, ההתמודדות הטובה שלו עם קלטים גדולים וכמו כן רצינו אלגוריתם שיודע להתמודד עם רעש - זאת משום שהווקטורים שאנו מקבלים מ-DOC2VEC הינם וקטורים הנוצרים משפה שאינה מוגדרת היטב (שפת הדיבור).

אלגוריתם Louvain:

אלגוריתם זה הינו אלגוריתם חמדן למציאת קבוצות בגרפים.

האלגוריתם מנסה למקסם את ערך מדד המודולריות של החלוקה.

כדי לחשב את ערך המודולריות, משווים עבור כל קבוצה את מספר הקשתות בתוך הקבוצה בגרף הנתון, למספר הקשתות הצפוי בגרף אקראי המתאים לגרף הנתון. כאשר מספר הקשתות הצפוי בגרף האקראי הוא מספר הקשתות שהיינו מצפים שיהיו בתוך הקבוצה בגרף שבו לכל צומת יש את אותה הדרגה, אך הקשתות היו מחוברות לצמתים בצורה רנדומלית.

ערך המודולריות של החלוקה גבוה כאשר מספר הקשתות בין צמתים השייכים לאותה קבוצה גבוה ביחס למה שהיינו מצפים (כלומר ביחס לגרף אקראי).

עבור גרפים גדולים מאוד, ההנחה כי בגרף האקראי הקשתות של צומת מסוים יכולות להתחבר לכל צומת אחר היא בעייתית, למשל מפני שבגרפים מספיק גדולים, ההסתברות שתהיה קשת בין שתי קבוצות היא קטנה מ- 1, ואז קיום של קשת כזו מעיד על קשר חזק בין הקבוצות, מה שאומר שמודולריות מקסימלית מתקבלת עבור חלוקה שכוללת קבוצה אחת של כל הצמתים.

כדי להתגבר על הבעיה הזו מגדירים את הפרמטר resolution שתפקידו לאפשר חלוקה לקבוצות קטנות, על ידי מישקול יחסי של החשיבות של קשתות בגרף האקראי.

כאשר ערך הפרמטר הוא 0 אין לו משמעות, ובכל שהוא גדל כך האלגוריתם מותאם למציאת קבוצות קטנות יותר.

האלגוריתם מאתחל את החלוקה להיות קבוצה לכל צומת בגרף.

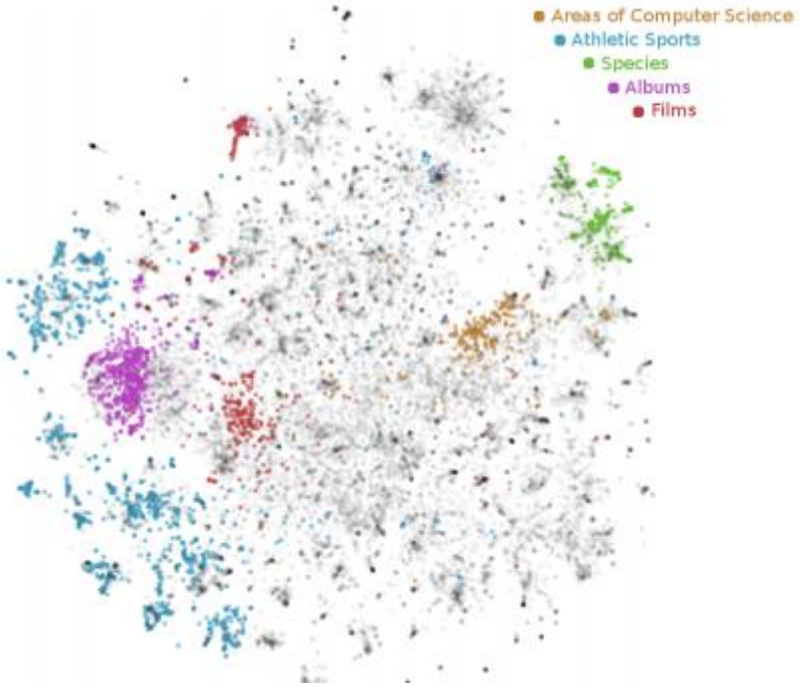
לאחר מכן האלגוריתם עובר כל צמתי בגרף לפי סדר כלשהו ומנסה להגדיל את ערך המודולריות על ידי העברת הצומת לאחד מהקבוצות אליהם שייכים הצמתים השכנים.

האלגוריתם ממשיך לנסות לשפר את ערך המודולריות עד אשר לא ניתן לשפר את המודולריות של הקבוצות.

אלגוריתם Word Splitter:

אלגוריתם זה משתמש ב-DOC2VEC למידול המשפטים לווקטורים וב- OPTICS למציאת קבוצות הוקטורים.

הרעיון מאחורי האלגוריתם הינו להשתמש במרחב הוקטורי אותו DOC2VEC יוצר בסיווג המשפטים שלו בשביל למצוא מילים בעלות דו - משמעות. למשל, בסיווג כתבות מויקיפדיה ניתן היה לראות כי וקטורי ה-DOC2VEC השייכים לכתבות באותם נושאים היו מקובצים אחד ליד השני במרחב הוקטורי[1].



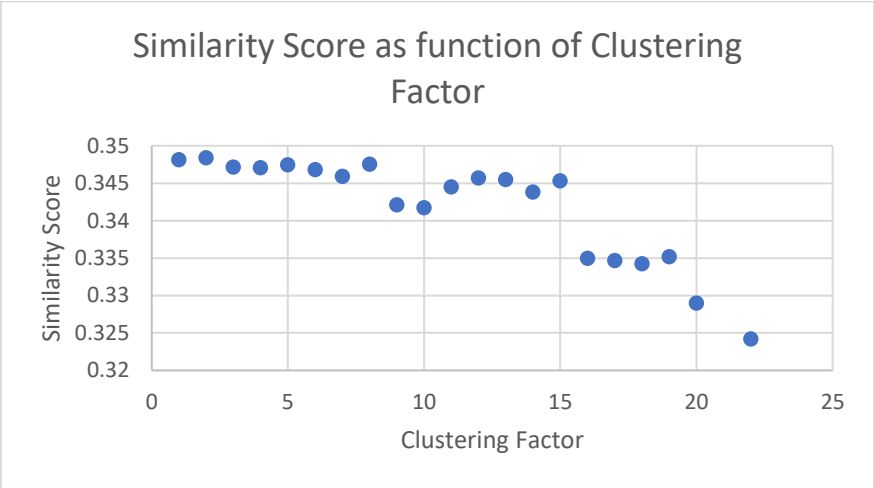
ניתן לראות בתמונה כי מאמרים הקשורים למדעי המחשב מקובצים אחד ליד השני (בחום) וכי מאמרים הקשורים למינים גם מקובצים אחד ליד השני (בירוק). ולכן, אם נקח לדוגמא את המילה עכבר, אשר יכול להופיע בהקשר של מדעי המחשב כעכבר מחשב ובהקשר של מינים כבעל חיים. נוכל לקבץ את המשפטים שהמילה עכבר מופיעה בהם וליצור מילים חדשות שאחד מהם תייצג עכבר מחשב והשניה תייצג את בעל החיים.

בעזרת נתינת הגדרה יותר מפורטת למילים אנו מקווים כי נצליח לשפר את תוצאות WORD2VEC ו-DOC2VEC בסיווג דימיון בין מילים ומשפטים.

ניסויים:

הפרמטר היחידי של האלגוריתם הינו Clustering Factor, פרמטר זה שולט במספר המינימלי של איברים בקבוצה. אם מילה מסויימת מופיעה x פעמים בטקסט, אז אלגוריתם הכיווץ יקבל x וקטורים אותם הוא ינסה לקבץ, נסמן ב-cf את ה-Clustering Factor ונקבל כי המספר המינימלי של איברים בכל קבוצה הינו $\left\lceil \frac{x}{cf} \right\rceil$. כאשר $cf = 1$ אנו מקבלים את ההתנהגות הרגילה של Doc2Vec מכיוון שהמיספר המינימלי של הקבוצה הינה כל האיברים בקבוצה ולכן האלגוריתם לא ימצא קיבוץ לעולם. בגרף המצורף ניתן לראות כי במקרה של הקורפוס של Brown האלגוריתם אינו מציג שיפור משמעותי במציאת דימיון בין משפטים, כאשר $cf = 2$ ניתן לראות שיפור מזערי אך לא משמעותי מספיק בשביל לייחס זאת לאלגוריתם ולא לרנדומליות של Doc2Vec בייצירת הוקטורים.

