**עיבוד שפה טבעית - תרגיל בית רטוב 2**

מגישים: אור קרופניק 302629027

רואי סינוף 201542982

**אימון**

כללי

1. הן עבור המודל הבסיסי והן עבור המודל המתקדם, עבור כל המאפיינים השתמשנו במבנה נתונים מסוג מילון (או מילון של מילונים). לדוגמה:

* p\_word נשמר כמילון שבו המפתחות הן המילים שמשמשות כ"אבות" והערך שלהן הוא מס' תת-המאפיין
* p\_word, c\_word, c\_pos נשמר כמילון שבו המפתחות הן המילים שמשות כ"אבות". עבור כל מפתח כזה, הערך הוא מילון נוסף שבו המפתחות הן המילים שהיו "בנים" של "אב" זה (כלומר יש קשת מ-p\_word ל-c\_word) וערכיהן הוא מספרי הפיצ'רים.

1. במסגרת האימון השתמשנו בחבילות הקוד הבאות:
   1. numpy - עבור וקטור המשקולות.
   2. random – על מנת לבצע shuffle על המשפטים אותם לומד הפרספטרון.
   3. edmonds – על מנת לקרוא לפונקציית ה-mst של Chu-Liu-Edmonds.
   4. pickle – כדי לשמור את הפיצ'רים שיצרנו על מנת שנוכל לטעון אותם בעתיד מבלי לבצע parsing מחדש.
2. במהלך הפירסור אנחנו יוצרים אובייקטים מסוג משפט, הכוללים בין היתר רשימה של אובייקטי קשתות וכן היסטוגרמה של הפיצ'רים שנדלקים בעץ – דבר זה שימש אותנו בהמשך באלגוריתם הפרספטרון (ראה הערה 4).
3. באלגוריתם הפרספטרון, עבור כל משפט שקיבלנו יצרנו 2 גרפים שלמים: אחד הכולל את הפיצ'רים של קשת ספציפית והשני הכולל את סכום משקלי הפיצ'רים הללו. את שלב עדכון המשקלים ביצענו **תמיד** (גם אם לא טעינו בעץ שפרשנו), כאשר בפועל אם צדקנו בפרסור הרי שחיסרנו וחיברנו אותם ערכים. חיבור הערכים בוצע ע"י שימוש בהיסטוגרמת הפיצ'רים שצויינה בסעיף 3, בעוד שחיסור הערכים בוצע ע"י חיסור 1 בוקטור המשקלים עבור כל הפיצ'רים המתאימים לקשת זו בגרף הפיצ'רים.

מודל בסיסי

להלן סיכום כמויות הפיצ'רים במודל הבסיסי:

|  |
| --- |
| 1 : 9993  2 : 8876  3 : 37  4 : 15908  5 : 14162  6 : 45  8 : 31314  10 : 33936  13 : 749  total : 115020 |

הזמן שלקח להריץ את הלימוד עבור המודל הבסיסי על לפטופ בעל 4 ליבות, מעבד i7 ו-RAM של 16GB הוא:

|  |  |
| --- | --- |
| מס' האיטרציות | זמן הלימוד |
| 20 | 0:20:31 |
| 50 | 1:08:00 |
| 80 | 1:44:00 |
| 100 | 2:05:00 |

מודל מתקדם

במסגרת המודל הוספנו פיצ'רים המתייחסים לאינדקס המילים בקשת (הן ה"אבא" והן ה"בן"), למרחק בין המילים בקשת, לגודל המשפט הכולל, למילים המופיעות לפני / אחרי המילים שבקשת וכן ל"אחים" של ה"בן". חלק מהפיצ'רים ממש מחקנו מאחר שפגעו בתוצאות בצורה ברורה (לדוגמה כמות ה"אחים"), ועל אחרים בסופו של דבר החלטנו לוותר, אך השארנו כתובים (מאחר שהם לא פגעו בביצועים, אך גם לא שיפרו). להלן הפיצ'רים אשר בסופו של דבר השתמשנו במסגרת מודל זה:

