דוח פרויקט

אור ישראל - 204684724

איתמר איילון - 206024796

ניסוי ראשון:

התחלנו להבין איך הדאטה עובד כדי לחשוב איך לבנות לו מודל. ראינו שהוא מורכב משדות של לייבלים, משפטים, ו-parse tree בכל שורה.

ניסינו להשתמש בחוקים של parse tree על מנת להסיק האם משפט אחד יכול לגרור את השני. עברנו על המצגות וחקרנו באינטרנט, והמימוש הרגיש מסובך להתחיל ממנו אז החלטנו ללכת לגישה יותר פשוטה:

השתמשנו רק בשדות שני המשפטים ובלייבלים שלהם. חשבנו שמשפט אחד יכול לנבוע מהשני אם הוא דומה לו. וע"פ מה שלמדנו בהרצאה ניתן לקבוע האם שני משפטים דומים ע"י המרחק בין שני הוקטורים המייצגים את המשפטים.

יצרנו מטריצה שכל שורה במטריצה היא הייצוג הוקטורי של משפט וההיפותזה שלו. לצורך כך השתמשנו ב-tf-idf vectorizer, והשתמשנו במסווג לינארי כדי לנסות לסווג את המשפטים ללייבלים שלהם.

אימנו את המודל עם הדאטה של devs וב-test קיבלנו דיוק של 57%. כשרצינו להריץ אותו על כל ה-train, קיבלנו שאין מספיק זיכרון.

חיפשנו באינטרנט ומצאנו שיטות שונות לעבודה עם sparse matrix וכך הצלחנו לחסוך בהרבה זיכרון.

ניסינו להשתמש בוקטורייזר אחד על המשפטים ובוקטורייזר אחר על ההיפותזות ולסווג בעזרת שניהן.

שמנו לב שכאשר השתמשנו באותו vocabulary (מהוקטורייזר של המשפטים) לשני הוקטורייזרים, הדיוק ירד למרות שההיגיון שלנו אמר שהוקטורייזר השני חייב לעבוד על אותו אוצר מילים של הראשון, כי אחרת ההשוואות אינן משמעותיות. ההנחה שלנו היא שהתוצאה הייתה מקרית.

לבסוף לאחר מחשבה והתבוננות במקורות הבנו שהשימוש ב- tf-idf vectorizer מתייחס לפרמטרים של מילים בודדות ולא לוקח בחשבון את ההקשרים בין המילים.

(אם כי גם שימוש ב-ngrams כלשהו לא הנבנו תוצאות יפות)

לאחר מכן ניסינו להשתמש ב- edit distance (מספריית nltk) על מנת להסיק את הדמיון בין המשפטים להיפותזות.

ניסוי זה הניב לנו תוצאות נמוכות יותר וגם הבנו שעלולה להיות בעיה כאשר במשפט ובהיפותזה יהיו מילים רחוקות אחת מהשנייה אך קרובות במשמעותן למשל:

"Sofa" ו- "Couch". המודל יגיד שהמילים לא דומות אבל קל לראות שהן מילים נרדפות.

ויותר מזה, משפטים יכולים להכיל הרבה יותר מילים מאשר ההיפותזות שלהן (או להיפך). וכתוצאה מכך המרחק יהיה מאוד גדול למרות שהמשפטים דומים סמנטית.

התוצאות שקיבלנו עבור כל השלבים הללו:

שיטת חישוב	*cosine-similarity*	*edit-distance*
2-way	0.687	0.664
3-way	0.442	0.353

* זמן האימון והטסט לוקח קצת מעל 3 דקות.

ניסינו להבין מדוע ההפרשים כאלה גדולים וכשהדפסנו את ה-confusion matrix ראינו שהבעייתיות נובעת מ-neutral.

<u>ניסוי שני:</u>

לאחר הרבה מאוד טעויות וההבנה מדוע הן נגרמות, ומהעובדה שמספר המילים במשפט לא מספר שום דבר על סמנטיקה, החלטנו לבנות מודל deep-learning שיתבסס על משהו אחר מאשר מספור מילים ולהשתמש ב-word2vec.

ניסינו לחשוב איך לגשת לבעיה של השוואת 2 משפטים ובדיקה האם המשפט הראשון הוא הסקה של השני. ופרקנו את הבעיה לתת בעיה בה ניסינו להבין איך בכלל לחשב את הסמנטיקה בין כל 2 מילים לפני שנחשב בין כל 2 משפטים.

לשם כך למדנו איך לעבוד עם word2vec – בכדי לבחון סמנטיקה בין 2 משפטים, אפשר לבדוק האם יש קשר סמנטי כלשהו בין המילים של המשפטים השונים.