ניתן לראות כי כפי שצפינו, עבור ערכי cf גבוהים, האלגוריתם מתחיל לפצל כל מילה והתוצאות מתחילות לרדת.



<u>הצעה לפרויקט המשך:</u> לאחר בחינה של הקורפוס, ראינו כי המילים אותן הוא מנסה לפצל הם מילים חסרות משמעות (המון שמות, קללות וכדומה). ניסינו לאמן את האלגוריתם על ויקיפדיה ונתקלנו בבעיות של חוסר זיכרון וזמני ריצה ארוכים. אנו מצליחים לאמן את האלגוריתם עם גודל וקטור של 256 לכל ויקיפדיה אך גודל הוקטור מאוד קטן למספר המילים והמשפטים בויקיפדיה ולכן איננו מצליחים לראות שיפור כלשהו בתוצאות. נשמח לנסות להרחיב את השימוש באלגוריתם לריצה על מאגרי מידע גדולים יותר.

אלגוריתמים גרפיים

מבוא:

באלגוריתמים אלה אנחנו משתמשים בגרף כדי למדל את הקשר בין המילים, כל צומת בגרף מייצגת מילה, ובשני האלגוריתמים הגרפיים הראשונים המשקל על הקשת בין המילים מייצג את מידת הקשר בהם.

מידת הקשר בין w1 ו- w2 מוגדר על ידי:

$$Connection(w_1, w_2) = \frac{\frac{1}{2}[\#(\mathrm{w1}) + \#(\mathrm{w2})]}{\frac{1}{2}[\#(\mathrm{w1}) + \#(\mathrm{w2})] - \#(\mathrm{w1}, \mathrm{w2})}$$

כאשר #(wi) הוא מספר ההופעות של wi ב- corpus ו- #(w1, w2) הוא מספר ההופעות של w1 ו- w2 ביחד באותו החלון.

חלון הוא רצף של מילים מה- corpus באורך מסוים. למשל עבור המשפט "אני מסביר מה זה חלון", אם מסתכלים על חלונות באורך 3, החלונות הם: " אני מסביר מה", "מסביר מה זה" ו- "מה זה חלון".

נשים לב כי $\#(\mathrm{w1}, \mathrm{w2}) \leq \min\{\#(\mathrm{w1}), \#(\mathrm{w2})\} \leq \frac{1}{2}[\#(\mathrm{w1}) + \#(\mathrm{w2})]$ ולכן הביטוי שנקבל הוא אי שלילי.

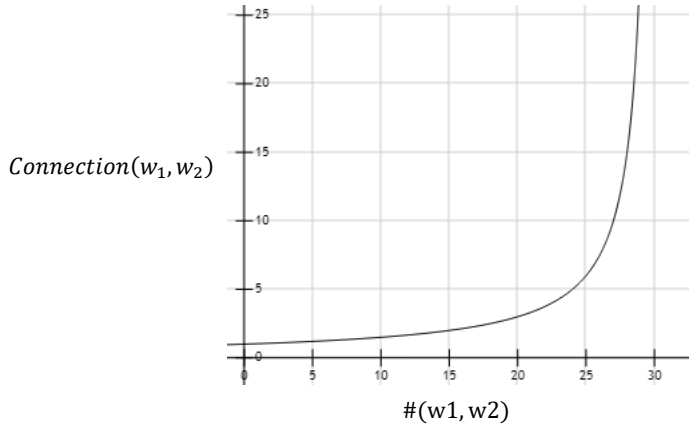
הגדרה זו משתמשת במספר ההופעות המשותפות של המילים ביחס למספר ההופעות של כל מילה כדי להגדיר את הקשר בין המילים.

לדוגמה אם בכל פעם ש- w1 מופיעה בחלון גם w2 מופיעה ובכל פעם ש- w2 מופיעה בחלון גם w1 מופיעה אז נקבל

$$\#(\mathrm{w1},\mathrm{w2}) = \frac{1}{2}[\#(\mathrm{w1}) + \#(\mathrm{w2})]$$

במקרה זה מידת הקשר תהיה אינסוף.

אם נניח לדוגמה כי המילה w1 מופיעה 20 פעמים, והמילה w2 מופיעה 40 פעמים אז נקבל כי $\frac{1}{2}[\#(\mathrm{w1}) + \#(\mathrm{w2})] = 30$. כעת מידת הקשר בין המילים כפונקציה של $\#(\mathrm{w1},\mathrm{w2})$ נראית כך :



כרצוי מידת הקשר בין המילים עולה ככל שמספר ההופעות המשותפות של המילים עולה.
למעשה נשתמש באלגוריתמים 1 ו- 2 נשתמש בפונקציה $Connection(w_1, w_2)$ כדי למדוד את מידת הקשר בין זוג מילים, וב- $1/Connection(w_1, w_2)$ כדי לקבוע את המשקל על הקשת בגרף. בגרף ממושקל ככל שהמשקל על קשת (u,v) יותר גדול כך המרחק הצמתים u ו- v יותר גדול. הערך $Connection(u, v)$ הוא מידה לרמת הקירבה בין 2 המילים, ולכן נקבע את משקלי הקשתות להיות ביחס הפוך לערך זה: - $1/Connection(u, v)$.

אלגוריתם 1

הרעיון של האלגוריתם הראשון הוא לחלק את כל המילים באנגלית ל- clusters ואז לקבוע כי למילה יש משמעויות שונות אם יש לה קשרים חזקים עם clusters שונים. כלומר כל cluster שלמילה יש קשר חזק איתו ייצג משמעות של המילה.

למשל נצפה כי למילה "עכבר" יהיו קשרים חזקים עם מילים השייכות ל- cluster של עולם המחשבים, וקשרים חזקים עם מילים ב- cluster של עולם החיות.

כמובן ששיטה זו היא נאיבית מדי, מפני שהיא מבצעת חלוקה גסה של השפה למילים, ועל כן האלגוריתם עלול לפצל מילים שהן חסרות משמעויות רבות מכיוון שהן נמצאות בהקשרים שונים, ומפני שהאלגוריתם מחפש פתרון גלובלי לכל המילים.

למשל האלגוריתם עלול לפצל את המילה תיק למספר מילים שונות שכולן תחת המשמעות של תיק גב, זאת מכיוון שתיק גב מופיע בהקשרים שונים (תיק גב לבית הספר, תיק גב לטיולים, תיק גב למשרד וכו...). היינו רוצים שהמילה תיק לא תקבל משמעות נפרדת בכל הקשר כזה מפני שמבחינתנו למילה תיק בכל ההקשרים האלו יש את אותה המשמעות.

ניתן לטעון כי הפיצול של המילה תיק לתתי המשמעויות של תיק גב יכולה לשפר את הייצוג של כל תת-משמעות, אך לשם כך דרוש corpus גדול שמתוכו תתבצע למידה של הייצוג הווקטורי של כל משמעות. בנוסף יש לזכור שלאחר פיצול המילים, בהינתן הופעה של מילה בטקסט נרצה לדעת לזהות איזו משמעות של המילה היא המשמעות המתאימה, וכבל שמספר המשמעויות של המילה גדל, יהיה יותר קשה לבצע משימה זו בהצלחה.

האלגוריתם לקביעת המשמעויות של המילים (אימון):

תהליך הלמידה:

1. ביצוע preprocessing על ה- corpus.
2. לכל זוג מילים, מחשבים את מידת הקשר בניהן.
3. מחשבים את m זוגות המילים בעלות הקשרים הכי חזקים. (m הוא פרמטר של המודל).
4. יוצרים גרף ממושקל G עם צומת לכל מילה מבין המילים מ-3, וקשת לכל זוג מילים מבין הזוגות ב-3, כאשר משקל הקשת הוא **ביחס הפוך** למידת הקשר בין המילים.
5. מבצעים clustering לפי אלגוריתם Louvain ומסירים קלאסטרים שגודלם קטן פי t מגודלו של הקלאסטר הגדול ביותר (t פרמטר של המודל), או קלאסטרים שגודלם קטן מ-min_cluster_size (גם פרמטר של המודל).
6. לכל מילה w המיוצגת ב-G:
   a. אתחל $x_1, x_2, \dots, x_k \leftarrow 0$.
   b. לכל קשת (w, u), עבור i האינדקס של ה- cluster אליו שייכת u, בצע $x_i += Connection(w, u)$
   c. עבור $x_{max} = \max\{x_1, x_2, \dots, x_k\}$ ולכל i כך ש- $x_i \geq x_{max} * d$, (d הוא פרמטר של המודל), קבע כי ה- cluster ה- i מייצג משמעות של w.

