שימוש במודל ה-NLP - 'BERT' לצורך וקטוריזציה של מאמרים וסיווגם לקטגוריות במספר היררכיות

פרויקט בבינה מלאכותית
הפקולטה למדעי המחשב, הטכניון

חברי הצוות בפרויקט

אדם בוטח, 312247125
botach@campus.technion.ac.il

אביאל שמחי, 305376063
aviel@campus.technion.ac.il

מבוא

באוקטובר 2018 הציגו במאמרם [1] חוקרים בצוות ה-AI Language של חברת Google מודל ייצוג שפה חדש המכונה 'BERT'.

הכלי (בגרסתו הבסיסית) הינו רשת נוירונים בעלת 12 שכבות שבהינתן משפט קלט אחד (או שניים) פולטת לכל מילה במשפט וקטור בן 768 מימדים המייצג את המשמעות הסמנטית של המילה.

אולם, החידוש העיקרי של המודל החדש ביחס למודלי word embedding קודמים בא לידי ביטוי ביכולתו של המודל לזהות את <u>ההקשר</u> שבו מילים מופיעות בתוך משפטים באמצעות המילים הבאות לפניהן או אחריהן באותו המשפט. תוספת ההקשר לסמנטיקה מאפשרת למודל לפרש נכון מילים בעלות מספר משמעויות שונות (כגון המילה 'bank' באנגלית) בהתאם לאופן השימוש שבהן. כך, המודל מסוגל לפלוט מספר וקטורים שונים עבור אותה מילה בהתאם לפירושיה השונים, בשונה ממודלי word embedding קודמים (כגון Word2Vec) המספקים לכל מילה וקטור אחד בלבד.

מאז יציאתו של המודל בשנה שעברה BERT החל לשמש באפליקציות רבות בתחום ה-NLP וברבות מהן השיג תוצאות מרשימות מאוד שטרם נראו בעבר.

אולם, למיטב ידיעתנו טרם נעשה מחקר מקיף שבוחן כיצד ניתן לנצל את יכולותיו של המודל לצורך <u>קלסיפיקציה של מסמכים שלמים</u>. לפיכך, בפרויקט זה החלטנו לבחון את הסוגיה הזו בעצמנו. בנוסף, לצורך הפרויקט החלטנו שלא להתעסק בבעיית סיווג המסמכים הקלאסית אלא לקחת אותה <u>צעד אחד קדימה</u>:

בהינתן מאגר מסווג <u>היררכית</u> של מאמרים בקטגוריות שונות, נרצה לחקור <u>כיצד ניתן לחזות את מסלול הסיווג המתאים בהיררכיית הקטגוריות עבור מאמר חדש</u>. אנו נבצע זאת על מאגר (חלקי) של מאמרים מתוך 'ויקיפדיה' האנגלית.

לדוגמא, מסלול אפשרי בהיררכיית הקטגוריות בוויקיפדיה עבור המאמר 'Election Day' הוא:

Wikipedia Root -> Politics -> Voting -> Elections

כאשר המאמר הנ"ל נמצא תחת הקטגוריה האחרונה במסלול: 'Elections'.

אם כך, בהינתן סט הווקטורים למילים המתקבל ע"י BERT בהרצתו על מאגר המאמרים נרצה להכליל את רעיון הוקטוריזציה גם עבור מאמרים, וכך לסווג בהצלחה מאמר חדש במספר רמות.

לפרויקט שלפניכם ישנן אם כך שתי מטרות:

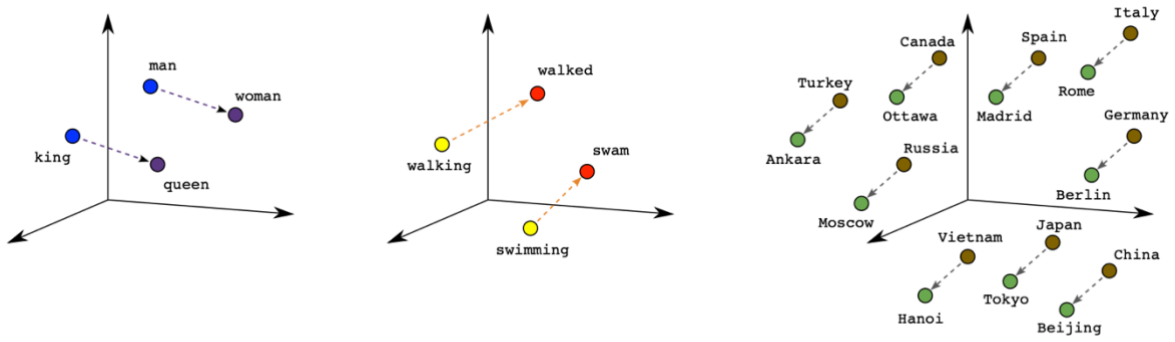
1. השוואה בין שיטות וקטוריזציה שונות לביצוע document embedding באמצעות פלט הכלי BERT.
2. בניית מסווג היררכי שבהינתן מאמר חדש יציע את היררכיית הקטגוריות שתחתיה יש לסווג את המאמר בתוך מאגר מאמרים המבוסס (חלקית) על ויקיפדיה האנגלית.

בדו"ח שלפניכם נתאר בהרחבה את תהליך החשיבה שעברנו בכדי לחקור ולהשיג את המטרות הנ"ל.

תוכן עניינים

Word Embedding – רקע

כפי שציינו במבוא, הפרויקט שלנו (בבסיסו) סובב סביב בעיית למידה קלאסית למדי – סיווג מסמכים.
בכדי לפתור את הבעיה, ולאפשר למחשב "להבחין" בין מסמכים מקטגוריות שונות, צריך בראש ובראשונה למצוא דרך לייצג את המסמכים שמצד אחד תהיה בפורמט שאיתו המחשב מסוגל לעבוד (מספרים!), ומצד שני, ששני מסמכים יהיו "קשורים" זה לזה בפורמט החדש אם ורק אם הם היו קשורים זה לזה גם בפורמט הישן (טקסט).
פתרון ברור לבעיה מגיע מכיוון האלגברה הליניארית – מרחבים וקטורים.
בגישה זו, שנקראת word embedding, כל מילה בשפה טבעית כלשהי (לדוגמא – אנגלית) תיוצג ע"י וקטור במרחב n מימדי כאשר המטרה היא שווקטורים של כל שתי מילים יהיו "דומים" זה לזה במרחב אם ורק אם אותן מילים קשורות זו לזו במציאות.



Male-Female Verb Tense Country-Capital

חשוב להבחין שטרם הגדרנו מהו "דמיון" בין וקטורים.
באופן אינטואיטיבי ניתן להגדיר ששני וקטורים דומים זה לזה אם המרחק האוקלידי בין הנקודות שאותם הם מגדירים במרחב ה-n מימדי, אם מניחים שהם יוצאים מהראשית, קרובות זו לזו. לחילופין, מקובל יותר במודלים רבים ב-NLP עבור word embedding להגדיר ששני וקטורים יהיו קרובים זה לזה ככל שהזווית ביניהם קטנה יותר (מושג המכונה Cosine Similarity, ועליו נפרט עוד בהמשך).

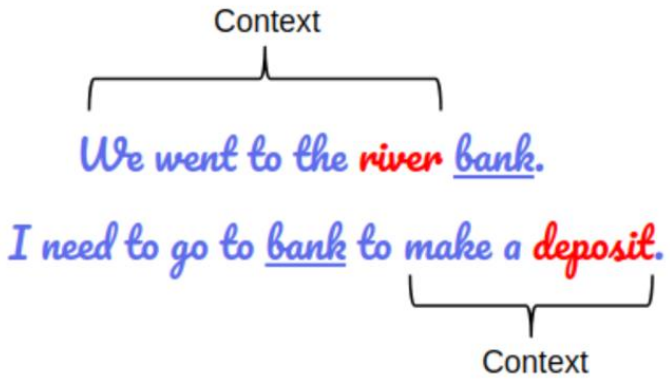
לאחר שבידינו יהיו וקטורי המילים, המטרה שהצבנו בפנינו בביצוע הפרויקט הייתה להכליל את רעיון ה-word embedding גם עבור מסמכים. כלומר שאלנו את עצמנו את השאלה:
כיצד ניתן להשתמש בוקטורים של מילים המרכיבות מאמר מסוים בכדי לייצר וקטור יחיד שייצג את המאמר כולו?

בשנים האחרונות פותחו בקהילת ה-NLP מגוון רחב של כלים שונים עבור word embedding. אחד המפורסמים שבהם הינו הכלי word2vec שפותח ע"י צוות המחקר של Google ופורסם ב-2013. הכלי, שבבסיסו הינו רשת נוירונים בעלת שתי שכבות, נועד להיות מאומן על קובץ טקסט גדול (לדוגמא ויקיפדיה האנגלית בשלמותה). בתום תהליך האימון המודל מכיל וקטור עבור כל מילה שאותה "ראה" בקובץ הטקסט

שעליו אומן. המטרה היא, כפי שכבר ציינו, ששתי מילים הקשורות זו לזו סמנטית יהיו קרובות זו לזו גם במרחב הוקטורי המתקבל.

אף על פי ש-word2vec היה כלי מהפכני בתקופה בה יצא, אחת המגרעות הגדולות שלו, ושל כלי word embedding נוספים הייתה חוסר היכולת שלהם "לתפוס" את ההקשר (context) של המילים שאותן הן ראו, ולבטא את אותו ההקשר בוקטורי הפלט שלהם.

נביט למשל בדוגמא הבאה : [2]



ברור כי למילה 'bank' כפי שהיא מופיעה בשני המשפטים הנ"ל יש משמעות סמנטית שונה לחלוטין : במשפט הראשון הכוונה היא כמובן לגדת נהר, בעוד שבמשפט השני מתכוונים לבנק שבו מפקידים כסף.

הבעייתיות במודלים הישנים הינה שבהרצתם על שני המשפטים הנ"ל הם מתעלמים לחלוטין מההקשר שבו הופיעה המילה 'bank' ופולטים עבורה את אותו הוקטור בדיוק.

בכדי לפתור את הבעיה הנ"ל הגיעו חוקרי ה-NLP למסקנה שכלים חדשים יותר צריכים להיות מסוגלים לתפוס גם את ההקשרים של המילים שאותם הם רואים, ולהיות מסוגלים לתת לאותה המילה וקטורים שונים בהתאם להקשרים השונים שבהם היא מופיעה. בנוסף, כיוון שההקשר של מילה מסוימת יכול להופיע לפניה <u>או</u> אחריה במשפט (לדוגמא המילה 'river' במשפט הראשון, והמילה 'deposit' במשפט השני), הכלים החדשים יהיו חייבים להיות מסוגלים לסרוק את המשפטים שהם מקבלים בשני כיוונים.

בשל יציאתו לאחרונה של כלי חדש שעונה על הדרישה הנ"ל, החלטנו, בהמלצת המנחה גיא, לזנוח את הרעיון להשתמש ב-word2vec עבור צרכי ה-word embeddings של הפרויקט (הרעיון אותו תיארנו בהצעת הפרויקט), ולהשתמש בכלי החדש.

כלי זה נקרא BERT.

רקע – הכלי BERT

BERT (או בשמו המלא: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers) הינו מודל word embedding שפותח ע"י חוקרים בצוות ה-AI-Language של חברת Google ויצא לראשונה בסוף שנת 2018.

הכלי (בגרסתו הבסיסית), הינו רשת נוירונים בעלת 12 שכבות שבהינתן משפט קלט אחד (או שניים) פולטת לכל מילה במשפט וקטור[1] בן 768 מימדים המייצג את המשמעות הסמנטית של המילה כתלות בהקשר שלה בתוך המשפט.

כדי להמחיש את הרעיון הנ"ל נביט בדוגמת הרצה של המודל על המשפט הבא: [3]

$$\text{“}After\ stealing\ money\ from\ the\ bank_1\ vault,$$
$$the\ bank_2\ robber\ was\ seen\ fishing\ on\ the\ Mississippi\ river\ bank_3.\text{”}$$

במשפט הנ"ל, ברור כי שני המופעים הראשונים של המילה 'bank' מתייחסים לבנק שבו מפקידים כסף, ואילו המופע השלישי מתייחס לגדת נהר.

הטבלה הבאה מתארת השוואה בין וקטורי הפלט שהתקבלו עבור המופעים השונים של המילה 'bank' בהרצת המודל על המשפט. ההשוואה התבצעה באמצעות פונקציית Cosine Similarity, שמחזירה ערך בין 0 ל-1 כאשר 1 מסמל התאמה מושלמת ו-0 מסמל חוסר קשר מובהק:

מופע א'	מופע ב'	ציון Cosine Similarity
$bank_1$	$bank_3$	0.680
$bank_1$	$bank_2$	0.946

התוצאות הנ"ל מדגימות בצורה יפה מאוד כיצד BERT מסוגל לזהות הקשר בין מילים במשפט, גם אם מדובר במילה אחת בעלת משמעויות שונות. כאן חשוב שוב גם לציין שבמודלי word embedding קודמים דוגמת word2vec עבור כל שלושת המופעים של המילה 'bank' היה מתקבל וקטור אחד בלבד, שהיה נטול הקשר, ושהיה צריך "להיקרע" בשל הניסיון לייצג מספר משמעויות סמנטיות שונות בבת אחת.