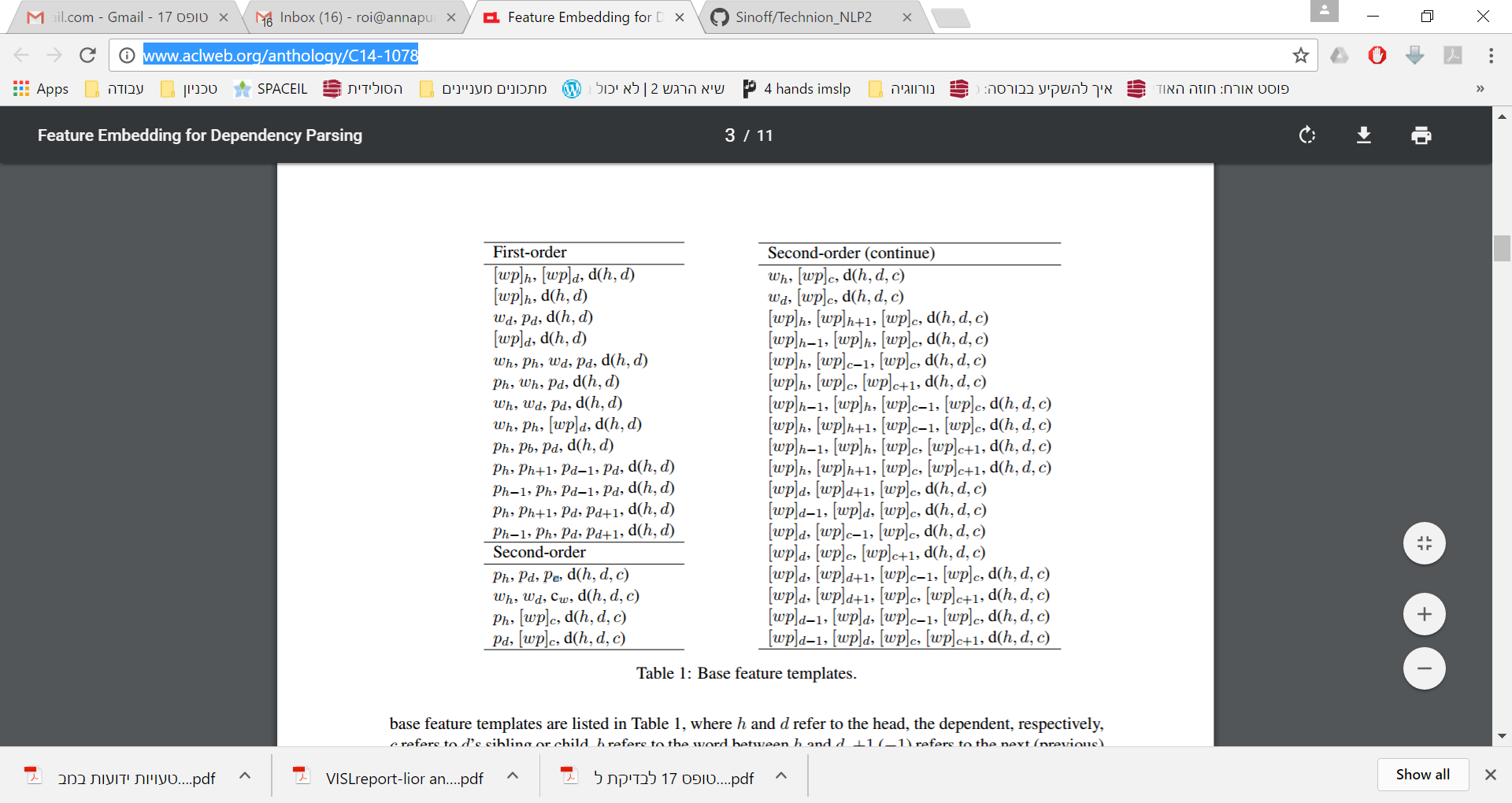
|  |  |
| --- | --- |
| **Feature Number** | **Meaning** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

להלן כמויות הפיצ'רים האלה:

|  |
| --- |
|  |

להלן הסברים לשינויים שביצענו:

1. ראשית, נציין שבחרנו את הפיצ'רים על סמך אינטואיציה ועל סמך מה שלמדנו בכיתה, אך בהמשך חיפשנו גם בספרות ומצאנו לדוגמה את הטבלה הבאה[[1]](#footnote-1). לשמחתינו הפיצ'רים שבחרנו תאמו במרביתם לאלו שהוצעו במאמר, אך הוא גם נתן לנו מוטיבציה לוותר על פיצ'רים מסויימים.