הסבר:

באלגוריתם אנחנו ראשית מבצעים חלוקה של כל המילים לקבוצות על ידי שימוש באלגוריתם קלאסטרינג על הגרף. מכיוון שהקשתות בגרף ממושקלות ביחס הפוך לקשר בין המילים, אז ניתן לייחס לחלוקה הזו משמעות של חלוקה לפי נושאים קשורים.

הסיבה לכך שאנחנו מוסיפים רק m קשתות לגרף ולא את כולן היא שרוב המילים הן בעלות קשר מאוד חלש, ואין טעם לחשב את הקלאסטרים על גרף מלא.

בשלב 6 אנחנו קובעים לכל מילה מהן המשמעויות שלה:

אנחנו מחשבים לכל מילה w ולכל קלאסטר C את מידת הקשר של w עם C :

$$Connection(C, w) = \Sigma_{u \in C} Connection(u, w)$$

לכל מילה u ב-C אנחנו מוסיפים למידת הקשר של w עם הקלאסטר את מידת הקשר של w עם u.

לאחר חישוב הקשר של המילה עם כל קלאסטר, נסנן את הקלאסטרים שהקשר איתם נמוך (על ידי הדרישה $x_i \geq x_{max} * d$) ונקבע כי כל קלאסטר שעבר את הסינון מייצג משמעות של המילה.

אופן סיווג המילים:

בהינתן הופעה של w בטקסט, נרצה לקבוע באיזו משמעות של w מדובר. לשם כך נסתכל על החלון שבו מופיעה, נסמן את החלון הזה ב- $w_1, w_2, \dots, w_n$. לכל קלאסטר C שמייצג משמעות של w נמצא את מידת הקשר בין הקלאסטר לחלון, שמוגדר להיות סכום הקשרים בין המילים בחלון ל- C כפול מידת הקשר של w עם C.

כלומר לכל קלאסטר C מגדירים:

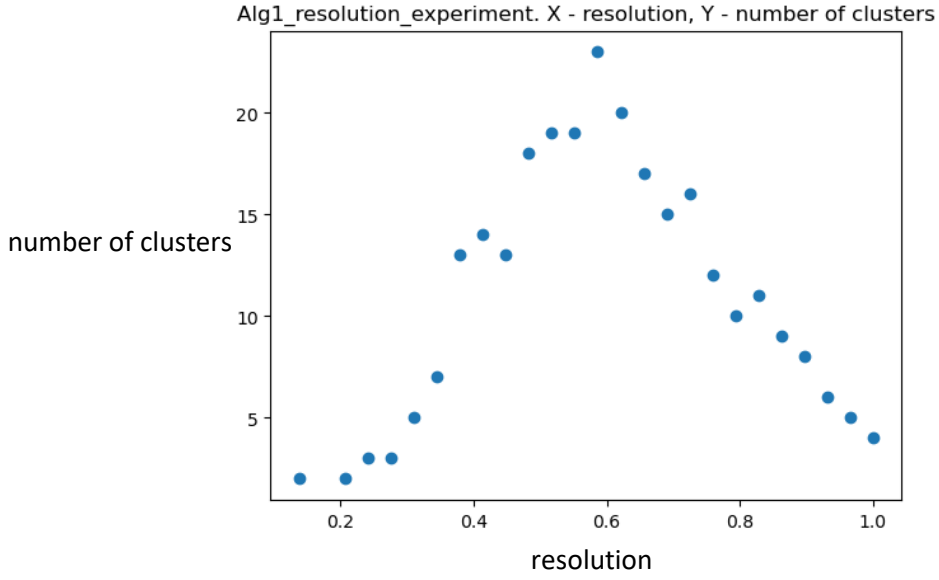
$$Connection(C, Window) = \Sigma_{i=1}^{n} Connection(C, w_i) * Connection(C, w)$$

נקבע כי משמעות של w בהופעה זו היא המשמעות המתאימה לקלאסטר שקיבל $Connection$ מקסימלי עם החלון.

ניסויים:

הפרמטרים המרכזיים במודל הם הפרמטר resolution של אלגוריתם הקלאסטרינג Louvain, והפרמטר m – מספר הקשתות המוכנסות לגרף.

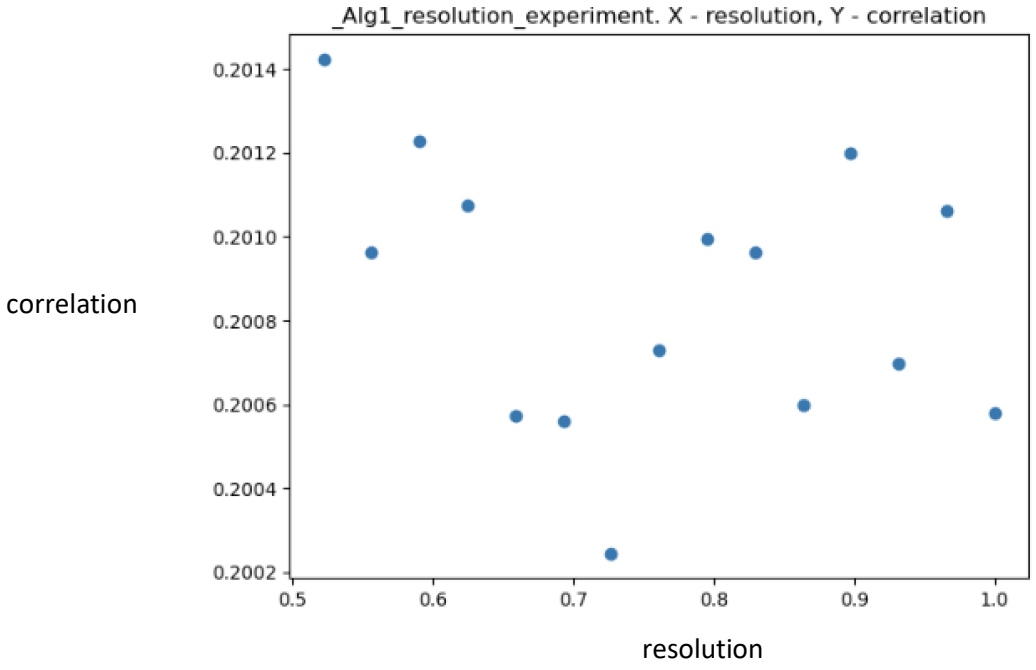
הפרמטר resolution משפיע על הגדלים של הקלאסטרים ועל תוכנם.
לאחר ביצוע הקלאסטרינג באמצעות אלגוריתם louvain, המודל מתייחס רק לקלאסטרים שאינם קטנים מדי ביחס לגודלו של הקלאסטר הגדול ביותר.
ניתן לראות את ההשפעה של הפרמטר resolution על מספר הקלאסטרים ההללו בגרף הבא:



ניתן לשים לב כי כצפוי עליה בערך ה- resolution מביאה לגדילה במספר הקלאסטרים, שנובעת מכך שהאלגוריתם מותאם למציאת קלאסטרים קטנים יותר. עבור ערכים גבוהים של resolution מספר הקלאסטרים שהמודל מתייחס אליהם קטן, וזה נובע מכך שהאלגוריתם Louvain מוצא קלאסטרים שגודלם קטן min_cluster_size.

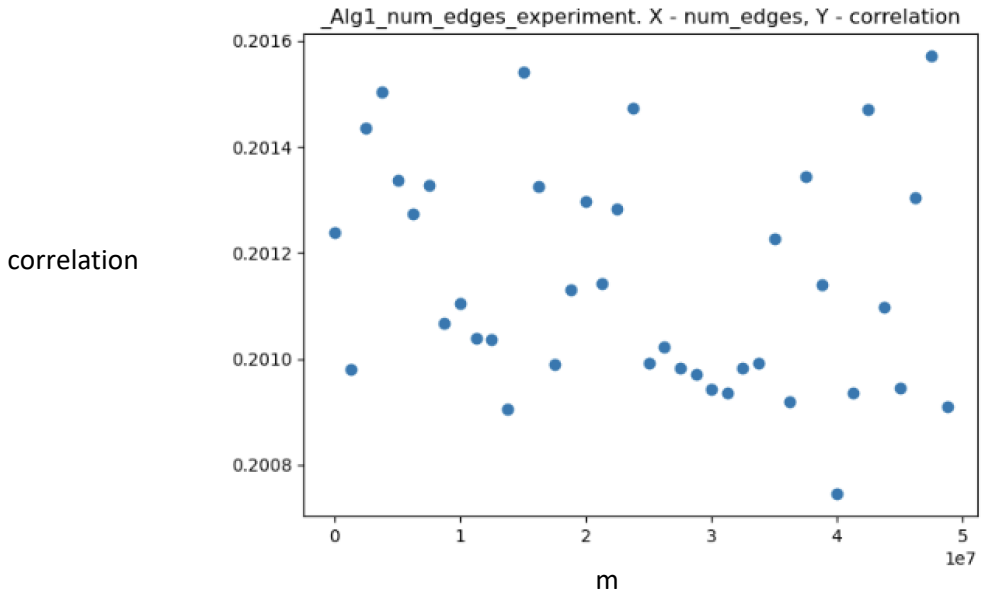
ערכים שונים של הפרמטרים גורמים ליצירת קלאסטרים שונים בגודלם ובתכולתם. לפי אופן פעולת המודל, כל משמעות שנקבעת למילה מתאימה לקלאסטר, ולכן נצפה שערכים שונים של הפרמטר