כאן התחלנו בכך שהסרנו את המשפטים וההיפותזות אשר אין להם label כדי למנוע בלבול נוסף מהמודל.

לאחר שאימנו את המודל word2vec על הדאטה חשבנו על דרך להמיר משפט לוקטור שמיוצג בצורה הטובה ביותר ע"י השימוש ב word2vec.

ניסינו ללכת בהיגיון שמשפט מורכב ממילים וכל מילה שמוסיפים למשפט פשוט מגדילה את אורך המשפט (אורך הוקטור) ועלולה לשנות את הסמנטיקה (הזווית ביחס לצירים). כלומר ננסה לסכום את כל הוקטורים המייצגים מילים במשפט וכך נוכל לייצג את המשפט ע"י וקטור (ובדקנו האם גם הממוצע של המילים יכול לעזור).

לבסוף נצמיד את שני הוקטורים אחד ליד השני ונסווג בעזרת LR.

וקיבלנו את התוצאות הבאות:

שיטת חישוב	*sum*	*mean*
2-way	0.713	0.738
3-way	0.563	0.588

* זמן האימון והטסט לוקח באיזור ה-80 שניות.

בשביל לשפר את הביצועים הסרנו כפילויות לאימון ה-word2vec , וכל המילים באותיות קטנות.

<u>ניסוי שלישי:</u>

רצינו לחשוב על דרך להשתמש ב-tokens איכותיים יותר, וחשבנו על מודל מתקדם יותר של SpaCy, שיוכל לעזור לנו לנתח משפטים בצורה איכותית יותר.

התחלנו לעבוד איתו וראינו שלוקח לו הרבה מאוד זמן להתאמן.

השקענו הרבה מאוד זמן בלחפש דרכים להגדיר אותו בצורה שיעבוד מהר יותר, אבל עדיין לא מהר מספיק. בנוסף לזמן האיטי הוא גם דרש הרבה זיכרון בדיסק.

בסופו של דבר הגענו למסקנה שאנחנו נשתמש בו בשביל לקבל טוקנים טובים ואיכותיים שיהיה ניתן להשתמש בהם, אז אותם כבר בנינו לבדנו בעזרת ספריית nltk.

השתמשנו ב lemmatizer, סיננו stop words, punctuations, numbers, וכך נשארנו עם משפטים שאכן ניתן להוציא מהן קונטקס כלשהו מבלי הרבה רעשים של מילים שיכולות לפגוע במודל.

כדי לשדרג עוד יותר את המודל השתמשנו ב doc2vec, שהוא עצמו משתמש ב-word2vec בכדי לקבל וקטורים למשפטים.

ניסינו להבין איך לעבוד עם המודל, איפה נמצאים הוקטורים אחרי שמאמנים את המודל, מה בכלל עושים איתם, איך הוא מסווג את המשפטים ומתי.

בהתחלה שלחנו למודל קלט של רשימה בה כל איבר הוא TaggedDocument כאשר כל איבר כזה מכיל משפט (premise) וה-tag שלו הוא מספר המייצג מספר בין 1 לכמות המשפטים. וכן אותו דבר לגבי ההיפוטזה.

ניסינו לאמן שני מודלים נפרדים של doc2vec חשבנו שאת הוקטורים שמייצגים את המשפטים וההיפותזות שלו ניתן להוציא דרך הפרמטר dv.vectors, ואיתם ביחד עם הלייבלים ניתן לאמן את המסווג LogisticRegression.

במודל הזה הגענו לדיוקים של כ- 50 אחוז.

כדי לאמן, ביצענו tokenize על המילים כמו שרשמנו קודם ויצרנו רשימה של TaggedDocument כקלט למודל, אך שהפעם חיברנו את המשפט (premise) וההיפותזה שלו למשפט אחד אשר הוא הטקסט, והלייבל הוא ה-tag.

בכך קיבלנו בסוף האימון מודל doc2vec עם 3 וקטורים אשר כל אחד מייצג את ה-tag המתאים לו.

מכאן השתמשנו במטודה infer_vector של המודל, שמקבלת משפט ומחזירה וקטור הסק שהכי מתאים למשפט ממה שאימנו את המודל. ביצענו זאת לכל המשפטים (עם ההיפותזות שלהם) בדאטה, ואת המטריצה המתקבלת שלחנו למסווג (LR).

וקיבלנו את התוצאות הבאות:

עבור 2-way: דיוק של 0.673

עבור 3-way: דיוק של 0.484

* זמן האימון והטסט לקחו יותר מחצי שעה.