[1] בפועל הפלט של המודל מעט מסובך יותר: המודל מחזיר מערך רב-מימדי המאפשר גישה לפלט של כל אחת מ-12 השכבות הנסתרות, כאשר עבור כל שכבה ניתן לגשת לוקטור שהתקבל בפלט של אותה השכבה עבור כל מילה במשפט (המיוצגת ע"י token). במסגרת הפרויקט הוקטור שאנחנו בחרנו לקחת עבור כל מילה הינו הוקטור שהתקבל בפלט השכבה ה-11 של המודל, כיוון שנטען שהשכבה האחרונה ברשת קרובה מדי לפונקציות המטרה של האפליקציות שעליהן המודל אומן בשלב ה-pre-training (שעליהן נפרט בהמשך), ולכן הוקטורים של השכבה האחרונה עלולים להיות מוטים לטובת האפליקציות הללו.

המודל של BERT עבר ע״י מפתחיו אימון מקדים (pre-training) על מקבץ טקסט עצום שכלל את כל תוכן ויקיפדיה האנגלית (מעל 2.5 טריליון מילים) וגם חלק ממאגר הספרים של Google (מעל 0.8 טריליון מילים).
האימון המקדים כלל ביצוע ממושך של שתי משימות NLP מרכזיות:
Masked Language Modeling: אימון המודל על אוסף משפטים כאשר בכל משפט מילה אחת נבחרת באקראי ומכוסה ע״י אסימון כיסוי [MASK]. לאחר מכן, על המודל לחזות מתוך ההקשר בו נמצא אסימון הכיסוי (המילים המופיעות לפניו או אחריו במשפט) מהי המילה שכוסתה. העובדה שהמודל של BERT מסוגל ללמוד את הקשר הנ״ל מתוך מילים המופיעות לפני או אחרי האסימון הינה דוגמא לדו-כיווניות של המודל (ועל כן המילה bidirectional מופיעה בשמו).
Next Sentence Prediction: אימון המודל על אוסף של זוגות משפטים כאשר חלק מהזוגות הינם של משפטים עוקבים וחלק של שני משפטים שנבחרו באקראי. מטרת המודל הינה ללמוד לחזות אילו מבין זוגות המשפטים הינם משפטים עוקבים.

השילוב של שתי השיטות הנ״ל במהלך האימון המקדים העניק ל-BERT את היכולת להבחין בהקשרים של מילים שונות בתוך משפט ובהקשרים בין משפטים חופפים. היכולת הזו של BERT לזהות הקשר, בנוסף למשמעויות סמנטיות, היא זו המפרידה בין מודל זה לבין רבים ממודלי ה-word embedding שפותחו עד היום.

לפיכך, חשבנו שיהיה מסקרן לחקור במסגרת פרויקט זה כיצד ניתן להכליל את יכולות ה-word embedding של BERT עבור מילים בהקשרים של משפטים בודדים בכדי לבצע וקטוריזציה למאמרים שלמים.

תיאור כללי של מהלך העבודה

כמתואר בתרשים הבא, הבעיה שלנו מצריכה כוונון פרמטרים לא רק בתהליך הלמידה אלא גם בתהליך איסוף הנתונים והוקטוריזציה. בכל אחד מהשלבים האלו כוונון הפרמטרים הוא חשוב מאוד ומשפיע על פתרון הבעיה בצורה ניכרת.

תרשים זרימה

פרמטר לכוונון : אלגוריתם האיסוף וכמות המאמרים
פלט : מאמרים בתצורת טקסט וטופולוגית קטגוריות

פרמטר לכוונון : אופן ביצוע הוקטוריזציה
פלט : מאמרים בתצורת וקטורים וטופולוגית קטגוריות

פרמטר לכוונון : הפרמטרים של אלגוריתמי הלמידה
פלט : סיווג וקטורי המאמרים לקטגוריות

השוואת התוצאות שהשיגו אלגוריתמי הלמידה השונים באמצעות שיטות הוקטוריזציה השונות והסקת מסקנות

שימוש במסקנות מהשלבים הקודמים לבניית מסווג היררכי יחיד שבהינתן מאמר חדש יחזה את המסלול המתאים עבורו בהיררכיית הקטגוריות

הדוח כפרקים

הפרקים הבאים יעסקו בכל אחד מהתחנות הנ״ל, כאשר נשתדל לפרט ככל הניתן על השיפורים שביצענו בכל שלב.

איסוף הנתונים

תכנון ובחירת אופן איסוף הנתונים

היררכיית הקטגוריות בויקיפדיה לא מהווה עץ (תיתכן תת קטגוריה הנמצאת תחת שתי קטגוריות אם, מה שעלול להוביל למעגלים). לכן, על מנת לקבל בסיס נתונים טוב נדרשנו לתכנן ולשנות שוב ושוב את אופן איסוף הנתונים.

את תהליך החשיבה שעברנו והבעיות איתן התמודדנו לאורך תהליך כריית הנתונים ובניית האלגוריתם עבורו נתאר בהרחבה בפרק זה.

רעיון האלגוריתם הבסיסי

מטרת העל שהגדרנו לאלגוריתם כריית הנתונים הייתה להחזיר מאגר מסודר ונקי מרעשים של מאמרים המסודרים לפי קטגוריות בצורה היררכית.

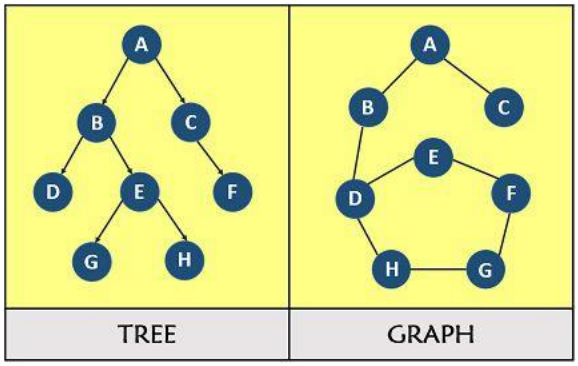
הרעיון הבסיסי היה לרוץ באופן רקורסיבי על ויקיפדיה החל מהשורש שלה (בשלב זה לא היינו בטוחים שאכן יש כזה) ולסרוק אותה בצורה הדרגתית בהתאם להיררכיית הקטגוריות שבה. במהלך הריצה תכננו לאסוף את המאמרים ולשמור אותם ואת הקטגוריות שתחתן הם נמצאים בצורה מסודרת.

נקודות וניסויים אותם בחנו

עץ? או אולי גרף רגיל?

- **תיאור הבעיה**

כיוון שהיררכיית הקטגוריות אינה עץ, שימוש באלגוריתם רקורסיבי כנ"ל יוכל לגרום לכפילויות במאמרים או בקטגוריות שיאספו על ידיו. כלומר, יתכנו מאמרים שישנם שניים (או יותר) מסלולים מהשורש אליהם בהיררכיית הקטגוריות בוויקיפדיה. בכדי להתמודד עם הבעיה הנ"ל שקלנו האם להשאיר את הקטגוריות כגרף רגיל (ובמצב זה המסווג יצטרך להחזיר יותר ממסלול אחד למאמר אם קיים), או למדל אותן כעץ (ע"י קטימת ענפים), וכך להימנע מבעיה זו.



- **מסקנות**

בפגישת ההתייעצות עם גיא הסכמנו כי מידול בעזרת גרף יוכל להוסיף קושי רב מאוד לפתרון הבעיה, שגם כך איננה פשוטה. לכן ניסינו תחילה למדל את הקטגוריות כעץ ונוכחנו לדעת שגם במידול זה ישנן בעיות נוספות. לכן החלטנו לבסוף לדבוק במודל זה עבור הפרויקט כולו.

חשוב גם לציין שבכדי לבצע את המידול כעץ היינו צריכים לזרוק מאמרים שכרינו אשר הופיעו ביותר מקטגוריה אחת. כך וידאנו שכל מאמר במאגר שלנו משתייך בדיוק לקטגוריה אחת, ובפרט לקטגוריה אחת בכל רמה בהיררכיה, מה שיוצר אילוץ של מסלול יחיד מהשורש אל כל מאמר.

הדרישות למימוש ולהרצת האלגוריתם

לצורך בניית אלגוריתם כריית הנתונים עמדנו בפנינו שתי שאלות מרכזיות:

מהו השורש של ויקיפדיה?

כל אלגוריתם לכריית נתונים מצריך שורש (אחד או יותר) שממנו אמור להתחיל תהליך הכרייה, וכך גם האלגוריתם שלנו. כאמור, ויקיפדיה ממודלת כגרף מכוון, ולא כעץ, אך כיוון שגרף זה ממודל בצורה היררכית קיווינו שגם לו קיים איזשהו שורש: דף מרכזי אחד שמרכז תחתיו לינקים לכל הקטגוריות המרכזיות של האנציקלופדיה האינטרנטית החופשית. לאחר שגילינו שדף זה איננו דף הבית של ויקיפדיה האנגלית, ושדף הבית איננו מכיל אף לינק לדף שכזה, מצאנו לאחר שיטוט באינטרנט את הדף המיוחל: הדף נקרא 'Main Topic Classifications', והוא משמש כמעין קטגוריית אם כללית ביותר שעומדת בראש ההיררכיה של ויקיפדיה ומכילה לינקים לכל הקטגוריות הגדולות ביותר של האנציקלופדיה.

כיצד ניתן לכרות מאמרים מויקיפדיה בצורה יעילה ונוחה?

כידוע, כריית נתונים (ובפרט מאמרים) ישירות מדפי HTML בצורה סדרתית הינה משימה לא פשוטה. לכן, חיפשנו ספריית Python פשוטה ונוחה למתכנת שתאפשר לנו לכרות את הנתונים ללא צורך להתעסק ישירות עם תוכן HTML. הספרייה Wikipedia-API ענתה על כל הצרכים שלנו. לספרייה ממשק פשוט שמקבל שם של מאמר או קטגוריה בויקיפדיה, מוריד את תוכן דף האינטרנט של אותו מאמר/קטגוריה, ויוצר עבורו אובייקט מסודר. כל אובייקט שכזה מכיל מידע שממנו אפשר להסיק האם מדובר בדף המייצג קטגוריה או מאמר: במידה ומדובר בקטגוריה, הוא מכיל רשימה של כל קטגוריות הבת של אותה הקטגוריה ושל כל המאמרים שנמצאים ישירות תחת אותה קטגוריה. ובמידה ומדובר במאמר, האובייקט מאפשר גישה נוחה לכל חלקי הטקסט של המאמר: summary, sections וכד'. חשוב גם לציין שהעובדה שיכולנו לגשת לחלקי המאמר השונים בקלות באמצעות הממשק של הספרייה התבררה כחשובה מאוד בהמשך עבור תהליך הוקטוריזציה, כיוון שעבור כל שיטת וקטוריזציה היינו צריכים גישה לתוכן שונה בדף של כל מאמר.

לאחר שגילינו מהו שורש ויקיפדיה ומצאנו את הספרייה הנ"ל היו בידינו כל הכלים שהיינו צריכים בכדי לבנות אלגוריתם כרייה רקורסיבי בסיסי עבור ויקיפדיה.

האלגוריתם הבסיסי שיצרנו היה וריאציה של אלגוריתם ה- Preorder לעצים: החל מהשורש (קטגוריית העל 'Main Topic Classifications'), האלגוריתם עובר בכל שלב על הבנים של הקטגוריה הנוכחית: אם הבן הנוכחי הינו תת-קטגוריה הוא מוסיף אותה ואת המאמרים הנמצאים תחתיה למאגר, ואז ממשיך באופן רקורסיבי עבור כל אחת מקטגוריות הבת שלה. במידה והבן הנוכחי הינו מאמר האלגוריתם מוסיף את המאמר למאגר תחת הקטגוריה הנוכחית.

כיוון שויקיפדיה מכילה אלפי קטגוריות ומיליוני מאמרים הוספנו לאלגוריתם גם מגבלות על העומק שאליו יוכל להגיע ועל כמות הקטגוריות והמאמרים שיוכל לכרות בכל רמה, וזאת בכדי לוודא שיסיים לרוץ בזמן סביר. המאמרים והקטגוריות שנכרו תחת מגבלה זו נבחרו בשלב הראשון באופן שרירותי.

בחינת תתי הקטגוריות בהיררכיה שאותן נבחר לאסוף

השאלה הבאה שנצבה בפנינו הייתה: אילו תתי קטגוריות בהיררכיה כדאי לאסוף? ואילו לא כדאי?

- **תיאור הבעיה**

מצד אחד לא רצינו לאסוף תתי קטגוריות כלליות מדי שיוכלו להקשות על סיווג מאמרים ברמות העליונות של העץ. פתרון אפשרי שהצענו לבעיה היה לדלג על השכבות העליונות שנמצאות מיד תחת קטגוריה מרכזית (לדוגמה - מיד תחת הקטגוריה 'היסטוריה' נמצאות תתי הקטגוריות 'היסטוריה מקומית' וגם 'היסטוריה תרבותית' אשר דומות מאוד זו לזו). מצד שני גם לא רצינו לכרות תתי קטגוריות ספציפיות מדי, לדוגמה כאלו שנמצאות בתחתית ההיררכיה, כיוון שבמקרים רבים קטגוריות אלה הכילו מאמרים בודדים בלבד.