ישפיעו בצורה רבה על ביצועי המודל. ניתן לראות את ההשפעה של הפרמטר על הקורלציה בגרף הבא:



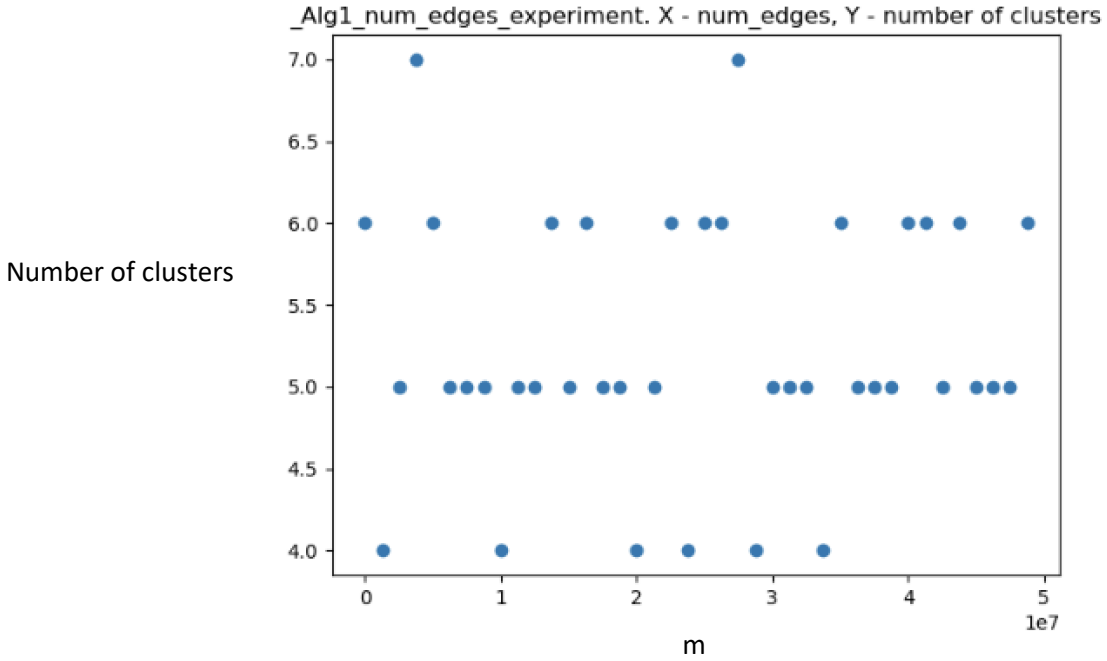
קשה לקבוע לפי הגרף מה הערך הטוב ביותר עבור הפרמטר, מפני שהקורלציה משתנה בצורה לא עקבית והשינויים הם יחסית קטנים. ייתכן שהשינויים נובעים מאי הדטרמיניסטיות שבאלגוריתם Doc2Vec, טענה שאפשר לבדוק על ידי הרצת האלגוריתם מספר רב של פעמים עבור כל ערך של הפרמטר resolution, מה שייתן לנו תוצאות יציבות יותר מבחינה סטטיסטית.

הפרמטר m הוא מספר הקשתות המוכנסות לגרף. היה ניתן לצפות כי עבור ערכים גבוהים של m היינו מקבלים קורלציה גבוה יותר כי הגרף מכיל יותר מידע על היחסים בין המילים, אך ניתן לראות בגרף הבא כי זה לא המצב. הגרף מתאר את ערכי הקורלציה שהתקבלו (ציר y) עבור ערכי m שונים (ציר x), כאשר הערכים המסומנים על ציר x הם כפולות של 1e7 קשתות.



הגרף מצדיק את ההנחה כי אין טעם להכניס לגרף את כל הקשתות מכיוון שהקשר בין רוב המילים הוא מאוד חלש, ואין הבדל מהותי בין אי-קיומה של קשת בגרף, לקיום של קשת עם משקל גבוה מאוד.

הגרף הבא מבטא את מספר הקלאסטרים שהתקבלו כפונקציה של מספר הקשתות, ומכיוון שלא ניתן לראות קשר בין שני אלו, גם הגרף הזה מחזק את ההנחה.



אלגוריתם 2

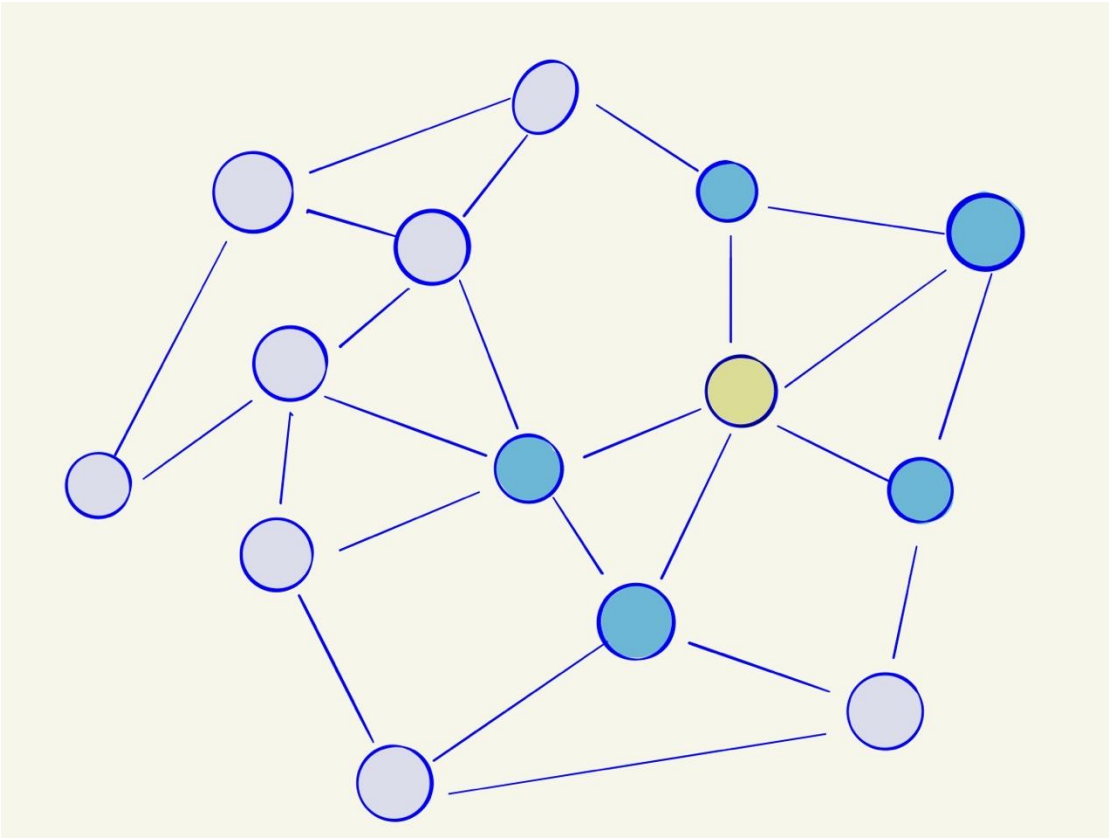
הרעיון באלגוריתם זה הוא לשפר את אלגוריתם 1 על ידי ביצוע של חלוקות רגישות יותר. כלומר באלגוריתם החדש אנחנו מקבלים חלוקה מתאימה לכל מילה, ולא חלוקה יחידה עבור כל המילים.