1. באופן כללי החלטנו לוותר על פיצ'רים אשר מופיעים גם עם dist (המרחק בין האב לבן) וגם ללא – השארנו רק את אלה הכוללים את המרחק. זה נבע הן מאחר שלתחושתינו למרחק יש השפעה גדולה על הקשר בין המילים (כפי שלמדנו בכיתה), הן מאחר שהמאמר הנ"ל פעל בצורה דומה, אך מעל הכל מאחר שהתוצאות שקיבלנו היו מיטביות בצורה זו.
2. כפי שצויין, ניסינו מספר רב של פיצ'רים אשר בסוף לא השתמשנו בהם (ואת חלקם אף מחקנו – כגון פיצ'רים המתייחסים לכמות האחים, אשר פגעו בביצועים). לדוגמה, החלטנו לוותר על פיצ'רים המתייחסים לאורך המשפט בסופו של דבר מאחר שהם לא תרמו לביצועים. עם זאת, ייתכן שאם היינו עושים חלוקה כלשהי של הערכים למחלקות יותר מוצלחות (ע"י הגדרת ספי אורך משפט – כגון "קטן מ-10", "בין 10 ל-15" וכו') פרמטר זה היה נותן לנו בכל זאת מידע אינדיקטיבי.

**הסקה**

1. במסגרת ההסקה ניסינו שתי אופציות למציאת ה- MST. הסיבה לכך היא שקוד ה- Chu-Liu-Edmonds כלל באגים ולכן לא תמיד יצר עצים (לעיתים היו מעגלים, לעיתים הגרף שהתקבל חזרה לא היה קשיר וכו'), וניסינו לפתור זאת באמצעות השיטה השנייה. עם זאת, בסופו של דבר התוצאות שהתקבלו עבור השיטה הראשונה היו טובות יותר, על אף המקרים בהם הגרפים שהתקבלו לא היו עצים תקינים. להלן שתי השיטות:
   1. Onetree – במסגרת שיטה זו, חיברנו את צומת ה"root" לכל הצמתים (כמובן רק בכיוון אחד), ופונקציית ה-MST היא שבחרה את הקשת המיטבית.
   2. Best Tree – במסגרת שיטה זו, חיברנו את צומת ה"root" בכל פעם רק לצומת אחת והרצנו את אלגוריתם ה-MST על עץ זה. בסוף בחרנו את העץ המיטבי מבין הריצות השונות.
2. נציין שהשיטה הראשונה מהירה יותר משמעותית – בערך פי 25, דבר המתאים לגודל המשפטים הממוצע (ובהתאם לכמות הקריאות ל- MST בשיטה 2).

**מבחן**

מודל בסיסי

להלן זמני הריצה והתוצאות עבור המודל הבסיסי:

|  |  |  |  |
| --- | --- | --- | --- |
| מס' איטרציות | זמן הריצה | אחוז הדיוק | המחשב עליו בוצעה הריצה |
| 20 | 0:07 | 29.8% | 4 i7 CPUs, 16 GRAM |
| 50 | 0:08 | 30.2% |
| 80 | 0:08 | 26.9% |
| 100 | 0:08 | 30.85% |

מודל מתקדם

נציין שהמימוש שלנו יכול להיות מהיר יותר אם נמחק פיצ'רים שאינם בשימוש, אך מאחר שממילא זמני הריצה סבירים, לא טרחנו לנסות ולשפר זאת, למקרה שנרצה בסופו של דבר להשתמש בפיצ'רים אלה.