- **הדרך לפתרון**

נקדים ונאמר שלאורך תהליך כריית הנתונים ביצענו מספר רב מאוד של ניסויים ו"כיולים" של אלגוריתם הכרייה בכדי להפיק עבור תהליך הלמידה העתידי מאגר נתונים מסודר ונקי מרעשים ככל הניתן. לצורך פשטות חילקנו את השינויים המרכזיים והמהותיים ביותר שביצענו באלגוריתם לשלושה שלבים מרכזיים:

- **מסקנות - שלב ראשון**

לאחר מספר ריצות של האלגוריתם הבסיסי (שלא כלל סינון או בחירה ידנית כלשהי של קטגוריות) שמנו לב שבמהלך הריצות נאספו מספר רב של ערכי 'זבל' מסוגים שונים:

ראשית, מבדיקה ידנית עלה שברמות מסוימות בהיררכיה ישנן מספר רב של קטגוריות-ביניים שתוכן הערך שלהן הינו לא יותר מ-'רשימת מכולת' כללית של תתי-קטגוריות בנושא מסוים. לדוגמא: תתי הקטגוריות Health by Continent/Individual/Country (שנמצאות תחת קטגוריית העל Health). בנוסף, תחת כמעט כל קטגוריה ישנה תת קטגוריה עם הסיומת 'stubs' אשר מכילה ערכים קצרים או ערכים בבניה של אותה הקטגוריה. את הקטגוריות הללו החלטנו שצריך לסנן כיוון שחזינו שהכללתן במאגר תרעיש אותו ותפגע בהמשך בתהליך הסיווג.

שנית, נתקלנו גם במספר לא קטן של מאמרים שהיוו מעין "אינדקסים" עבור מאמרים אחרים, לדוגמא: המאמר Outline of Engineering, שנמצא ישירות תחת הקטגוריה Engineering ונועד להכיל רשימה מחולקת לנושאים של מאמרים חשובים באותה קטגוריה.

כדי למנוע מקטגוריות וממאמרים כנ"ל להרעיש את מאגר המידע שלנו הוספנו לאלגוריתם כריית הנתונים פונקציית סינון המכילה את מילות הפתח הבאות, שראינו שהינן משותפות לרבים מערכים מסוגים אלה:

index of, outline of, by, list of, lists, associated with, comparisons, stubs, in, template

במידה ופונקציית הסינון נתקלת באחת מן המילים הנ"ל בכותרת של קטגוריה או מאמר כלשהו היא מדלגת עליהם בתהליך הכרייה.

- **מסקנות - שלב שני**

לאחר הרצת האלגוריתם המשופר ובחינת מדוקדקת של התוצאות החדשות הגענו למסקנה כי על אף שהצלחנו לסנן את מרבית קטגוריות ומאמרי ה'זבל', המאגר שלנו עדיין לא היה טוב מספיק.

ראשית, ניכר שהבחירה השרירותית שהאלגוריתם מבצע כשהוא נדרש לבחור מספר מוגבל של תתי הקטגוריות מבין אלה הקיימות בכל רמה (זוהי כאמור מגבלה שהצבנו בכדי להגביל את זמן הריצה) לא משיגה תוצאות טובות מבחינת תוכן. הסיבה לכך היא שבמקרים רבים האלגוריתם בחר קטגוריות עם מספר מועט מאוד של מאמרים ותתי קטגוריות, או גרוע מכך, אסף כמה קטגוריות כלליות מידי. מכאן, מכיוון שבחרנו למדל את המאגר שלנו כעץ ללא כפילויות, נוצר מצב שבו הייתה חפיפה רבה במאמרים של מספר קטגוריות ולכן נאלצנו לזרוק מאמרים רבים בכדי לשמור על המודל שלנו כעץ עם מסלול יחיד מהשורש לכל מאמר.

שנית, דבר נוסף שדנו בו בשלב זה היה השאלה מהו גודל המאגר המקסימלי שאיתו נוכל להתמודד מאוחר יותר בתהליך הלמידה באמצעות אמצעי החישוב המוגבלים שעמדו לרשותינו (מחשבינו האישיים).

בכדי לפתור את הבעיות הנ"ל ביצענו מספר שינויים נוספים באלגוריתם הכרייה:

- בכדי לפתור את בעיית כח החישוב החלטנו להגביל את תהליך הכרייה ל-3 רמות בלבד. קיווינו שכך נוכל להגביל את כמות המאמרים שתיאסף לכ-6000 מאמרים בסה"כ.
- בכדי לפתור את הבעיה החפיפה בין הקטגוריות החלטנו להחליף את הבחירה השרירותית של האלגוריתם בבחירה ידנית של תתי קטגוריות בכל רמה. לצורך כך בחרנו בכל רמה מספר קטגוריות בעלות מספר גדול יחסית של מאמרים שגם לטעמנו (באופן גס) אינן חופפות יתר על המידה.

  בסה"כ בחרנו 80 קטגוריות ב-3 רמות, כאשר ברמה הראשונה לדוגמא נבחרו הקטגוריות המרכזיות הבאות:

  Technology, Politics, Health, Crimes, Entertainment

  בנוסף, לעיתים אף החלטנו לדלג על רמה או שניים בהיררכיה בכדי לדלג על קטגוריות כלליות מדי ולהגיע לקטגוריות "מעניינות" יותר שכללו תחתן יותר תוכן. קטגוריות כאלה הן לדוגמא: Crimes שנמצאת תחת קטגוריית העל – Crime. בעוד שקטגוריית העל Crime מרכזת תחתה נושאים כלליים שונים הקשורים לפשיעה (כגון Criminal Law, Criminology וכד'), החלטנו שיהיה מעניין יותר לדלג על הקטגוריה הזו ולקחת דווקא את תת הקטגוריה שלה Crimes, שתתי הקטגוריות שלה מייצגות כל אחת סוג שונה של פשע (Murder, Theft, Corruption וכו').

  את רשימת הקטגוריות שיש לאסוף בכל רמה, האלגוריתם החדש מקבל כפרמטר בקלט.
- נציין בנוסף שתהליך בחירת הקטגוריות היה תהליך ממושך שדרש עשרות ריצות. לכן, בכדי לשפר את זמן הריצה של האלגוריתם הפחנו את האלגוריתם למקבילי החל מהרמה הראשונה, כאשר לאחר הוספת חמשת הקטגוריות המרכזיות האלגוריתם יוצר thread עבור כל קטגוריה מרכזית וכורה את תת העץ שלה בנפרד לתוך מאגר אחד מרכזי.

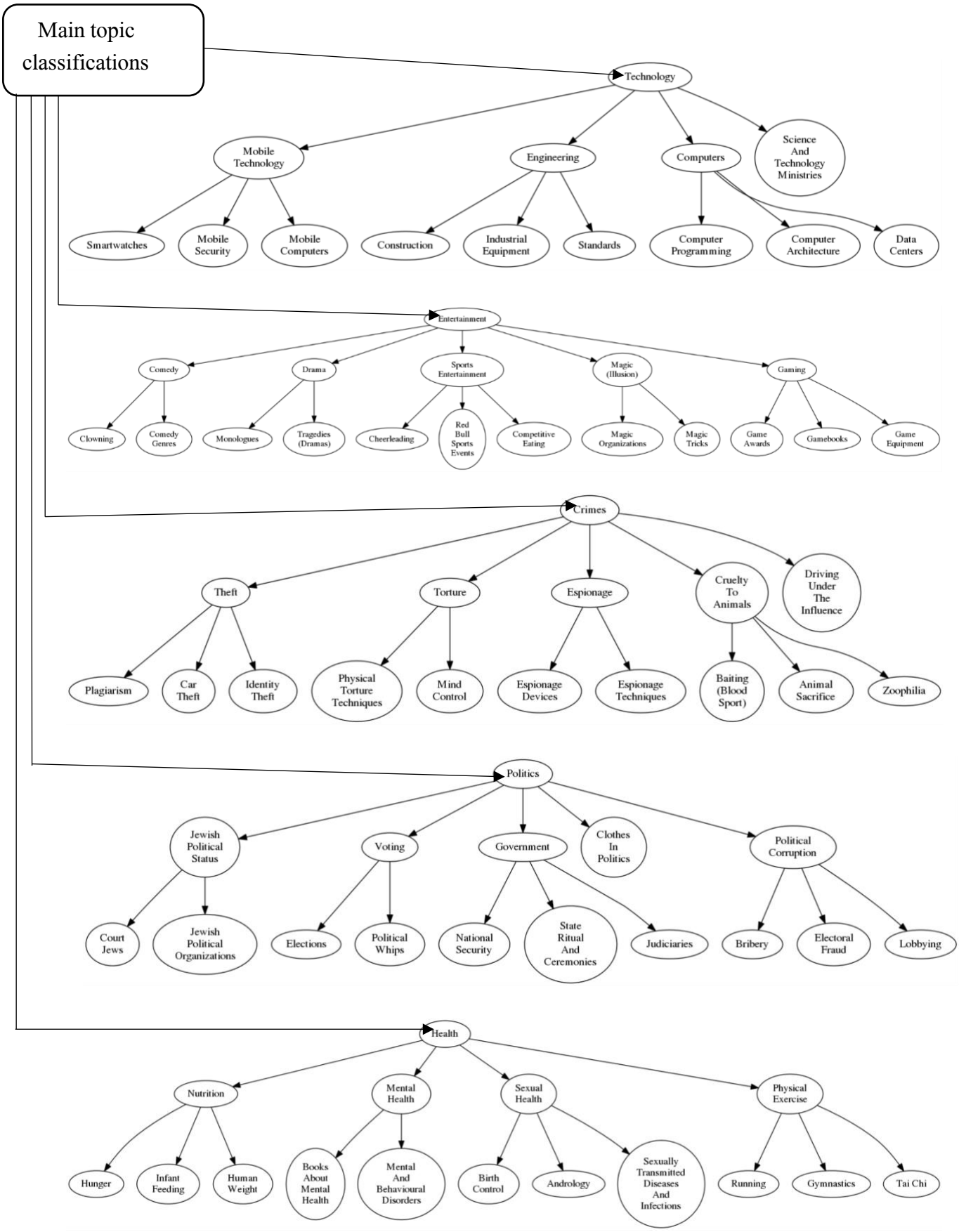
- **מסקנות - שלב שלישי**

בסיום ההרצות של האלגוריתם עם המגבלות החדשות גילינו כי מספר המאמרים שנאספו אינו מספיק גדול ביחס לכמות הקטגוריות שבחרנו, מה שעלול היה להקשות בהמשך על תהליך הלמידה. לפיכך, בכדי להגדיל את כמות המאמרים במאגר החלטנו לכרות מאמרים נוספים מקטגוריות ברמה 3 שלא נבחרו במקור ולהוסיף אותם למאגר הנתונים שלנו כמאמרים ישירים תחת קטגוריית האם המתאימה להם ברמה 2.

גרף המתאר את מאגר המידע

בעמ' הבא מוצג גרף המתאר את היררכיית 81 הקטגוריות הכלולות במאגר המידע הסופי של הפרויקט:

היררכיית הקטגוריות



בהמשך, כאשר נציג את שיטות הוקטוריזציה שבהם בחרנו להשתמש, נציג גם מספר סטטיסטיקות נוספות על המאמרים במאגר המידע שכרינו המתקשרות לשיטות הוקטוריזציה.
את הסטטיסטיקות הנ"ל ניתן למצוא בעמ' 20.

מימוש – אלגוריתם הכרייה ועיבוד המידע

המחלקה Miner

המחלקה אחראית על כל תהליך כריית הנתונים וביצוע הpost-processing למידע שנאסף.
המחלקה מייצרת instance יחיד (singleton) אשר לצורך יצירתו מקבל (בין היתר) את שמה של קטגוריית העל שתשמש בתור שורש תהליך הכרייה ואת רשימת הקטגוריות שיש לאסוף (לפי רמות). לאחר יצירתו האובייקט מריץ את אלגוריתם כריית הנתונים כפי שתואר לעיל ושומר את המידע שנאסף במחלקות המתאימות במבנה הנתונים שיתואר בהמשך.
קוד : Miner.py

עיבוד המידע בתום תהליך הכריה

בסוף תהליך הכרייה מבצע אובייקט ה-Miner גם תהליך post-processing על המידע שנאסף שכולל שני חלקים מרכזיים :

- הוספת מזהים ייחודיים (מספריים) למאמרים ולקטגוריות שנאספו. המזהים המספריים ישמשו אותנו בהמשך לצורך תהליך הלמידה והסיווג כיוון שמודלי הלמידה בהם נשתמש מצפים למחלקות סיווג המסומנות ע"י מספרים.
- חישוב קבוצת ה-Descendant Articles עבור כל קטגוריה. מאמר נחשב descendant (צאצא) של קטגוריה מסוימת אם הוא בן ישיר של הקטגוריה או בן של איזושהי קטגוריה הנמצאת בתת העץ של אותה הקטגוריה. הקבוצה הנ"ל, אשר מחושבת באופן רקורסיבי ע"י מעבר על תת העץ של כל קטגוריה שנכרתה, תשמש אותנו בהמשך לצורך בניית אלגוריתם הסיווג ההיררכי.

קוד : Miner.py

שמירת המידע שנאסף

בסוף שלב ה-post-processing מאגר המידע כולו מומר לפורמט JSON ונשמר לקובץ לוקלי שממנו הוא יוכל להיטען מחדש עבור שימוש בריצות הבאות.
קוד : FilesIO.py (שמירה וטעינה של קבצי JSON)

מימוש - המחלקות ומבנה הנתונים

בתום תכנון אלגוריתם הכרייה דנו בינינו כיצד יהיה ראוי לשמור בצורה מסודרת את הנתונים שיאספו. השאיפה הייתה למדל את מאגר המידע שלנו כמבנה נתונים יחיד שיכיל בתוכו את כל המידע (קטגוריות, מאמרים) בצורה מסודרת, ואותו נוכל בהמשך להעביר לשאר חלקי הפרויקט שיזדקקו למידע.

בכדי להקל השימוש ב-data בשאר חלקי הפרויקט הגענו למסקנה שעל המאגר להיות מסודר בצורה כזו שיאפשר גישה מידית לכל קטגוריה/מאמר שאספנו וזאת תוך שמירת המידע ההיררכי והקשרים בין הקטגוריות והמאמרים. כמו כן על המבנה גם להכיל מראש שדות עבור תוצאות שיתקבלו רק בשלבים מאוחרים יותר של הפרויקט, לדוגמא שדה לשמירת הוקטורים עבור כל מאמר שיתקבלו מאוחר יותר בתהליך הוקטוריזציה.