תחילה ניצור גרף G דומה לגרף באלגוריתם 1: כל צומת בגרף מייצגת מילה, משקל הקשת בין 2 מילים מייצג את מידת הקשר בניהן, לכל מילה w נגדיר ב- $Closest_m(w)$ את m המילים שיש ל-w את הקשר הכי חזק איתן. לכל מילה $u \in Closest_m(w)$ נוסיף לגרף G את הקשת (w,u) עם המשקל - $1/Connection(w,u)$. כלומר באלגוריתם הנוכחי, לכל מילה מופיעים בגרף m הקשתות שמייצגות את הקשרים הכי חזקים שלה (לעומת אלגוריתם 1 שבו מוסיפים את m הקשרים הכי חזקים).

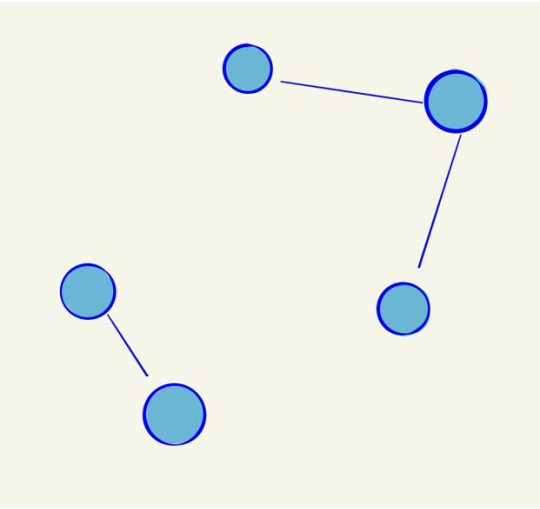
לכל מילה w נגדיר תת-גרף G_w, שהצמתים בו הם הצמתים השכנים של w (בלי w), והקשתות הן הקשתות בין צמתים אלו בגרף G.

נמצא את רכיבי הקשירות של תת-גרף זה. לכל רכיב קשירות שעומד בקריטריונים שנגדיר בהמשך, נקבע כי הוא מייצג משמעות של המילה w.

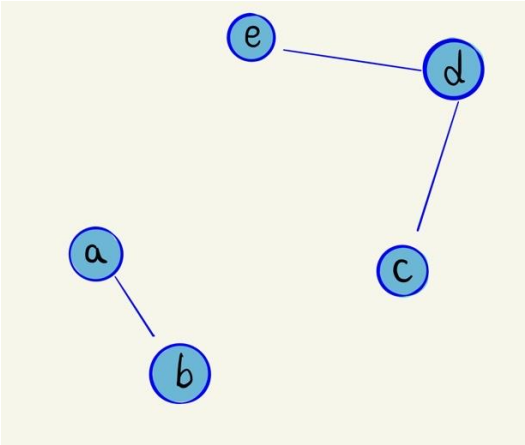
לדוגמה, נסתכל על הגרף הבא:



אם הגרף הנ"ל הוא G, ו- w היא המילה המתאימה לצומת המסומן בצהוב, אז הגרף G_w יכיל את הצמתים עם המילוי הכחול:



ומספר רכיבי הקשירות בגרף G_w הוא 2.
נסמן את הצמתים בשביל המשך ההסבר.



הסיבה להגדרה זו של משמעויות המילה היא שאם בגרף החדש שהגדרנו עבור w יש רכיבי קשירות שונים, אז בין מילים מרכיבי קשירות שונים יש **קשר חלש**, למשל בדוגמב בין המילים המתאימות לצמתים a ו- c יש קשר חלש יחסית, כי העובדה שאין בניהם קשת מעידה על כך ש- $c \notin Closest_m(a)$ ו- $a \notin Closest_m(c)$. באופן דומה, בין מילים באותו רכיב קשירות יש **קשר חזק,** למשל בדוגמה מתקיים $a \in Closest_m(b)$ או $b \in Closest_m(a)$.
בנוסף ל- w יש קשר חזק עם כל רכיב קשירות מפני שהוא מורכב מהשכנים של w.

היתרון מההגדרה החדשה של הקשתות ב- G ומאופן ההגדרה של המשמעויות עבור מילה, היא שכעת כל מילה מקבלת חלוקה שמתאימה לה באופן ספציפי ומאפיינת את מובנים השונים המתאימים למילה, ולא מקבלת משמעויות על פי חלוקה גלובלית.

נגיד כי החלוקה לפי רכיבי הקשירות של תת הגרף עבור מילה w היא "טובה" (כלומר מייצגת משמעויות של המילה) אם מידת הקשר של כל רכיב בחלוקה עם w פחות או יותר שווה, כלומר לכל C1,C2 בחלוקה, מתקיים:

$$\alpha \leq \frac{\min\{Connection(w, C_2), Connection(w, C_1)\}}{\max\{Connection(w, C_2), Connection(w, C_1)\}}$$

עבור $0 < \alpha < 1$ כלשהו (פרמטר של המודל), וכשאר

$$Connection(C,w) = \Sigma_{u \in C} Connection(u,w)$$

האלגוריתם לקביעת המשמעויות של המילים (אימון):

נסמן ב- C את ה- corpus שהתקבל לשם למידה.

תהליך הלמידה:

1. ביצוע preprocessing על C.
2. לכל זוג מילים, חשב את מידת הקשר בניהן.
3. חישוב $Closest_m(w)$ לכל מילה w.
4. יצירת גרף ממושקל G עם צומת לכל מילה w, ועם הקשתות המושרות על ידי הקבוצות $Closest_m(w)$ שהוגדרו ב-3, כאשר משקל הקשת (w,u) הוא $\frac{1}{Connection(w,u)}$.
5. לכל מילה w:
   a. צור תת-גרף G_w של G בו הצמתים הם השכנים של w ב- G.
   b. חשב את רכיבי הקשירות של G_w: $C_1, C_2, \dots, C_k$.
   c. חשב $M = \max\{Connection(C_i,w)\}_i$
   d. לכל $1 \le i \le k$ המקיים
   $$\alpha * M \le Connection(w, C_i)$$
   קבע כי C_i מייצג משמעות של w.

* הסיבה לקביעת המשקלים ביחס הפוך לקשר מוסבר בחלק המבוא לאלגוריתמים הגרפיים.

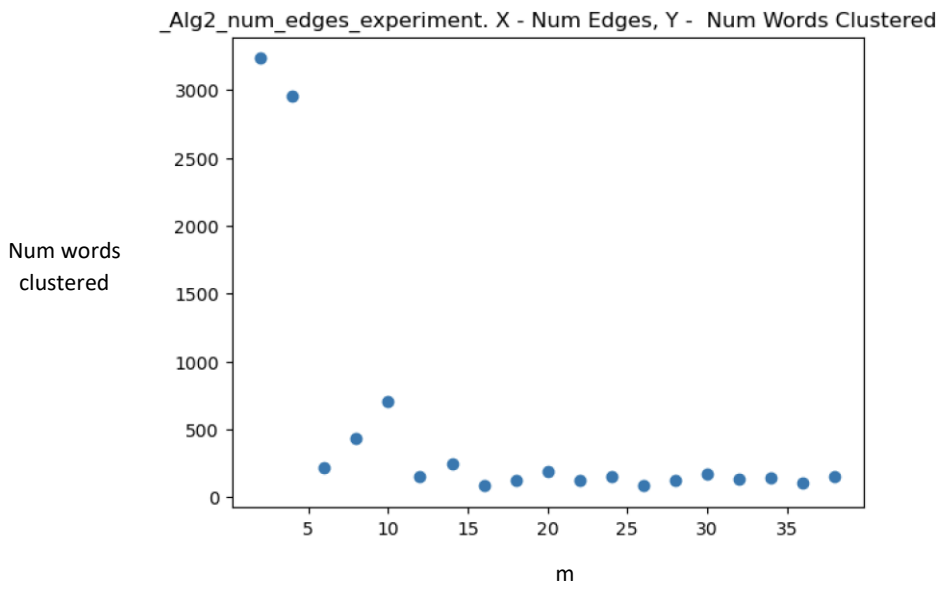
אופן סיווג המילים:

באופן דומה לסיווג המילים של אלגוריתם 1.