<u>ניסוי רביעי – אחרון:</u>

רצינו לשנות את כל המודל, לא הסתפקנו רק ב sentence embedding עם מסווג לינארי פשוט, והחלטנו להשתמש ב-TensorFlow בשביל לבנות את הרשת.

בחיפושים נרחבים באינטרנט הבנו ש-keras היא ספרייה מאוד נוחה לבניית רשתות נוירונים.

בהתחלה ניסינו להשתמש בשכבות LSTM ושכבת Embedding המסופקים על ידי הספרייה.

ניסינו לשלוח POS של משפטים והיפוטזות שאותם חילצנו באמצעות Spacy.
שירשרנו אותם עם תו הפרדה בינהם ועשינו padding על מנת להשלים את המשפט לאורך המשפט הכי ארוך בdata-set.

לאחר מכן, השתמשנו בטוקנייזר ושלחנו את זה לרשת שלנו.

התוצאות היו לא מספקות ובמהלך ארבעת הepochs הראשונים ראינו שזה עומד על כ-40 אחוז וויתרנו.

החלטנו להשתמש במודל שיצא כבר עם ציונים גבוהים ושנות אותו כך שהסיווג יתבצע בעזרת רשת FCNN פשוטה.

התחלנו משימוש ב word2vec כמו בניסוי 2, קיבלנו את ה-embedding, ובמקום לשלוח את הוקטורים המתקבלים למסווג לינארי, שלחנו לרשת FCNN.

הארכיטקטורה: $input \rightarrow [Linear \rightarrow Relu]\ x\ 2 \rightarrow softmax/sigmoid$

אופטימייזר: $adam$

פונק' הפסד: $cross\ entropy$

שכבה חבויה ראשונה בעלת 64 נויירונים, השנייה בעלת 128.

לא השתמשו במודל עמוק כדי לא לגרוע את זמן אימון ולא מודל פשוט מידי שלא ילמד כלום.

כבר עבור המודל הפשוט הזה ללא שינויים בהיפר-פרמטרים התוצאות היו מפתיעות:

2-way: 0.816 3-way: 0.702

מכאן ניסינו לשנות את המודל, רצינו לנסות לבנות מודל שמקבל 2 קלטים שונים, אחד יהיה למשפטים והשני להיפותזות, והוא יחזיר כפלט את אחד הלייבלים. המודל הניב תוצאות זהות למודל הפשוט.

גם ניסינו כמו בהתחלה לבנות מודל RNN עם שכבות LSTM, האימון היה ארוך בהרבה והתוצאות היו מעט יותר נמוכות מהמודל הפשוט.

כמו כן הוספנו batch normalization בין השכבות, זה בהחלט הקשה על המודל להתאמן והוא נתן דיוק גבוהה יותר, אפס כי בזמן מבחן התוצאות היו זהות. אפקט שלא היה לנו כל כך ברור מדוע, כיוון שדאגנו שהמודל לא יבצע over-fitting אבל עדיין הפרדיקציות על דאטה שלא ראה מעולם לא משתפרות למרות שהמודל כן.

גם את התוצאות האלה היינו יכולים לשפר בוודאות – לשנות את ה-embedding, לאמוד ארכיטקטורות שונות, לבצע אופטימיזציה להיפר-פרמטרים, אך הרגשנו מסופקים מהתוצאות האלה אחרי כמות השעות האדירה שהשקענו. והבדיקות הללו יכולות לקחת הרבה מאוד זמן, בכל פעם לאמן מחדש את המודל בצורה שונה.

<u>הוראות הפעלה:</u>

יש להתקין את ה-requirements.

בקובץ main.py קיים קבוע ARGS, מילון בו הנתיבים של snli datasets.

וקיים קבוע CONFIG בו ניתן לקבוע על איזה dataset מעוניינים להריץ את המודלים.

פונק' test_model מריצה מודל עבור 2\3-way. כל מודל מבצע train לאימון, ואחרי test שומר את ה-acc, confusion mat באובייקט.

בנוסף קיימים 6 קבצים נוספים (experiment_[1-4], baseSNLIModel, timing).
כולם צריכים להיות באותה תיקייה של ה main כדי שהקוד ירוץ.

הסבר קצר על הקבצים:

baseSNLIModel – נותנת אבסטרקציה למודל שמסתמך על דאטה סט של snli.

experiment_[1-4] – כל הניסויים שביצענו. בתוכם גם יש הערות לעוד קוד שניסינו במקביל באותו ניסוי (למשל פרמטרים שונים או גישות שונות).

timing – ניתן להוסיף אנוטציה כדי למדוד זמנים של פונק'.