תיאור המחלקות

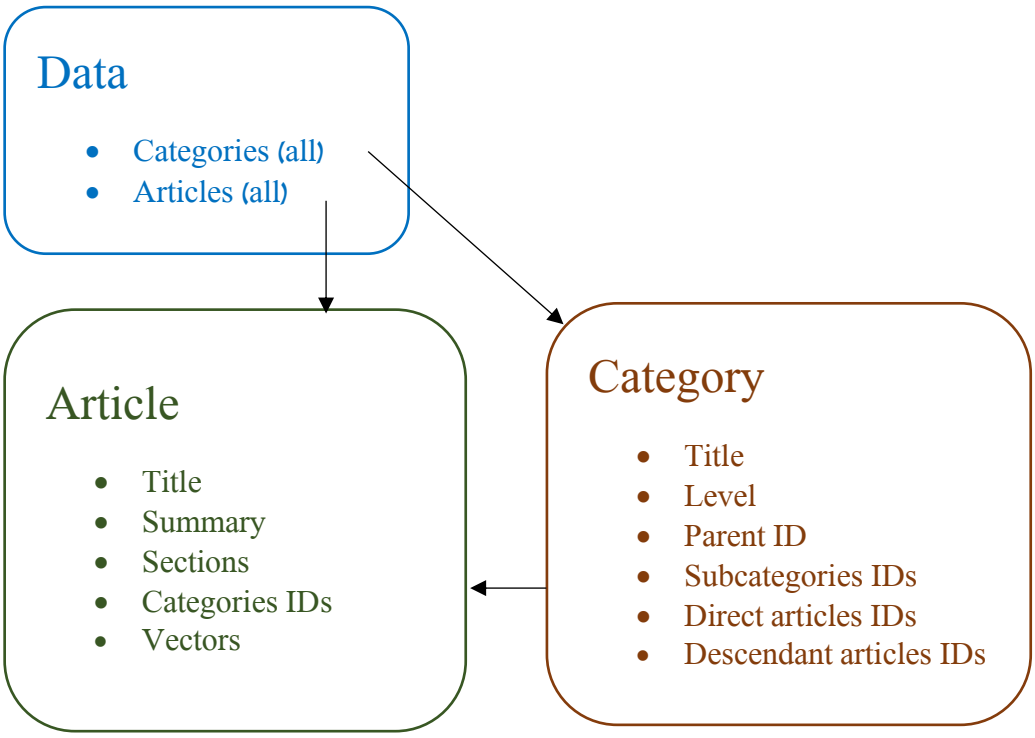
לצורך יצירת מבנה נתונים שיעמוד בדרישות הנ"ל הגדרנו 3 מחלקות מרכזיות (מוגדרות בקובץ DataClasses.py):

Data – המחלקה המרכזית עבור מבנה הנתונים. מאגר הנתונים תוכנן כך שבסופו של דבר הוא ייוצג ע"י instance בודד (singleton) של מחלקה זו. המחלקה מכילה תחתיה שני אוספים (מילונים) מרכזיים: אחד עבור הקטגוריות, והשני עבור המאמרים שנכרו. כך ניתן לגשת לכל מאמר/קטגוריה בצורה ישירה באמצעות המזהה המתאים.

Category – מחלקה המייצגת קטגוריה. לכל קטגוריה שמרנו (בין היתר) את שמה, הרמה בה היא נמצאת, מזהה קטגוריית האם שלה, מזהי קטגוריות הבת שלה, מזהי המאמרים שנמצאים ישירות תחת הקטגוריה, מזהי המאמרים שהינם צאצאים של הקטגוריה ברמה כלשהי מתחתיה.

Article – מחלקה המייצגת מאמר. לכל מאמר שמרנו (בין היתר) את שמו, תוכן המאמר (המחולק ל-summary + sections) ומילון ייעודי עבור וקטורי המאמר (שיתווספו בשלב מאוחר יותר). כמו כן שמרנו גם את מזהי הקטגוריות שתחתן המאמר נמצא, בכדי שנוכל לסנן בשלב מאוחר יותר את המאמרים שנמצאים תחת יותר מקטגוריה אחרת וכך לשמר את מבנה המאגר שלנו כעץ.

המחלקות השונות והקשרים ביניהן מתוארים בתרשים הבא:



- ממבט על החלוקה לעיל עלולה לעלות השאלה מדוע לא שמרנו ישירות את האובייקטים המתקבלים ישירות מהספרייה Wikipedia-API עבור הקטגוריות והמאמרים. הסיבה לכך היא שאובייקטים אלה הכילו המון מידע מיותר, ובנוסף, בשל אופטימיזציה של הספרייה, השדות של האובייקטים הללו אינם זמינים ברובם עד שניגשים אליהם (lazy access). לכן החלטנו ליצור מחלקות ייעודיות משלנו עבור הקטגוריות והמאמרים, ובהם לשמור רק את המידע הרלוונטי.

שלב הוקטוריזציה

בפרק זה נסביר ונתאר את תהליך הוקטוריזציה שביצענו עבור המאמרים שאספנו בשלב כריית הנתונים. נתאר את שיטות הוקטוריזציה שאותן בחרנו לחקור ונתאר כיצד השתמשנו במודל השפה של BERT בכדי לייצר את הוקטורים הללו עבור כל מאמר ומאמר.

שיטות הוקטוריזציה

כפי שהצגנו זאת במבוא, אחת ממטרותיו של הפרויקט שלפניכם הינה להעריך את יכולותיהן של שיטות וקטוריזציה שונות ולהשוות בין הביצועים שניתן לקבל בעזרתן בסיווג של מאמרים. ממבט ראשוני ניתן לנחש באופן אינטואיטיבי שאילו נייצר וקטור אחד מכל מילותיו של מאמר (לדוגמא ע״י חישוב הוקטור הממוצע של כל וקטורי המילים), הוקטור הזה ייצג את נושא המאמר בצורה הטובה והמלאה ביותר. מצד שני, ברור גם שאת הנושא של מרבית הערכים בויקיפדיה ניתן להסיק רק מקריאת פסקת הפתיחה או אפילו מהמשפט הראשון שלהם, ואילו שאר חלקי המאמרים עלולים במקרים מסוימים דווקא להסיח את דעת הקורא מהנושא המרכזי שעליו המאמר מדבר (׳מרוב עצים לא רואים את היער׳...). השאלה המרכזית שעומדת בפנינו אם כך הינה:

מהו החלק במאמר שעיבודו לוקטור יוביל לייצוג הטוב ביותר עבור המאמר כולו בתהליך הסיווג?

בכדי לענות על השאלה הנ״ל, ותוך התחשבות בכוח העיבוד המוגבל שעמד לרשותינו (כפי שנראה בהמשך תהליך בחירת הפרמטרים עבור המסווגים השונים הסתבר להיות יקר מאוד בזמן ריצה), החלטנו להגדיר 3 שיטות וקטוריזציה עבור מאמרים שאותן נחקור במסגרת פרויקט זה:

First-Sentence – ייצור וקטור עבור מאמר תוך שימוש במשפט הראשון שלו בלבד.

Summary – ייצור וקטור עבור מאמר תוך שימוש בפסקת הפתיחה שלו בלבד.

- בויקיפדיה נהוג שפסקת הפתיחה של מאמר מהווה תקציר (summary) למאמר.

Full-Text – ייצור וקטור עבור מאמר תוך שימוש בכל מילות המאמר.

אופן המימוש ויצירת הוקטורים

כפי שכבר ציינו מוקדם יותר כשהצגנו את BERT כמודל שפה (עמ׳ 6) , הכלי (בגרסתו הבסיסית) הינו רשת נוירונים בעלת 12 שכבות שבהינתן משפט קלט אחד (או שניים) פולטת לכל מילה במשפט וקטור בן 768 מימדים המייצג את המשמעות הסמנטית של המילה כתלות בהקשר שלה בתוך המשפט.
מכאן, בהינתן משפט בודד ניתן לייצר וקטור יחיד שייצג את המשפט כולו ע״י הרצת המודל של BERT על המשפט ואז חישוב ממוצע הוקטורים המתקבלים ע״י המודל עבור המילים השונות במשפט. גם הוקטור המתקבל יהיה בן 768 מימדים שכן חישוב הממוצע נעשה כמובן עבור כל מימד בנפרד. באופן זה ייצרנו את הוקטור המייצג את המאמר בשיטת ה- First-Sentence.

באופן דומה, ניתן להכליל את הנכתב לעיל גם לאוסף של משפטים, וכך לייצר וקטור אחד שייצג פסקה או מסמך שלם ע"י חישוב ממוצע הוקטורים של כל משפטיו. כך ייצרנו את הוקטורים עבור השיטות Summary ו-Full-Text.

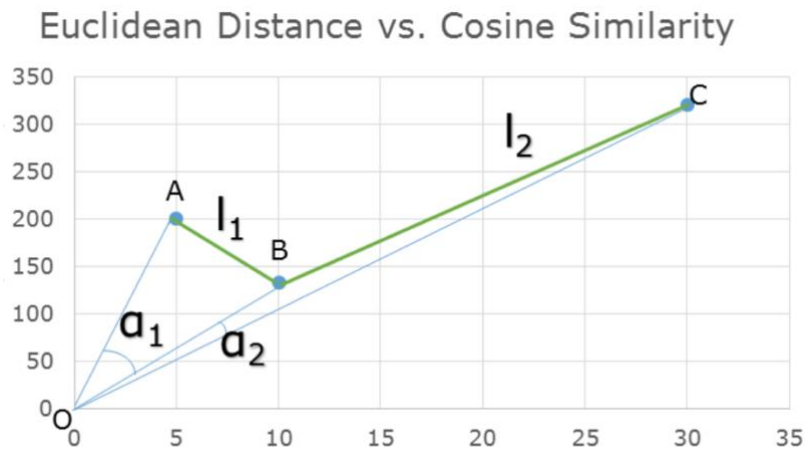
קובץ הקוד המכיל את הרצת BERT ומימוש שיטות הוקטוריזציה הינו : TextVectorizer.py

הערות:

- כאמור, כיוון שתהליך הוקטוריזציה מתבצע ע"י BERT משפט-משפט, בטרם התהליך היינו צריכים לפרק את המאמרים השונים למשפטים המרכיבים אותם. מסתבר שפעולת פירוק מסמכים למשפטים איננה טריוויאלית כיוון שיתכן שתו הנקודה ('.') יופיע לעיתים גם באמצע משפט ולא רק בסופו (למשל אם הביטוי "Dr. Seuss" מופיע באמצע המשפט). בכדי להתגבר על הבעיה הנ"ל השתמשנו בפונקציה ייעודית לכך המכונה 'sent_tokenize' הנמצאת בספריית ה-NLP : NLTK.
- כפי שהוזכר בשלב כריית הנתונים בכל אובייקט המייצג מאמר במאגר הנתונים שלנו יש מילון שנועד לאחסן את כל הוקטורים של המאמר בכל שיטות הוקטוריזציה לקראת תהליך הלמידה.

אופן ההשוואה בין וקטורים

מוקדם יותר, כשהצגנו את מושג ה- word embedding (עמ' 4), הזכרנו שבמודלים רבים ב-NLP, ובפרט ב-BERT, נהוג להשוות בין וקטורי המייצגים מילים מסמכים באמצעות פונקציית ה - Cosine Similarity.
כאמור, בהינתן שני וקטורים פונקציה זו מחזירה ערך בין 0 ל-1, כאשר 1 מסמל התאמה מושלמת ו-0 מסמל חוסר קשר מובהק.
כשנכנסנו לעומק העניין גילינו שמרבית המסווגים שבהם תכננו להשתמש במסגרת הפרויקט אינם בנויים לתמוך בפונקציית הנ"ל לצורך השוואה בין דגימות, כיוון שהם מצפים לקבל דגימות שאופן ההשוואה ביניהן אמור להתבצע באמצעות חישוב מרחק אוקלידי פשוט.
הבעיה שנוצרת היא, ששני וקטורים במרחב שהזוית ביניהן קטנה מאוד אינם בהכרח מייצגים שתי נקודות במרחב הקרובות זו לזו, ולהיפך.
לצורך המחשת הבעיה נביט בשרטוט הבא : [4]



אם נרצה למצוא מהו הוקטור ה"דומה" ביותר לוקטור OB, נקבל שמדובר בוקטור OA אם נבצע את ההשוואה באמצעות מרחק אוקלידי (כיוון ש – l1<l2). לעומת זאת, אם נבצע את ההשוואה באמצעות Cosine Similarity נקבל דווקא שהוקטור OC דומה יותר ל-OB, כיוון שהזווית a2 קטנה מהזווית a1.

הדוגמא לעיל ממחישה מדוע שימוש בוקטורים המתקבלים מ-BERT כקלט למסווגים המתבססים על סקלת מרחק אוקלידי עלול להוביל לתוצאות לא נכונות.
בשלב זה הבנו שאנו נדרש לבצע טרנספורמציה כלשהי לוקטורים המתקבלים מ-BERT בכדי להשתמש בהם עם המסווגים הנ"ל. על הטרנספורמציה ליצור מצב שבו שני וקטורים שעברו את הטרנספורמציה יהיו קרובים זה לזה אוקלידית **אם ורק אם** שני הוקטורים המקוריים היו קרובים זה לזה מבחינת הזווית בינהם.