ניסויים:

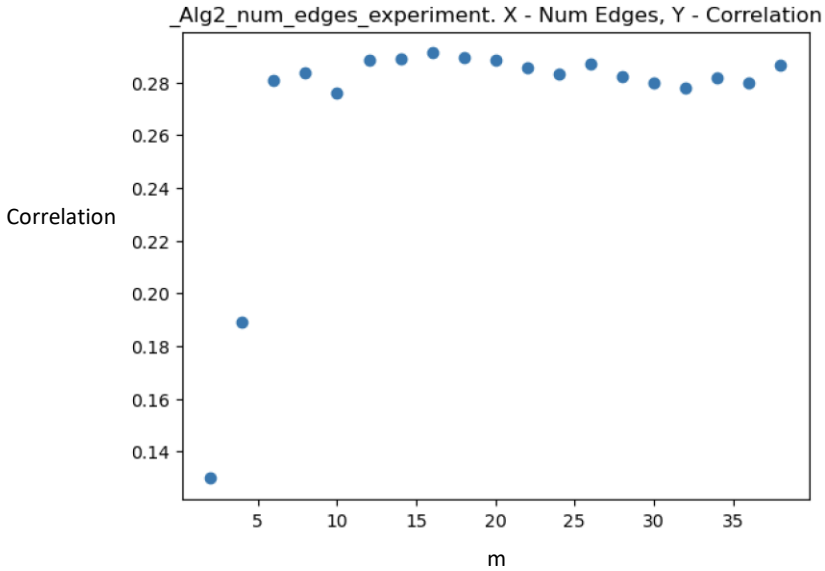
הפרמטר שמשפיע בצורה הכי משמעותית על המודל הוא m – מספר הקשתות שאנחנו מוסיפים לגרף עבור כל מילה.
ערך נמוך של m יגרום לכך שבגרף יהיו מעט קשתות, לכל מילה יהיו לכל הפחות m שכנים, אך הסיכוי שאלו יהיו באותו רכיב קשירות יהיה נמוך יותר כי הגרף דליל יותר, לכן נצפה לקבל מספר רכיבי קשירות שהוא קרוב יותר ל- m.
ערך גבוה של m פוטנציאלית יכול להביא לכך שיהיו הרבה מאוד משמעויות לכל מילה, אך פרקטית מכיוון שבגרף יהיו יותר קשתות אז מספר רכיבי הקשירות יקטן בכל תת-גרף ביחס למספר הצמתים בו. למשל אם $m = |V|(|V|-1)$ אז בכל תת גרף של מילה יהיה רכיב קשירות יחיד כי כל המילים מקושרות.

ניתן לראות בגרף הבא את הקשר בין m לבין מספר המילים שקיבלו משמעויות שונות (שזה אומר שבתת גרף המתאים להן היו לפחות שני רכיבי קשירות שעמדו בקריטריונים להיחשב משמעות).



בציר x מופיע הערך של m ובציר y מופיע מספר המילים שקיבלו יותר ממשמעות אחת.
ניתן לראות שאכן מספר כאשר m מאוד קטן (2 או 4) אז מספר המילים שקיבלו יותר ממשמעות אחת גבוה מאוד כפי שציפינו. באופן מפתיע כאשר מעלים את מספר הקשתות ל- 6 יש ירידה מקצת פחות מ- 3000 מילים ל-בערך 250 מילים.

את ההשפעה של m על הקורלציה ניתן לראות בגרף הבא:



באופן שמתאים לגרף הקודם, כאשר מספר הקשתות קטן הרבה מאוד מילים קיבלו יותר ממשמעות אחת, מה שגרם לירידה בביצועים של המודל.

אף תוצאה של המודל לא עוקפת את התוצאה הממוצעת של ה- Doc2Vec הרגיל, שמגיע לערך קורלציה של 0.3 בממוצע.

אלגוריתם 3:

באלגוריתם 3, המטרה היא לזהות עבור זוג מילים w_1, w_2 שלכל אחת מהן יש הקשר משותף עם מילה w, האם ההקשר הוא משותף לשלושתן, ולפיכך לקבוע מהן המשמעויות של w.

למשל למילה סכין (בתפקיד w) יש הקשר משותף עם המילים עגבניה ומלפפון, וההקשר משותף לשלושתן. לעומת זאת למילה סכין יש הקשר משותף עם המילים עגבינה ודם, אך ההקשר לא משותף לשלושתן.

עבור מילים w, w_1, w_2 נסמן ב- $\#(w, w_1, w_2)$ את מספר ההופעות של המילים ביחד (באותו חלון), וב- $\#(w, w')$ את מספר ההופעות של w, w' ביחד.

מכיוון שבכל חלון שבו מופיעות שלושת המילים, בהכרח מופיעות w_1 ו- w_2 אזי

$$\#(w, w_1, w_2) \leq \#(w, w_i), i \in \{1,2\}$$

ולכן

$$\frac{\#(w, w_1)}{\#(w, w_1, w_2)} \geq 1, \frac{\#(w, w_2)}{\#(w, w_1, w_2)} \geq 1$$

ולכן

$$\frac{\#(w, w_1)}{\#(w, w_1, w_2)} * \frac{\#(w, w_2)}{\#(w, w_1, w_2)} = \frac{\#(w, w_1) * \#(w, w_2)}{\#(w, w_1, w_2)^2} \geq 1$$

נסמן $\alpha = \frac{\#(w,w_1)*\#(w,w_2)}{\#(w,w_1,w_2)^2}$, או α_{w,w_1,w_2} במידה ולא ברור מההקשר על אילו מילים מדובר.

אם α קרוב ל- 1 אז הביטויים $\frac{\#(w,w_i)}{\#(w,w_1,w_2)}$ עבור $i \in \{1,2\}$ קרובים ל-1, ולכן כמעט בכל הופעה משותפת של w ו- w_1 מופיעה גם w_2, וכמעט בכל הופעה משותפת של w ו- w_2 מופיעה גם w_1.

כעת נניח של- w יש הקשר משותף עם w_1 והקשר משותף עם w_2 . אז נוכל להתייחס ל- α כעל מדד למידת הדמיון בין ההקשרים:

אם α נמוכה (קרובה ל- 1) אז בסבירות גבוהה הופעה משותפת של w ו- w_1 היא הופעה משותפת של w, w_1 ו- w_2, ולכן ההקשר המשותף ל- w עם w_1 הוא גם משותף ל- w_2.

לכל מילה נגדיר גרף $G_w = (V, E)$.

$$V = \{v \in W | \#(v, w) > 0\}$$

כלומר כל צומת בגרף מתאימה למילה שהופיעה בחלון ביחד עם w.

$$E = \{(u, v) \in W \times W | \#(u, v, w) > 0\}$$

כלומר יש קשת בין 2 מילים בגרף אם הן הופיעו באותו חלון ביחד עם w.

כאשר W היא קבוצת כל המילים ב- *corpus* (אחרי שלב ה- *preprocessing*).

נגדיר משקל על הקשתות: $weight(u, v) = \alpha_{w,w_1,w_2}$.

את המשמעות של המשקל אפשר לתאר כך:

בהנחה שלכל אחת מהמילים u ו- v יש הקשר משותף עם w, אם הקשת בין u ל- v היא קלה (α קרובה ל-1) אז המשמעות היא דומה.

כדי לקיים את ההנחה שלמילים יש הקשר משותף עם w, נוסיף לגרף רק קשתות של מילים u שמקיימות:

$$\#(u,w) > p * \min\{\#(u), \#(w)\}$$

עבור $p < 1$ כלשהו (פרמטר של המודל), וכאשר $\#(u)$ הוא מספר ההופעות של u בכל ה-*corpus*.
התנאי הנ"ל גורם לכך שיופיעו בגרף רק מילים רלוונטיות עבור w.

לפיכך הגדרת הקשתות ב- G_w היא:

$$E = \{(u,v) \in W \times W | \#(u,v,w) > 0, \frac{\#(u,w)}{\min\{\#(u), \#(w)\}} > p, \frac{\#(v,w)}{\min\{\#(v), \#(w)\}} > p\}$$

כדי להבין את המשמעות של קלאסטר בגרף G_w נניח כי לכל קלאסטר $C = \{u_1, \dots u_k\}$, לכל זוג צמתים $u, v \in C$, יש קשת בניהן והיא בעלת משקל נמוך (זה פחות או יותר מה שאלגוריתם הקלאסטרינג קובע לפיו את הקלאסטרים).
לפיכך לכל $u, v \in C$, ל- v ול- u יש הקשר משותף עם w (כי הן מקיימות את תנאי p), וההקשר הזה דומה, כי המשקל על הקשת (u,v) נמוך, ומפה נסיק כי הקלאסטר מייצג הקשר של w.

האלגוריתם לקביעת המשמעויות של המילים (אימון):

1. בצע preprocessing על הקורפוס.
2. לכל $w, w_1, w_2 \in W$ חשב $\#(w), \#(w_1, w_2), \#(w, w_1, w_2)$.
3. לכל מילה $w \in W$:
   a. צור גרף G_w.
   b. חשב קלאסטרים $c_1, \dots, c_k$ לפי אלגוריתם Louvain.
   c. עבור $i = 1 \dots k$:
      i. אם $|c_i| >$ min_cluster_size קבע כי c_i מייצג משמעות של w.