להלן זמני הריצה והתוצאות עבור המודל המתקדם (כפי שתואר בחלק ה"לימוד"):

|  |  |  |  |
| --- | --- | --- | --- |
| מס' איטרציות | זמן הריצה | אחוז הדיוק | המחשב עליו בוצעה הריצה |
| 20 |  |  | 4 i7 CPUs, 16 GRAM |
| 50 |  |  |
| 80 |  |  |
| 100 |  |  |

כללי

1. כפי שניתן לראות, התוצאות שהתקבלו במודל המתקדם טובות יותר משמעותית מהתוצאות שהתקבלו במודל הבסיסי. הדבר סביר מאחר שהמודל הבסיסי לא כלל מידע מהותי לגבי הקשתות (לדוגמה אורכן), ושהמודל המתקדם נבחר בתור האופציה המיטבית מתוך סט הרצות ואופציות מאפיינים רחב יותר בצורה משמעותית.

**תחרות**

1. תחזית של אחוזי הדיוק שאנו צופים לקבל – דומה לאלו שקיבלנו עבור המבחנים שביצענו לעיל, או מעט נמוכה יותר. עם זאת יכולות להיות מספר סיבות יכולות להשפיע על ההבדל בין בחינת הקבצים השונים:
   1. התאמות מסוימות שביצענו בעקבות הטעויות שראינו בעת הבחינה בקובץ המקורי עשויות להתאים ספציפית לקובץ זה, ולא להיות כלליות מספיק כדי לשפר תוצאות גם על קובץ התחרות (אולי אף overfitting, על אף שניסינו להימנע מכך). נציין שלהערכתינו הסיכוי שהדבר קרה לנו במקרה הנוכחי נמוכה יחסית לתרגיל הקודם (שגם בו זה לא קרה...), מאחר שהפעם "האינטואיציה הבלשנית" שלנו פחות מפותחת, שהרי "Dependency trees" הם פחות אינטואיטיביים לעומת POS והקשרים ביניהם (כפי שהיה בתרגול הקודם).
   2. באופן דומה למה שכתבנו בתרגיל הקודם – יכול להיות קיים הבדל ב-domain ממנו הגיעו ה-test לעומת ה-comp (על אף שבמבט חטוף על הקבצים שניתנו לנו במקרה זה, נראה שזה לא המקרה).
2. על מנת לוודא שההטייה שנובעת מביצוע שיפורים מקסימליים על קובץ ה-test תהייה נמוכה ביצענו מעין "Cross Validation" בסיסי מאוד – למדנו על ה-test והסקנו על ה-train, על מנת לראות שהתוצאות שאנו מקבלים עדיין טובות.  
   להוסיף תוצאות?

**סיכום**

נציין שבדומה לכלל הקורס, גם תרגיל זה היה מעניין ומעורר מחשבה. על אף שבתרגיל הקודם אחת המסקנות היתה שכוח חישובי יותר גדול היה יכול לעזור לנו מאוד, החלטנו לוותר על שימוש בשרת בענן, מאחר שהכוח החישובי הנדרש הפעם היה נמוך יותר משמעותית (הפרספטרון מהיר יותר משמעותית ביחס ל Gradient Descent אותו מימשנו בתרגיל הקודם).

מסקנה נוספת מהתרגיל הקודם היא שעל אף שיש יתרונות בקידוד בזוג, במסגרת הזמן של התרגיל זו שיטה בעייתית. לכן אמנם הגדרת מבני הנתונים (ובהתאם המחשבה על מימוש האלגוריתמים) נעשתה במשותף, אך הקידוד נעשה בנפרד (גם אם בעת ישיבה משותפת זה לצד זה). נציין כי למרות הנ"ל, דגלנו גם הפעם באחריות משותפת על הקוד וביצענו שיפורים ותיקונים בכל הקוד – מבלי להתייחס למי כתב את הקוד במקור.

בחלוקה גסה ניתן לומר שאור כתב את classes, depparser והבסיס של features, תיעד, וכן חקר לעומק, דיבג וניסה למצוא קוד נכון יותר של edmonds. רואי (שכתב פייתון לראשונה במסגרת התרגילים בקורס זה) כתב את main, learning, inference ומרבית הפיצ'רים המתקדמים (אשר האחריות לחשיבה עליהם הופקדה עליו, אך נעתשה כמובן בהתייעצות ובסיכום עם אור) וכן את הבסיס של דו"ח זה.

1. From: Chen, Wenliang, Yue Zhang, and Min Zhang. "Feature Embedding for Dependency Parsing." COLING. 2014. [↑](#footnote-ref-1)