הפתרון לבעיה פשוט יותר מכפי שניתן לחשוב: כל שצריך לעשות הוא לנרמל את הוקטורים המתקבלים.

אינטואיטיבית, לאחר הנרמול כל הוקטורים שיתקבלו יהיה באורך קבוע וזהה R כך שהנקודה המיוצגת ע"י כל אחד מהם תשב על ספירה n מימדית ברדיוס R. כיוון שנרמול וקטורים לא משפיע על הכיוון שלהם הזוויות בין הוקטורים השונים לא ישתנו לאחר הנרמול. מכאן, כיוון שמיתרים שווים בתוך ספירה נשענים על קשתות שוות, וכיוון שככל שמיתר ארוך יותר כך הקשת עליה הוא נשען ארוכה יותר ולהיפך, לאחר הנרמול סקלת המרחק האוקלידי תהיה שקלה לסקלת הזויות עבור כל הנקודות הנמצאות על הספירה.
לכן נוכל בהמשך להשתמש בוקטורים המנורמלים כקלט עבור המסווגים השונים המשווים באמצעות מרחק אוקלידי מבלי לפגוע בנכונות המידע והאלגוריתמים.

סטטיסטיקות על מאגר המידע בקשר שיטות הוקטוריזציה

כעת, משהצגנו את שיטות הוקטוריזציה השונות בהן השתמשנו נרצה להציג מספר סטטיסטיקות אודות המאמרים במאגר המידע שלנו המתקשרות לשיטות הנ"ל:

מס' מילים ממוצע במשפט הפתיחה	מס' מילים ממוצע בתקציר	מס' מילים ממוצע במאמר	מס' המאמרים	רמה
26.9	161	750.8	265	1
26.3	139.5	736.5	4781	2
25.5	135.7	651.1	2579	3
26.2	145.4	712.8	2541.7	AVG

מס' מילים ממוצע במשפט הפתיחה	מס' מילים ממוצע בתקציר	מס' מילים ממוצע במאמר	# מאמרים	קטגוריה
28.4	148.4	694	1472	Politics
24.6	143.2	667	1566	Health
27.3	141.5	786.5	1167	Crimes
25.7	134.7	543.6	1517	Technology
23.3	126.5	713.6	1289	Entertainment
25.9	138.9	680.9	1402.2	AVG

ראשית, ניתן לראות שכמות המאמרים שכרינו עבור כל קטגוריה ראשית במאגר הנתונים היתה פחות או יותר מאוזנת, ללא הבדלים קיצוניים בין הקטגוריות. מבחינת התפלגות המאמרים בין הרמות לעומת זאת, ניתן לראות כצפוי גידול משמעותי בכמות המאמרים בין הרמה הראשונה לשנייה (כמעט פי 20), כאשר לאחר מכן נראית ירידה דרסטית בכמות המאמרים ברמה השלישית. הסיבה לירידה זו בין הרמה השנייה לשלישית נעוצה בעיקר באלגוריתם כריית הנתונים, שכפי שתיארנו מנסה "לעבות" את הרמה השנייה בהיררכיה באמצעות מאמרים מהרמה השלישית שהקטגוריות שלהם לא נכרו.

מבחינת כמויות המילים בכל אחד מחלקי המאמרים השונים ניתן כצפוי לראות את ההבדל המשמעותי בין כמות המילים הממוצעת במשפטי הפתיחה, לבין הכמויות בתקצירי המאמרים ובמאמרים בשלמותם. באופן ממוצע במשפט פתיחה של מאמר יש רק כ-18.6% מכמות המילים שיש בתקציר המאמר, ורק כ-1.8% (!) מכמות המילים שיש במאמר כולו. מכאן ניכר בבירור יתרונן של שיטות הוקטוריזציה Full Text ו-Summary על פניה שיטת ה-First Sentence מבחינת כמות האינפורמציה הנכללת בהן, ולכן יהיה מעניין לראות האם ההבדלים הדרסטיים הנ"ל בכמויות האינפורמציה יבואו גם לידי ביטוי בתוצאות הסיווג שישיגו השיטות השונות בתהליך הלמידה בהמשך.

שלב הלמידה

בפרק זה נתאר בהרחבה את תהליך הלמידה המקדים שביצענו עבור המסווגים השונים.
לתהליך הלמידה המקדים בפרויקט שלנו מטרה עיקרית אחת: למצוא לכל מסווג את אוסף הפרמטרים הטובים ביותר עבורו (Hyperparameter Tuning) עבור כל קומבינציה של קטגוריה ושיטת וקטוריזציה.
אימון המסווגים השונים על כל קומבינציה כנ"ל ומציאת הפרמטרים הטובים ביותר עבורה לכל מסווג ישמשו אותנו למספר דברים:

1. השוואה בין שיטות הוקטוריזציה השונות - כדי להשוות בין שיטות הוקטוריזציה השונות נרצה לבחון כיצד כל מסווג מתמודד עם כל אחת מהשיטות ואיזו עדיפה עבורו. על מנת לעשות זאת נרצה קודם לכוון את הפרמטרים של אותו המסווג על כל אחת משיטות הוקטוריזציה בנפרד.
2. השוואה בין מסווגים - אנו מעוניינים לראות כיצד המסווגים השונים מתמודדים עם בעיית הסיווג ואיזה מסווג או משפחה של מסווגים עדיפים לפתרון הבעיה שלנו.
3. בניית מאגר פרמטרים עבור המסווג הסופי – כזכור, אחת ממטרותיו של פרויקט זה הינה לבנות מסווג היררכי יחיד שבהינתן מאמר חדש יחזיר את מסלול הסיווג המתאים למאמר בהיררכיית הקטגוריות במאגר. מאגר הפרמטרים אותו נבנה בפרק זה יסייע לנו בהמשך בבניית המסווג הסופי הנ"ל כפי שנראה בפרק הבא.

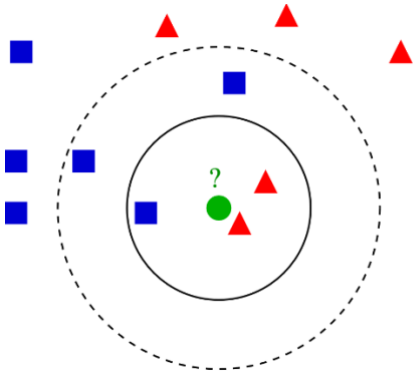
חלוקת הנתונים

את מאגר המאמרים שלנו חילקנו לשלוש קבוצות: למידה (train), אימות (validation) ומבחן (test).
הגודל של קבוצות האימות והמבחן הוא 15% כל אחת, כאשר בתוך כל קבוצה הקפדנו שהכמות היחסית של מאמרים בכל קטגוריה תשמר (stratified split).
קבוצת הלמידה שימשה אותנו ללמידה לצורך תהליך כיול הפרמטרים עבור המסווגים השונים.
את בחינת טיב הפרמטרים ביצענו על קבוצת האימות.
את קבוצת המבחן שמרנו בצד והיא שימשה אותנו להערכת טיב המודל הסופי (המסווג ההיררכי) אותו נתאר בפרק הבא.

נתחיל בתיאור התהליך ע"י הצגת מסווגי הבסיס איתם עבדנו, רקע על אופן פעולתם ואת הפרמטרים שבחרנו לכייל עבורם.

מסווגי הבסיס

k-Nearest Neighbors (KNN)



מסווג זה מתבסס על ההנחה כי לדוגמאות בעלי תכונות דומות, הצמודות זו לזו במרחב כנראה יהיה גם סיווג דומה.
בתהליך האימון המסווג שומר את דוגמאות האימון והסיווג שלהם.
לאחר מכן בתהליך הסיווג כאשר מתקבלת דוגמה חדשה המסווג מחפש את k דוגמאות האימון שהכי קרובות אליה – שכניה הקרובים ביותר - ועל פי הצבעת רוב בין שכנים אלה מסווג את הדגימה החדשה.

הפרמטרים לכיול

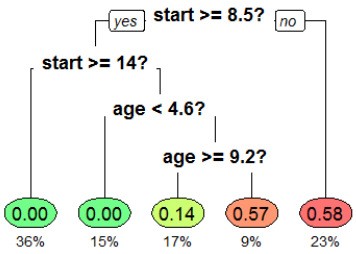
k - כמות השכנים הקרובים ביותר שעל פיהם הצבעתם יתבצע הסיווג.

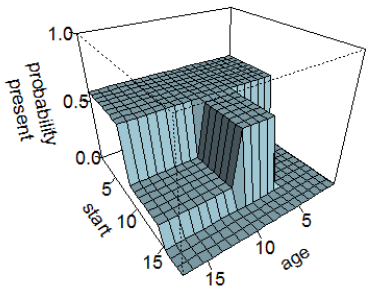
השערה ראשונית ויתרונות

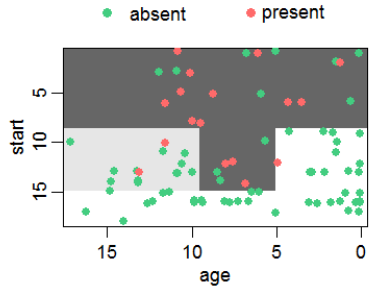
בשל התכונות המאפיינות את המידע שלנו (וקטורים מנורמלים של מאמרים), נצפה לראות התאמה בין הצלחת המסווג (בתום תהליך הכיול) לבין שיטת וקטוריזציה טובה. הסיבה לכך היא שעבור שיטת וקטוריזציה טובה אנו מצפים כי וקטורים בעלי תיוג זהה יהיו קרובים זה לזה במרחב - ואלו בדיוק השכנים שאותם בוחן המסווג.
כלומר, אנו מצפים כי מסווג זה יתרום לנו מידע נוסף לגבי הפיזור הוקטורים במרחב (מכיוון שהם מנורמלים הם נמצאים על "כדור" n- מימדי).

עץ החלטה (ID3)

מסווג זה הינו בעצם עץ החלטה שבו בכל צומת פנימי נשאלת שאלה על אחת התכונות של הדוגמה לסיווג, ובהתאם לתשובות נבחר המסלול הנכון עד לעלה - שלפיו מתבצע הסיווג.
העץ נבנה בהתאם לאלגוריתם Id3 כפי שלמדנו אותו בקורס 'מבוא לבינה מלאכותית': בכל שלב נבחרת התכונה שפיצול לפיה יוביל לתוספת האינפורמציה הגדולה ביותר (ירידה מקסמלית באנטרופיה). התהליך מסתיים כאשר נגמרות התכונות לפיצול (ובמקרה זה הצומת מוגדר כעלה המסווג לפי המחלקה הנפוצה ביותר בקרב דגימותיו), או כאשר כל הדגימות בצומת הנוכחי משתייכות לאותה מחלקה (ובמקרה זה הצומת מוגדר כעלה המסווג לפי מחלקה זו).







- תחת הספרייה sklearn מסווג זה מכונה 'Decision Trees'.

הפרמטרים לכיול

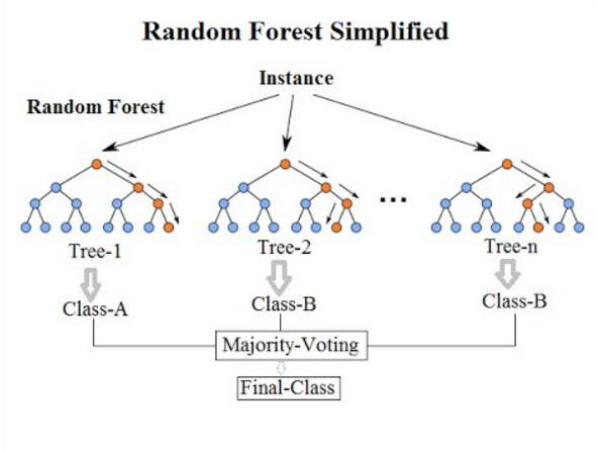
עבור מסווג זה בחרנו לכייל את הפרמטרים העיקריים אשר משפיעים על אופן ביצוע הגיזום:
min_samples_split: מספר הדוגמאות המינימלי הדרוש שיהיה בצומת פנימי על מנת שיתבצע פיצול.
min_samples_leaf: מספר הדוגמאות המינימלי הדרוש בצומת כדי שיחשב כעלה. נקודת פיצול בכל שלב באלגוריתם תילקח בחשבון אם ורק אם היא תשאיר לפחות min_samples_leaf דגימות בכל אחד מן הענפים הימני והשמאלי של תת העץ.

השערה ראשונית ויתרונות

המודל של מסווג זה פשוט מאוד ביחס לשאר המסווגים שנראה בהמשך (שמשתמשים בוועדות או במודלים מתוחכמים הרבה יותר). החיסרון המרכזי של מסווגי עץ סטנדרטיים כמו מסווג ה-ID3 הוא שהם בונים עץ יחיד אשר בא לספק 'נקודת מבט' יחידה בהתבסס על כל הדגימות והתכונות ב-dataset. לכן אין אנו מצפים להצלחה מרובה ממסווג זה.
עם זאת, בחרנו לבחון גם את המסווג הזה ממספר סיבות:

- ע"י כיול פרמטרי הגיזום של העץ אנו בעצם מבצעים Embedded feature selection. בשיטות מסוג זה מנגנון בחירת התכונות מוטמע בתהליך האימון. כאשר מתבצע גיזום בעץ החלטה, האלגוריתם למעשה בוחר לא להשתמש בתכונות נוספות. אנו נרצה להשתמש ביכולת זו של המסווג בכדי לבדוק האם גיזום העץ משפיע על טיב הסיווג, וכך נוכל לקבל אינדיקציה האם שימוש בכל התכונות (768 מימדים) המרכיבות את הוקטורים המתקבלים ע"י BERT אכן הכרחי לתהליך זה.
- שיפור ההערכה לגבי טיב הסיווג של המסווגים האחרים (לדוגמא עבור המסווג Random Forest, שאותו נציג מיד).
- הרחבת מרחב החיפוש אחר המסווג האופטימלי.