אופן סיווג המילים:

באותו אופן כמו באלגוריתמים 1 ו-2. הפעם הגרף שמתואר באלגוריתמים 1 ו- 2 אינו נבנה כחלק משלב האימון, ולכן אנו מייצרים את הגרף במיוחד בשביל ביצוע פעולת הסיווג.

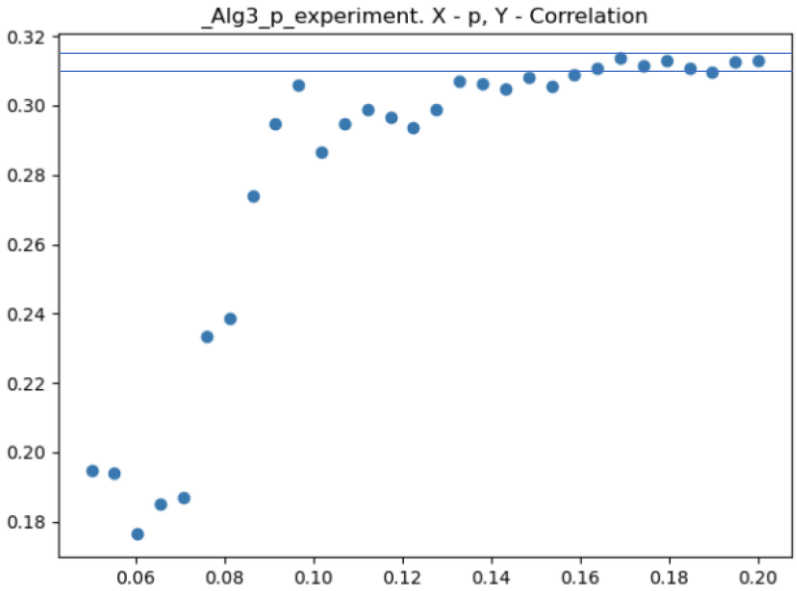
ניסויים:
ביצענו ניסויים על הפרמטרים *resolution* ו- *p*.
הפרמטר *p*, כאמור, קובע את מידת הקשר המינימלי בין שתי מילים w ו- u כדי שהן יופיעו האחת בגרף של השנייה. בפרט *u* תופיע בגרף ש- G_w ו- w תופיע ב- G_u אם ורק אם מתקיים $\#(u,w) > p * \min\{\#(u), \#(w)\}$. כפי שהוסבר, המטרה של הפרמטר היא להבטיח שלשתי המילים יש הקשר משותף.

לפיכך עבור ערכים קטנים של *p*, הנחת ההקשר המשותף בגרף היא יותר חלשה, כלומר יותר מילים יוגדרו כבעלות הקשר משותף עם מילים אחרות, מה שיביא לכך שיקבעו משמעויות למילים

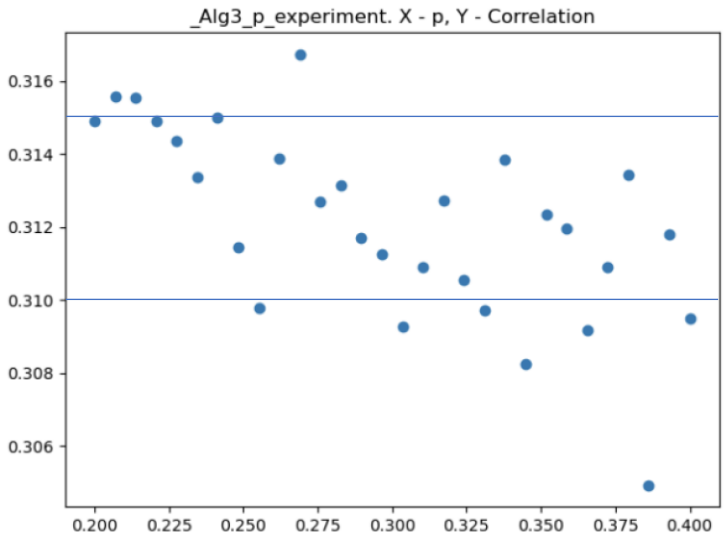
על פי מילים פחות קשורות אליהן, או במילים אחרת המשמעויות יזוהמו על ידי הקשרים לא רלוונטיים.
בהתאם ערכים גבוהים של *p* יגרמו לקביעת המשמעויות להיות יותר איכותית מהבחינה הזו שכל מילה שמופיע בגרף, ולכן "לקחה חלק בקביעת המשמעויות", היא מילה שיש לה קשר חזק עם המילה המקורית, אך מספר המילים בגרף קטן ולכן ההחלטה פחות מבוססת.

בנוסף ערכים גדולים של *p* יכולים להוות סינון של המילים שמקבלות יותר ממשמעות אחת. ניקח לדוגמה *p=0.2*, כדי שמילה תקבל יותר ממשמעות אחת עבור ערך זה, צריכות להיות קיימות 2 קבוצות מילים זרות *A* ו- *B*, כך שלכל מילה u בכל אחת מהקבוצות מתקיים: $(u,w) > 0.2 * \min\{\#(u), \#(w)\}$, כל קבוצה היא לפחות בגודל min_cluster_size והקשרים בין המילים בתוך כל קובצה הם מספיק חזקים, ובין הקבוצות הן מספיק חלשים, כדי שאלגוריתם הקלאסטרינג יקבע כי המילים ב- *A* הם באותו הקלאסטר והמילים ב- *B* הם באותו הקלאסטר.
שלושת התנאים האלו הם יחסית נדירים, כי התנאי $(u,w) > p * \min\{\#(u), \#(w)\}$ קובע שכל מילה שמופיעה באחת הקבוצות מופיעה לפחות פעם אחת על כל 5 הופעות של w , או ההפך.
לכן ערכים גבוהים של *p* מורידים את מספר המילים המתאימות לפיצול. המילים שנשארות מתאימות לפיצול הן מילים שקל לקבוע כי הן שייכות ליותר ממשמעות אחת, מפני שיש מספר מילים שמופיעות הרבה מאוד בסביבתן, ואותם ניתן לחלק בקלות לקבוצות שונות.

בגרפים הבאים ניתן לראות את הקשר בין הערך של *p* ובין הקורלציה:



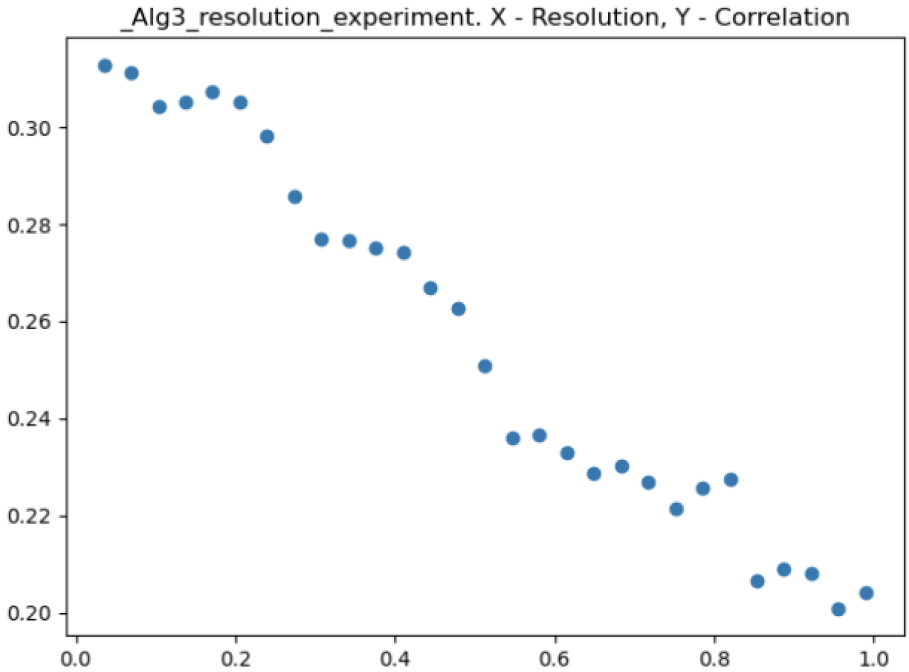
0.315



0.315

הגרף העליון מכיל ערכים קטנים מ- 0.2 של *p*, וכגרף התחתון מכיל ערכים גדולים מ- *0.2*.
הקו הכחול העליון בשני הגרף מסמן את ערך הקורלציה 0.315 והתחתון את הערך *0.31*.
ניתן לזהות עליה בקורלציה עד הערך *p=0.16* ואז ירידה קלה. ניתן להניח בהתאם להסבר הקודם על ערכים גבוהים של *p*, כי היחס בין הפיצולים המיותרים של משמעויות של מילים, לפיצולים המוצדקים מגיע עבור ערך זה למקסימום (פחות או יותר, יש עוד הרבה שיקולים אחרים שמשפיעים על הקורלציה כמו אילו מילים מופיעות ב- *dataset* שאיתו ביצענו את המבחן), והחל מערך זה גם מילים שהיינו רוצים שיקבלו יותר ממשמעות אחת לא קיימו את התנאי לפיצול ולכן ערך הקורלציה ירד מעט.