Random Forest



מסווג ה- Random Forest בא לפתור את בעיית ה'עומק' של מסווג ה-ID3 (שנעוצה כאמור בנק' מבט יחידה על ה-dataset) ע"י כך שהוא בונה ועדה של עצים ובסוף מגיע להחלטה ע"י ביצוע הצבעה בקרב כלל חברי הועדה. על מנת ליצור גיוון ב'נקודות הראיה' של חברי הועדה בכדי ליצור כל עץ האלגוריתם מגריל מספר דגימות ותכונות באקראי מתוך ה-dataset ומהן בונה את העץ באופן דומה לאופן שבו נבנה עץ באלגוריתם ID3. כך נוצרים מגוון עצים השונים זה מזה, ומכאן נגזר גם שמו של האלגוריתם.
מכיוון שהבחירה של מסווג זה נעשית על פי החלטה של מספר גדול של עצים, החלטות שגויות נקודתיות של עצים בודדים פחות באות לידי ביטוי.

הפרמטרים לכיול

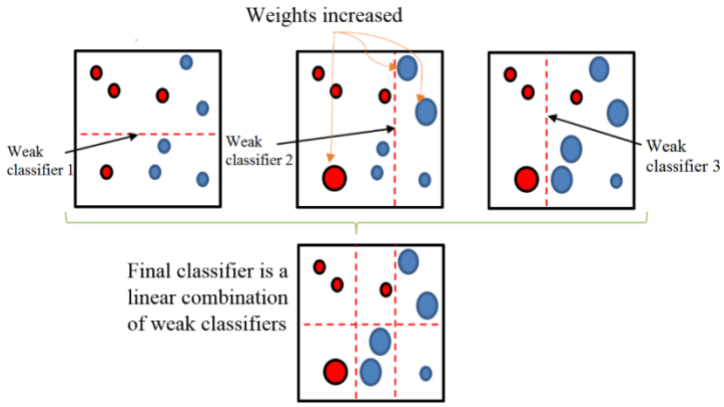
בדומה למסווג ה-ID3 גם במקרה זה בחרנו לכייל את פרמטרי הגיזום: min_samples_split ו-min_samples_leaf.

בנוסף לכך נכייל גם את הפרמטר n_estimators הקובע את מס' העצים החברים בועדה.

השערה ראשונית ויתרונות

נצפה לראות שיפור משמעותי ביחס לתוצאות של מסווג ה-ID3, אך לא נצפה שמודל זה ישיג את התוצאות הטובות ביותר שכן נראה בהמשך שישנם מודלים מתוחכמים יותר ממנו.

AdaBoost



מסווג ה-AdaBoost הינו מסווג המבצע תהליך המכונה 'boosting'. מסווגים מסוג זה יוצרים באופן איטרטיבי ועדות המבוססות על מסווגים 'חלשים' (מסווגים שאחוזי הדיוק שלהם טובים קצת יותר מהטלת מטבע – 50%) בכדי ליצור בסופו של דבר מסווג אחד חזק.

המסווג החלש בו מסווג ה- AdaBoost משתמש הוא עץ סיווג הנקרא stump. עץ זה הינו עץ סיווג בו הדגימות מפוצלות באופן בינארי לפי תכונה אחת בלבד, ולפיכך גובה העץ הינו 1.

בהינתן מגבלה על כמות המסווגים החלשים האלגוריתם עובד באופן הבא:

ראשית המסווג נותן משקלים שווים לכל הדגימות. לאחר מכן, עבור כל תכונה ב-dataset המסווג יוצר stump, מסווג את ה-dataset לפיה ובודק את אחוז הדיוק של אותו הstump. אז המסווג בוחר את ה-stump בעל אחוז הדיוק הגבוה ביותר מבין הנ"ל ומגדיר אותו כעץ הראשון בועדה, ומעניק לו משקל ביחס לאחוז הדיוק שלו. בשלב הבא המסווג מזהה את הדגימות שבהן stump זה טעה, ומגדיל את המשקלים שלהן ביחס למשקלים של הדגימות האחרות. כעת, המסווג יגריל מתוך הdataset הנוכחי מספר של דגימות, כאשר לדגימות בעלות משקל גבוה יותר תינתן עדיפות על פני שאר הדגימות, ומהדגימות הללו יצור dataset חדש. לבסוף על פי ה-dataset החדש המסווג יבנה את ה-stump השני באותו אופן שבו בנה את הראשון, וכן הלאה, באופן איטרטיבי. כך בעצם כל עץ בסדרה נבנה על פי הטעויות של העץ הקודם לו.

לבסוף, המסווג יקבל החלטה ע"י ביצוע הצבעה משוקללת בין כל החלטות עצי הועדה, תוך התחשבות במשקל של כל עץ (כאמור – עץ שהציג דיוק גבוה יותר בעת בנייתו יקבל משקל גבוה יותר, ועץ שהציג דיוק נמוך יקבל משקל נמוך, ובמקרים מסוימים אפילו משקל שלילי).

הפרמטרים לכיול

n_estimators: מספר המסווגים החלשים בועדה המתקבלת.

כיוון שכל עץ בועדה מבוסס על תכונה אחת בלבד נוכל גם ללמוד מכיול פרמטר זה כמה תכונות (מימדים) מתוך ה-768 המרכיבות את המרחב הוקטורי שיוצר BERT אכן נחוצות בכדי לקבל סיווג מוצלח.

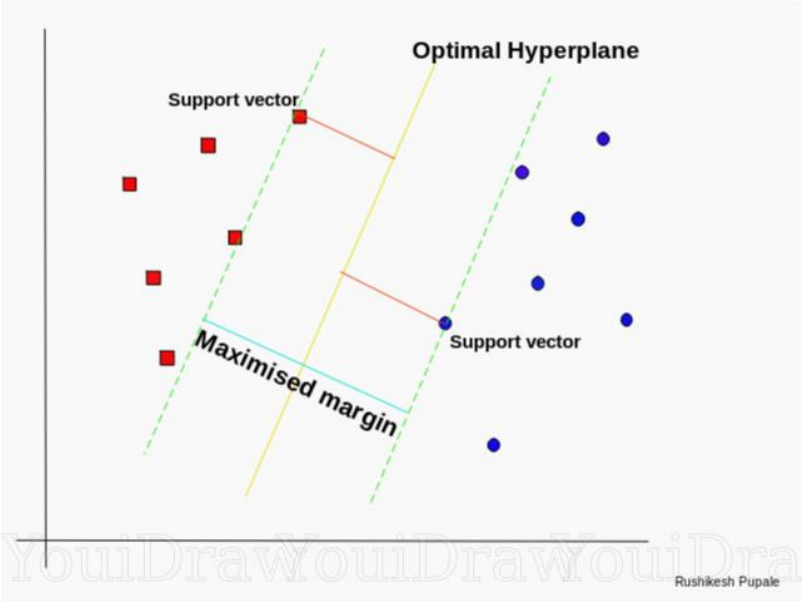
השערה ראשונית ויתרונות

בחרנו לבחון את המסווג על מנת שנוכל לקבל מידע על היכולות שלו עבור בעיית סיווג שלנו. כיוון שמסווג זה מבוסס על ועדה אנו צופים שהוא יהיה עדיף על מסווג ה-ID3 אך איננו יודעים באיזו מידה. כמו כן אנחנו לא בטוחים האם הוא בסופו של דבר ישתייך לקבוצת המסווגים העדיפים לבעיה שלנו.

SVM

Support Vector Machine הינו מסווג המנסה להבחין בין מחלקות במרחב n מימדי באמצעות מציאת מפרידים ליניאריים בדמות hyperplanes (מישורים ליניאריים בני n-1 מימדים). בהינתן שתי מחלקות סיווג לדוגמא, האלגוריתם ינסה למצוא מפריד אופטימלי בין המחלקות כך שהמרחקים ממנו לדוגמאות הקרובות ביותר מכל אחת מהמחלקות יהיו מקסימליים. המרחקים הנ"ל מכונים margins (שוליים), ואילו הדוגמאות הקרובות ביותר אל המפריד מכונות support vectors. כלומר מטרת האלגוריתם הינה למצוא את המפריד עם השוליים הרחבים ביותר.

[5]



כמובן שברוב מאגרי הנתונים במציאות לא יהיה ניתן להפריד בין הדגימות במרחב בצורה מושלמת באופן ליניארי, ולכן במקרים אלה האלגוריתם משתמש בשני פתרונות מרכזיים:

- Soft Margins – בשיטה זו המסווג מוכן לקבל את העובדה שיהיו כמה דגימות בסט האימון שלא יסווגו כמו שצריך וינסה לאזן בין מציאת המפריד עם השוליים הרחבים ביותר לבין המפריד שיספק את המס' הנמוך ביותר של שגיאות הסיווג. הפרמטר המשפיע על האיזון הנ"ל מכונה 'C' ואותו המסווג מקבל בקלט שלו. ככל שערכו של C גבוה יותר כך הקנס הניתן עבור דוגמאות המסווגות לא נכון גבוה יותר.

ערך גבוה מדי של פרמטר זה עלול לגרום ל-overfitting על קבוצת האימון, בעוד שערך נמוך מדי עלול לגרום ל-underfitting.

- Kernel Tricks – בשיטה זו המסווג מבצע טרנספורמציות על תכונות קיימות של הדאטה בכדי לייצר תכונות חדשות. התקווה היא שבמימדים החדשים שיתווספו הדוגמאות של המחלקות השונות יופרדו זו מזו באופן שבו יהיה אפשר להפריד בינהן באמצעות מישורים ליניאריים (בעוד שבמימדים המקוריים מפרידים אלה לא יהיו בהכרח לינאריים). סוג הטרנספורמציה שמופעלת על התכונות תלויה בקרנל של המסווג, שגם הוא פרמטר שניתן לבחירת המשתמש. קרנלים לדוגמא הם : rbf, polynomial, sigmoid, linear.

הפרמטרים לכיול

kernel : בחרנו להשתמש באחד מהבאים :

- linear - מנסה למצוא מפרידים לינאריים בלבד.
- rbf - (radial basis function) – בשיטה זו מייצרים תכונות חדשות ע"י מדידת המרחקים בין כל הדגימות במרחב לבין נקודות קבועות המהוות "מרכזים". על מרחקים אלה מפעילים פונקציה אסקפוננציאלית הפולטת לכל דגימה קואורדינטה חדשה במימד נוסף. כך מייצרים מימדים חדשים שבהם המסווג מנסה שוב להפריד בין הדגימות באופן ליניארי. לפונקציה האסקפוננציאלית הנ"ל יש פרמטר המכונה 'gamma', הקובע את מידת השפעתן של תכונות חדשות על גבולות המפריד המתקבל. בדומה לפרמטר ה-C, ערכים גבוהים של gamma עלולים להוביל לגבולות מפותלים יותר וכך ל-overfitting בעוד שערכים קטנים מדי עלולים להוביל ל-underfitting.

gamma : הפרמטר הניתן לקרנל ה- rbf המתואר לעיל.

C : הפרמטר המשפיע על שיטת ה-Soft Margins כפי שתוארה לעיל.

השערה ראשונית ויתרונות

מסווג זה ידוע כאחד המסווגים החזקים ביותר עבור בעיות קלסיפיקציה, ולכן אנו מצפים לקבל ממנו תוצאות טובות מאוד.

רשת נוירונים

אחת מאבני הבניין המרכזיות של הפרויקט שבנינו הינה הרשת העצבית, שכן כפי שכבר תיארנו BERT הינו מודל למידה עמוקה המבוסס על רשתות עצביות.

כיוון שכלי מתקדם זה אינו נלמד במסגרת הקורס "מבוא לבינה מלאכותית", החלטנו שראוי יהיה להרחיב עליו מעט במסגרת הפרויקט. לכן רקע מורחב על מסווג זה ופרמטריו השונים ניתן למצוא בנספח שבסוף הדו"ח (עמ' 50).

הפרמטרים לכיול

Units – מס' הנוירונים בכל שכבה חבויה של הרשת.

Epochs – מס' המעברים השלמים על ה-dataset המתבצעים בתהליך ה-fitting של הרשת.

Optimizer – אלגוריתם האופטימיזציה של הרשת. הפרמטר משפיע על איכות ומהירות ההתכנסות של הרשת. נרצה להשוות בין אלגוריתם ה-SGD הרגיל (Stochastic Gradient Decent) כפי שנלמד בקורס 'אלגוריתמים נומריים' לבין אלגוריתם חדש יותר המכונה 'adam' המבוסס על אלגוריתם ה-SGD ומתיימר להתכנס מהר יותר, ולכן אולי להשיג תוצאות טובות יותר בפחות epochs.

- בשלב זה נציין שבחרנו להשתמש בשתי שכבות חבויות עבור כל קומבינציה של פרמטרים שאותה נבחן ברשת הנוירונים.

השערה ראשונית ויתרונות

זהו מודל הסיווג המתוחכם והחדש ביותר בתחום הבינה המלאכותית, אולם ידוע שבכדי להשיג תוצאות טובות הוא מצריך dataset עצום וזמן tuning ארוך על חומרה ייעודית (GPU). לפיכך, על אף שאנו מצפים לקבל ממסווג זה תוצאות טובות מאוד צריך לזכור שמאגר הנתונים שלנו לא כל כך גדול ובנוסף את תהליך האימון נריץ על המעבדים (CPU) של המחשבים האישיים שלנו ולכן סט הפרמטרים שעליו נכייל את הרשת לא בהכרח יהיה מושלם כפי שהיינו רוצים.

אלגוריתם בחירת הפרמטרים

פונקציית הערכת טיב פרמטרי המסווגים

הרעיון

המאמרים שאספנו אינם בהכרח מהווים את ההתפלגות האמיתית של הקטגוריות ואף ישנן קטגוריות אשר כלל לא הוכנסו לתהליך כריית הנתונים מלכתחילה. לכן, נרצה להגדיר פונקציית ניקוד (score) אשר תנסה לתגמל קונפיגורציות של מסווגים שלכאורה יתאימו לכל קטגוריה שהיא.

הגדרה פונקציית ה-score בשלב האימון

בכדי להשיג את המטרה הנ"ל נרצה למצוא לכל מסווג את קונפיגורציית הפרמטרים הטובה ביותר עבורו בממוצע על כל הקטגוריות.

לצורך כך נחשב עבור כל קונפיגורציה של המסווג את מדד ה-recall שהשיגה הקונפיגורציה לכל קטגוריה. כזכור, מדד ה-recall שווה ליחס: tp/(tp+fn), כלומר מספר הדגימות השייכות לקטגוריה שסווגו נכון מתוך כלל הדוגמאות השייכות לקטגוריה.

כעת, נגדיר את הניקוד עבור הקונפיגורציה כממוצע של שלושת ערכי ה-recall הנמוכים ביותר שהתקבלו עבור הקטגוריות השונות. כך נקבל שלשלושת הקטגוריות שאיתן הקונפיגורציה הכי התקשתה תהיה השפעה שווה על הציון שלה, כאשר שימוש בממוצע גם ימנע הטיה גדולה מידי במקרה של קטגוריה סוררת, כזו שמרבית הקונפיגורציות יתקשו להתמודד איתה בצורה טובה.

אלגוריתם הבחירה

אלגוריתם בחירת הפרמטרים שכתבנו מבצע חיפוש ממצה (מעבר כל הקונפיגורציות) על סט פרמטרים שהוגדר מראש לכל מסווג בשיטת Cross-Validation.

האלגוריתם פועל כך:

- אתחל מילון ריק.
- לכל קומבינציה של מסווג, שיטת וקטוריזציה וקטגוריה (הנמצאת באחת מבין שלושת הרמות העליונות בהיררכיה) בצע:
  - עבור על כל קומבינציית פרמטרים אפשרית של המסווג מתוך הסט שהוגדר מראש, ובחן את המסווג על קבוצת הלמידה עם הקומבינציה הנ"ל בשיטת Cross Validation. בתום כל מעבר על קומבינציה כנ"ל שמור בצד את תוצאת ה-score שהמסווג השיג.
  - אמן את המסווג פעם נוספת על קבוצת הלמידה כולה עם קומבינציית הפרמטרים שהשיגה את תוצאת ה-score הגבוהה ביותר בשלב הקודם, ובחן אותו על קבוצת האימות. שמור במילון את תוצאת ה-accuracy שהתקבלה ואת קומבינציית הפרמטרים הנ"ל עבור השלשה המיוצגת ע"י הקטגוריה, המסווג ושיטת הוקטוריזציה הנוכחיים.

בתום ההרצה האלגוריתם מחזיר מילון המשמש כפונקציה שבהינתן קטגוריה, מסווג, ושיטת וקטוריזציה מחזירה את סט הפרמטרים הטוב ביותר של המסווג עבור הקטגוריה ושיטת הוקטוריזציה, ואת תוצאת ה-accuracy שהשיג המסווג עם פרמטרים אלה על קבוצת האימות.

מילון זה ישמש אותנו בהמשך לצורך בניית המסווג ההיררכי הסופי.

הערה:

- באלגוריתם הנ"ל כאשר אנו בוחנים את טיב הפרמטרים על קבוצת האימות אנו משתמשים בפונקציית ניקוד שמסתכלת על דיוק בלבד (accuracy). הרעיון מאחורי הבחירה הזו הוא שכאשר כבר יש בידנו מסווג עם הפרמטרים הטובים ביותר שלו, כבר פחות מעניינת אותנו הצלחתו של המסווג לפי פרמטר ה-recall, אלא נרצה כעת דיוק אבסולוטי גבוה של המסווג על הקטגוריה. אנו מאמינים שכך יהיה בידנו מידע מועיל שבעזרתו נוכל להשוות בין שיטות הוקטוריזציה ובין המסווגים השונים, ובנוסף נוכל להשתמש במידע זה בהמשך בכדי לבנות את המסווג ההיררכי הסופי.

מימוש

את החיפוש הממצה באלגוריתם בחירת הפרמטרים ביצענו באמצעות המחלקה GridSearchCV מספריית sklearn. נציין שלמחלקה זו יתרון משמעותי בכך שהיא מסוגלת לבצע את החיפוש בעזרת ניצול כל הליבות של המחשב בצורה מקבילית.

את קוד אלגוריתם בחירת הפרמטרים ניתן למצוא בקובץ: ParameterSelector.py.

את סטי הפרמטרים של המסווגים השונים שעליהם הרצנו את האלגוריתם הגדרנו בקובץ: Classifiers.py.

בקובץ זה ניתן גם למצוא את הקוד הממש את רשת הנוירונים, שאותה בנינו באמצעות הספרייה Keras עם backend המבוסס על TensorFlow.

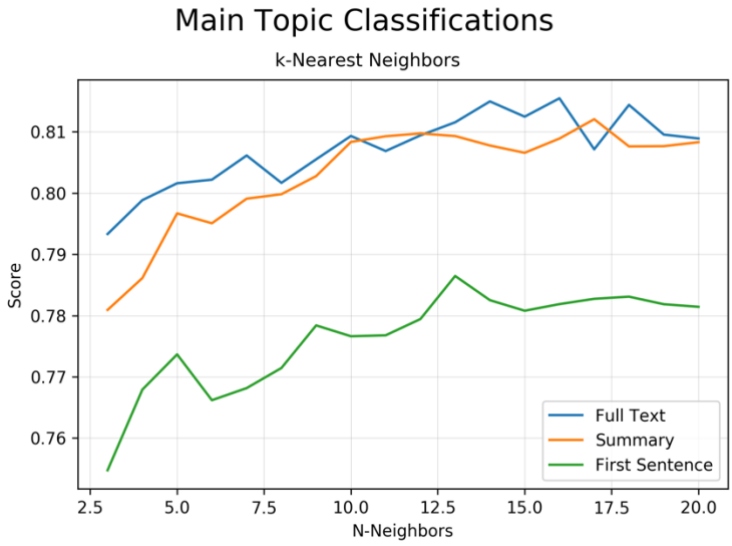
הערכת שיטות הוקטוריזציה והמסווגים

בפרק זה נשווה בין שיטות הוקטוריזציה עבור כל אחד מהמסווגים שבחרנו.
לכל מסווג יצרנו גרפים עבור הקטגוריות ברמה הראשונה והשנייה בהיררכיה. כל גרף מייצג את תוצאות תהליך הכוונון של פרמטר יחיד של אותו מסווג (אשר מוצג בציר ה-x) ובציר y נמצאת פונקציית המטרה (score) כפי שתיארנו אותה בפרק הלמידה. עבור מסווגים בעלי מספר גדול מ-1 של פרמטרים קיבענו בכל גרף את שאר הפרמטרים (אלה שאינם מופיעים בציר ה-x) לפי הערך שקיבלו בקומבינציה שהשיגה את תוצאת ה-score הגבוהה ביותר.
הגרפים מיוצרים בקוד בקובץ GraphsCreation.py, וניתן למצוא את כולם תחת התיקייה: Parameter Selection/Graphs.
עבור כל מסווג בחרנו שני גרפים שלדעתנו מציגים בצורה טובה את הקשר בין שיטות הוקטוריזציה למסווג.
עבור כל אחד נרצה לבחון את ההשפעה של פרמטרי המסווג ושיטות הוקטוריזציה על תוצאות הסיווג וננסה לשער מה היו הסיבות לתוצאות שהתקבלו.

מכאן ואילך נגדיר לצורך הדיון את המושגים הבאים:
הקטגוריה הראשית - Main Topic Classifications
קטגוריות המשנה - 5 קטגוריות הבת של הקטגוריה הראשית, כפי שהוזכרו בפרק כריית הנתונים

k-Nearest Neighbors

השפעת פרמטר השכנים

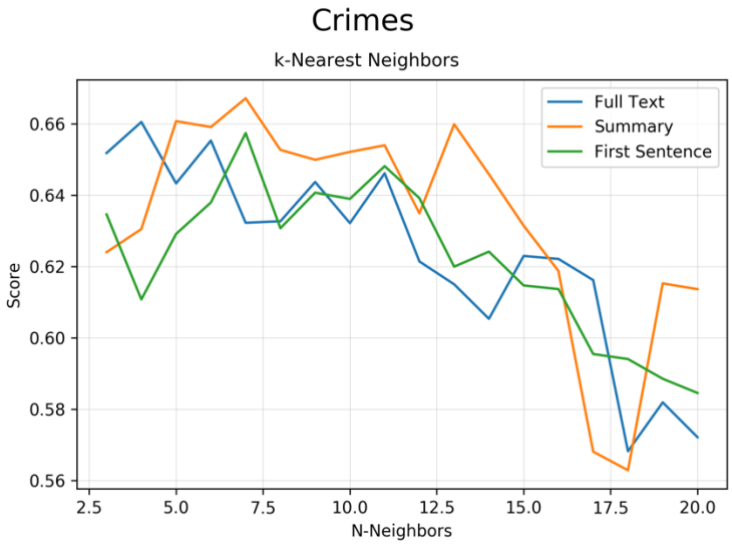


מהתבוננות בכלל הגרפים שיצרנו עבור המסווג (ובפרט בשניים משמאל) ניתן לראות שאין קשר מובהק בין מספר השכנים להצלחת המסווג. בעוד שבחלק מהקטגוריות, כגון הקטגוריה הראשית (משמאל) ניתן לראות שיפור קל ב-score עם הגדלת כמות השכנים, בקטגוריות המשנה לעומת זאת התוצאות לא היו חד משמעיות: בחלק מהמקרים (כגון ב-Health) הגדלה במספר השכנים הובילה לשיפור בתוצאות, ובמקרים אחרים (כמו ב-Crimes, משמאל למטה) התקבלה דווקא התוצאה ההפוכה.

אנו משערים שהסיבה להבדלים הנ"ל טמונה ברמת ההפרדה בין תתי-הקטגוריות של כל קטגוריה וקטגוריה.

כך לדוגמא הדמיון בין 'טכנולוגיה' ו-'פשע' ברמה הראשונה קטן יותר מהדמיון בין 'גניבה' ו-'ריגול' ברמה השנייה). לכן סביר להניח שכאשר ההפרדה בין תתי הקטגוריות גבוהה הגדלת כמות השכנים תשפר את דיוק המסווג (כיוון שסביר להניח שכמות גדולה יותר מהשכנים הקרובים ביותר לדגימה שאותה נרצה לסווג משתייכים לקטגוריה הנכונה), ואילו במקרים בהם אין הפרדה טובה בין תתי הקטגוריות במרחב הגדלה של מספר השכנים עלולה להוביל להכנסת רעש לתהליך הסיווג ולתוצאות לא נכונות.

השפעת שיטות הוקטוריזציה

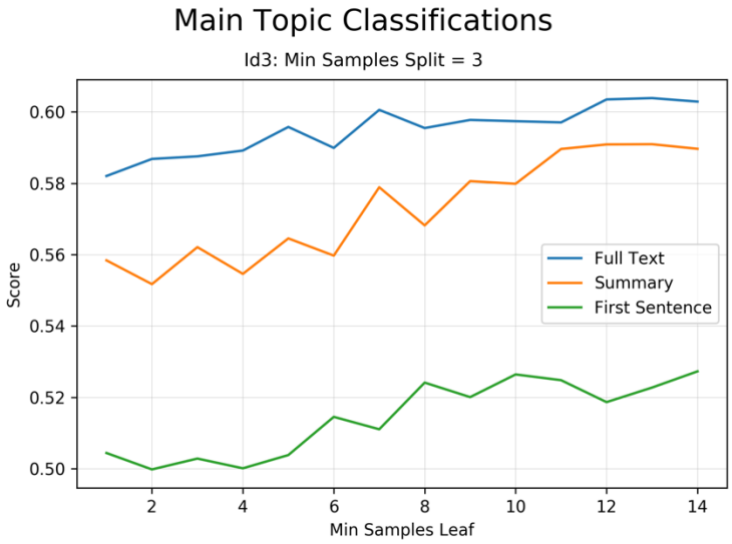


במרבית הגרפים שהתקבלו עבור המסווג ניתן לראות שאין הפרדה מובהקת בין שיטות הוקטוריזציה בציונים המתקבלים. עם זאת, במרבית המקרים הוקטוריזציה בשיטת First Sentence השיגה תוצאות טובות פחות מהשתיים האחרות (באחוזים בודדים).

אנו משערים שהסיבה להבדלים הזניחים בתוצאות עבור השיטות השונות היא שאין הבדל משמעותי בפיזור הוקטורים המתקבלים מהשיטות השונות במרחב. מכאן, כיוון שמסווג זה מסתמך על השכנים הישירים של כל דגימה בתהליך הסיווג לא מתקבל הבדל משמעותי בתוצאות.