הפרמטר *resolution* כמו שהסברנו, הוא פרמטר של אלגוריתם הקלאסטרינג *Louvain*. הוא משפיע על גודל הקלאסטרים ועל תוכנם.



ניתן לראות את השפעת האלגוריתם בגרף הבא, ציר *x* הוא ערכו של הפרמטר, וציר *y* הוא ערך הקורלציה. עבור ערכי *p* שונים מתקבל גרף שונה. למשל הניסוי על *p* התבצע עם ערך *resolution=0.6*, ועבורו קיבלנו ערך מקסימלי של 0.317, בעוד שבגרף זה עבור *resolution=0.6* מקבלים ערך קורלציה 0.23.

אלגוריתם 3 משיג שיפור קטן לעומת *Doc2Vec* הרגיל. הערך הממוצע על פני 20 חזרות של ה-*Doc2Vec* הרגיל שהופעל עבור אלגוריתם זה הוא בערך 0.305, ואילו האלגוריתם הגיע לערכים גבוהים מ- 3.1 עבור סדרה של פרמטרים במהלך ביצוע הניסויים, עם ערך מקסימלי של *3.17*, כאשר כל פעם האלגוריתם הורץ פעם אחת עם פרמטרים שונים.

<u>עבודה על קורפוס ויקיפדיה:</u>

במהלך העבודה גילינו כי הקורפוס *Brown* אינו מביא לתוצאות טובות, חלק מכך משום שהמילים אותם האלגוריתם מנסה לקבץ אינם מתאימים לקיבוץ – לדוגמא המילה ממנה עלה הרעיון לפרויקט *mouse* אינה מופיעה מספיק פעמים בשביל שהאלגוריתם ינסה לקבץ אותו. לכן ניסינו להתאים על האלגוריתם לעבודה על *Big Data* מסוג ויקיפדיה.

במהלך העבודה ניסינו להתשתמש בשיטות שונות לייעל את המערכת ולאפשר לה להתמודד עם כמות זו של מידע. למרות כישלוננו להגיע לאלגוריתם שייספק אותנו מבחינת זמני ריצה השתמשנו בדרכים הבאות להתמודדות עם בעיות שונות:

- פיצול המילונים השונים לגדלים קבועים – כך בכל פעם ניתן להשתמש בחלק מהמילון המכיל חלק מהמידע ולא לטעון למערכת מילונים גדולים מידי.
- שימוש ב-*multiprocessing* – למדנו להשתמש במיקבול על מנת להתמודד עם מספר המילונים הגדול שיצרנו וכמות המידע.
- *Principal Component Analysis (PCA)* – שיטה לצמצום מימד הוקטורים על ידי מציאת קורולציה בינם[3]. השתמשנו בשיטה זו בשביל לצמצם את כמות הזיכרון הנצרך על ידי אלגוריתם הקיבוץ ועל מנת להתמודד עם קללת המימדיות (*The Curse of Dimensionality[4]*) אשר גורמת למרחקים בין וקטורים במימדים גבוהים להיות חסרי משמעות.
- לימוד על אלגוריתם *HDBSCAN* להאצת אלגוריתם הקיבוץ.
- דגימת משפטים – על מנת להתמודד עם זמן ריצה ארוך באלגוריתם הקיבוץ דגמנו אחוז מסויים מהמשפטים ורק עליהם ביצענו את אלגוריתם הקיבוץ.
- עבודה על שרתים חיצוניים – למדנו איך לעבוד על שרתים חיצוניים על מנת להריץ את האלגוריתם על חומרה חזקה יותר.

<u>סיכום ודיון:</u>

במהלך הפרויקט הצאנו ארבעה אלגוריתמים לשיפור ביצועי *Doc2Vec* למציאת דימיון בין משפטים. האלגוריתמים השונים השתמשו במאגר המידע *Brown* כקלט לבעיה וביצעו עליו את הלמידה של כל אלגוריתם. לאחר מכן ערכנו ניסויים על כ-8000 זוגות של משפטים קצרים אשר דורגו על ידי בני אדם והשוונו את הקורולציה בין תוצאות האלגוריתמים לתוצאות האמת שנקעו על ידי אנשים.

נציין כי מלבד האלגוריתם הרביעי (אלגוריתם גרפי 3) אף אחד מהאלגוריתמים המוצעים לא הציג תוצאות המראות שיפור מביצועי *Doc2Vec*. ניתן לראות כי אלגוריתם ה-*WordSplitter* לא מצליח להשיג שיפור במציאת דימיון בין משפטים, זאת למרות הפיצולים השונים שנעשו באלגוריתם. נשים לב כי עבור מקדמי פיצול 3-2 אנו כן מצליחים להשוות את האלגוריתם לביצועי *Doc2Vec*. אלגוריתם 1 הגרפי מגיע לתוצאה הגבוה ביותר שלו כאשר מספר הקלאסטרים שהוא מצא הוא 1, ובמצב זה הוא לא משנה שום דבר לעומת *Doc2Vec* רגיל. אלגוריתם 2 הגרפי גם כן לא הצליח להשיג שיפור, עבור ערך נמוך של *m* (מספר הקשתות), הוא מפצל יותר מדי מילים, מה שפוגע בביצועים שלו, וככל ש- *m* גדל האלגוריתם מגיע לערכי קורלציה קרובים יותר לאלו של *Doc2Vec*, אך לא עוקף אותן. נראה שאלגוריתם 3 הגרפי הצליח להשיג שיפור קטן, אך השיפור לא משמעותי ולא נבדק בצורה מספקת מבחינה סטטיסטית.

ניתן לראות שאלגוריתם 3 מצליח לנצל את הפשטות של ספירות מילים כדי להגדיר גרף שמבטא בצורה טובה את מידת הקשר של מילה שלישית לקשר שבין 2 מילים.

נראה כי האלגוריתמים השונים אינם מצליחים לשפר את הביצועים של *Doc2Vec*, אם כי הבדיקה נעשתה על משימה בודדת – מציאת דימיון בין משפטים קצרים – וייתכן שעל משימות נוספות נקבל תוצאות שונות. ניתן לבצע מבחנים לבדיקת סנטימנט הטקסט ולראות האם המודלים שבנינו מצליחים להציג שיפור.

כפי שצויין לעיל, בדיקת האלגוריתמים על מאגרי מידע גדולים ואיכותיים יותר עשויה לתרום רבות ללמידת האלגוריתמים ולהביא לביצועיים טובים יותר ולכן אנו ממליצים להריץ את אלגוריתמים עלו על מאגרי מידע שכאלה ולשחזר את הניסויים שנערכו לבדיקת השינויים.

<u>מאגר המידע לניסויים:</u>

השתמשנו במאגר המידע *Semantic Textual Similarity 2012-2017 Dataset*.
מאדר זה משווה בין 8628 זוגות של משפטים.
משפטים אלה מחולקים לנושאים באופן הבא:

- חדשות: 4299 משפטים.
- כותרות: 3250 משפטים.
- פורומים: 1079 משפטים.

לכל זוג משפטים יש ציון דמיון בטווח 0-5 אשר בני אדם נתנו לו.

<u>רפרנסים:</u>

1. https://arxiv.org/pdf/1507.07998.pdf
2. מאגר לטסטים:

http://ixa2.si.ehu.eus/stswiki:
Eneko Agirre, Daniel Cer, Mona Diab, Iñigo Lopez-Gazpio, Lucia Specia. Semeval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. Proceedings of SemEval 2017.
The scores are released under a "Commons Attribution - Share Alike 4.0 International License" http://creativecommons.org/licenses/by-sa/4.0/

3. https://en.wikipedia.org/wiki/Principal_component_analysis
4. https://en.wikipedia.org/wiki/Curse_of_dimensionality