עץ החלטה (Id3)

השפעת פרמטרי המסווג



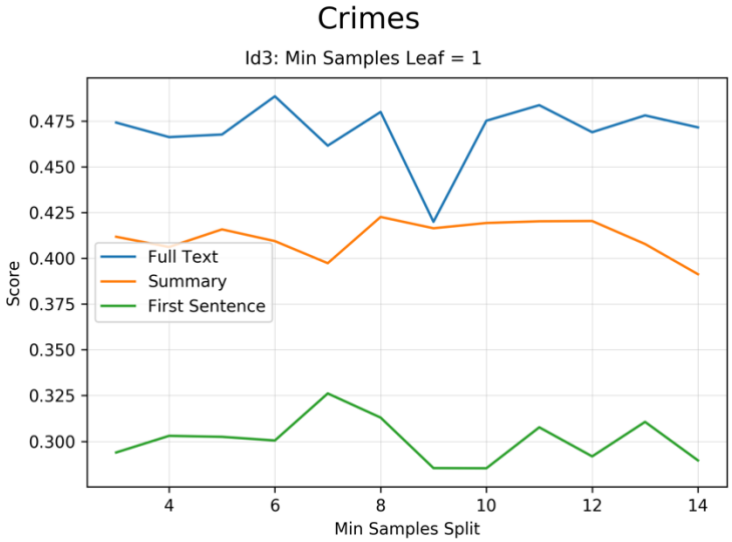
כפי שהזכרנו ברקע עבור מסווג זה הפרמטרים שבחרנו עבורו (Min Samples Leaf, Min Samples Split) משפיעים על גיזום עץ הסיווג המתקבל. זהו למעשה Embedded Feature Selection.

עם זאת נציין שלא ציפינו שגיזום העץ יביא לשיפור משמעותי בתוצאות כיוון שהנחנו מראש שסט התכונות ש-BERT מייצר (מרחב וקטורי בעל 768 מימדים) לא כולל המון תכונות מיותרות.

מבט בשני הגרפים שמשמאל אומנם תומך בהשערה ההתחלתית שלנו, אם כי בהמשך, כאשר נדון במסווג ה-AdaBoost, נגלה שבאמצעות בחירה מושכלת של כ-150 תכונות בלבד ניתן להשיג תוצאות טובות מאוד.

עם זאת, ניתן גם לראות שכן התקבל שיפור קל בגרף הקטגוריה הראשית (כ-2 אחוזים לכל שיטת וקטוריזציה) ע"י הגדלת פרמטר ה-Min Samples Leaf. אנו סבורים שהסיבה לכך היא ששימוש בכמות קטנה מדי של דגימות בעלי העץ מחמירה את -overfitting של המסווג על קבוצת האימון וכך פוגעת בתוצאותיו.

השפעת שיטות הוקטוריזציה



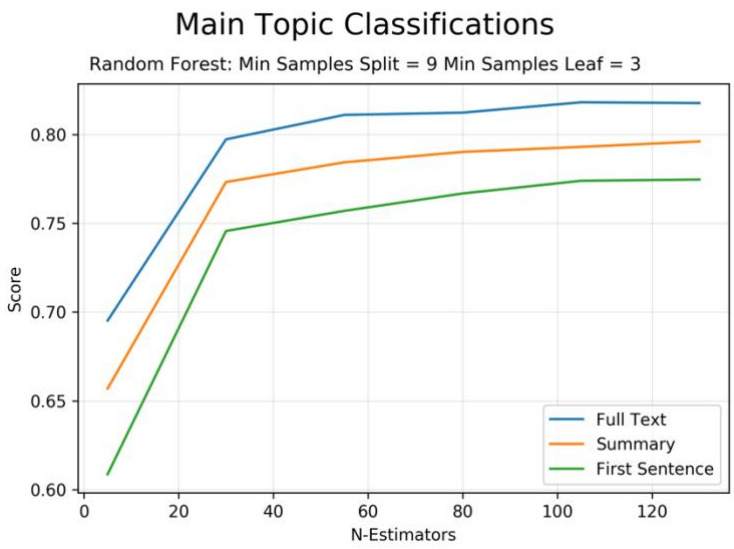
עבור עץ ההחלטה ניתן לראות ברוב המקרים ששיטות הוקטוריזציה המיודעות יותר (Full Text, Summary) השיגו את התוצאות הטובות יותר. במקרים מסוימים (כמו למשל בקטגוריה Crimes) הפער בתוצאות השיטות עבר אפילו את ה-15 אחוזים (!).

עם זאת, ניכר שהפער בתוצאות בין השיטות Full Text ו-Summary היה קטן ביחס לפער בין השיטות הללו ל-First Sentence, שהשיגה תוצאות משמעותית פחות טובות מהשתיים הנ"ל.

נזכיר כי מסווג ה- Id3 בוחר בכל שלב בתהליך בניית העץ את התכונה שהפיצול על פיה יוביל לתוספת האינפורמציה הגבוהה ביותר. במקרה זה התכונות הנ"ל הינן בעצם הממדים במרחב הוקטורי של BERT. אנו משערים שככל ששיטת הוקטוריזציה מיודעת יותר (מתבססת על מידע רב יותר במאמר), הדגימות המיוצגות על ידה מופרדות טוב יותר במרחב עבור כל מימד ומימד ולכן העץ המתקבל על ידן נותן סיווג מדויק יותר.

Random Forest

השפעת פרמטרי המסווג

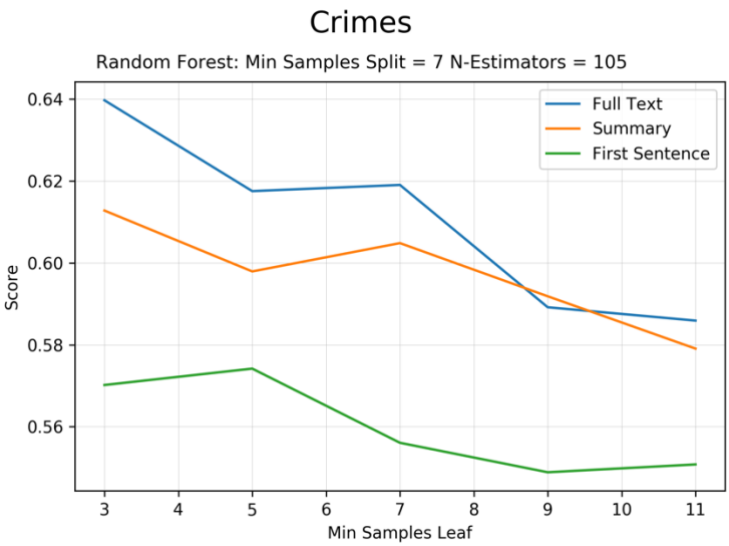


כזכור פרמטר ה-N-Estimators הוא הקובע את מס׳ העצים המשתתפים בוועדה שאותה בונה המסווג.
במרבית הגרפים המתייחסים לפרמטר זה, כמו גם בגרף משמאל, חוזרת על עצמה שוב ושוב אותה המגמה: בהתחלה, ככל שמעלים את מס׳ העצים בוועדה דיוק המסווג הולך ומשתפר. עם זאת, עבור איזשהו ערך מסוים (שנע בין 30-80 עבור מרבית הקטגוריות), מתקבלת רוויה ב-score שאותו מצליח המסווג להשיג וניכר שהוספה של עצים נוספים מעבר לערך זה אינה משפרת יותר את תוצאות המסווג.

התופעה הנ״ל הגיונית מאוד וקשורה באופן ישיר לאופן הפעולה של מסווגים המבוססים על ועדות: כיוון שכל עץ בוועדה נבנה על סמך תת קבוצה רנדומלית של דגימות מקבוצת האימון, וענפיו מפוצלים לפי תת קבוצה רנדומלית של תכונות, כל עץ בוועדה מכיל רק חלק מהתמונה הכוללת, והרעיון מאחורי האלגוריתם של המסווג הוא שהעצים השונים ישלימו את הפערים אחד של השני וכך יצליחו להשיג יחד סיווג טוב. עם זאת, הגיוני לשער שכמות קטנה מדי של עצים לא תכיל מספיק מידע בכדי להשיג תוצאות טובות, ומצד שני כמות גדולה יותר של עצים מהנדרש לא תציג עוד שיפור שכן תוספת העצים לא תוסיף עוד מידע משמעותי חדש לוועדה.

כמו כן, גם עבור מסווג זה, בדומה למסווג הקודם, לא מצאנו קורלציה ישירה בין שינוי פרמטרי הגיזום לבין תוצאות המסווג: לפרמטרים הנ״ל הייתה השפעה שונה מאוד בין קטגוריה לקטגוריה, אולם ניכר שבחלק מהקטגוריות ברמה השנייה התקבלו תוצאות טובות יותר עם ערכים נמוכים של Min Samples Leaf (3-5).
אנו משערים שהסיבה לכך היא שעבור כל תת-קטגוריה של קטגוריות אלה קיימות פחות דגימות בקבוצת האימון, ולכן שימוש בעלים ״גדולים״ מדי בעצים עבורן עלול להוביל לפגיעה ברמת הטהורות של העלים.

השפעת שיטות הוקטוריזציה



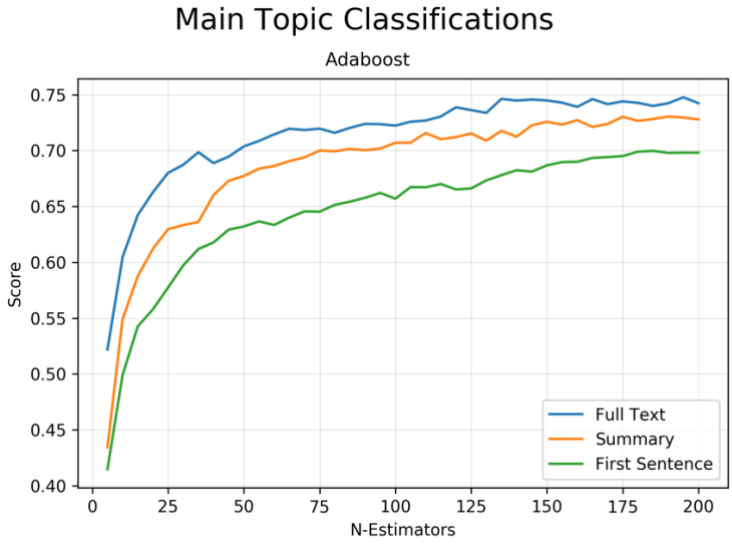
גם עבור מסווג זה ניתן לראות באופן מובהק ששיטות הוקטוריזציה המיודעות השיגו תוצאות טובות יותר, אך הפעם הפערים בין השיטות היו קטנים יותר מאלה שנראו עבור מסווג ה-Id3 ונעים סביב ה-5% בלבד.

ע״י התבוננות בגרפים של מסווג זה לעומת הגרפים של המסווג הקודם (Id3) ניתן להיווכח ביתרון של שימוש בוועדה של עצים על פני שימוש בעץ יחיד. כזכור, ההבדל בין מסווג ה-Id3 לבין המסווג הנוכחי הוא שבעוד שהראשון בונה עץ יחיד באמצעות כל הדגימות, השני בונה ועדה של עצים שכל אחד מבוסס על סט רנדומלי של דגימות ותכונות.

ואכן, ניתן לראות שעבור כל שיטת וקטוריזציה הושג בממוצע שיפור של 20-30% ב-score שהושג באמצעות שימוש במסווג ה-Random Forest ביחס לציון שהושג באמצעות מסווג ה-Id3. אנו סבורים שהסיבה לפערים הקטנים יותר בין השיטות שהתקבלו במסווג הנוכחי טמונה בכך שוועדת העצים הצליחה להפיק יותר מהמידע ומהפוטנציאל הטמון בוקטורים של כל שיטה ושיטה.

AdaBoost

השפעת פרמטר המסווג

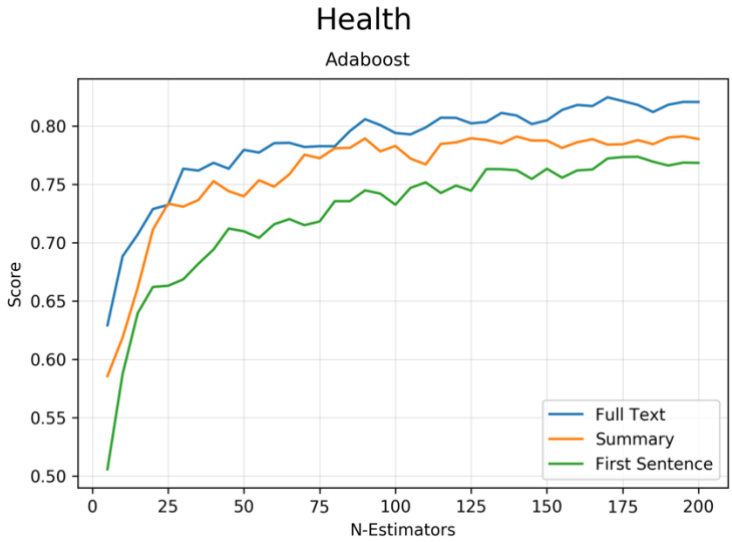


גם עבור מסווג זה, בדומה למסווג ה- Random Forest, חוזרת על עצמה מגמת הרוויה במרבית הגרפים המתייחסים לפרמטר ה-N-Estimators ככל שמעלים את מספר העצים בוועדה.

כזכור, הוועדה שבונה המסווג מורכבת ממספר רב של מסווגים חלשים (ועדה של עצים בגובה 1 המכונים 'stumps'), שכל אחד מהם מבוסס על תכונה אחת בלבד ולכן בעצם מהווה מסווג בינארי.

מכאן נובע באופן ישיר שמספר המסווגים המרכיבים את הוועדה הנ"ל זהה למספר התכונות הכולל שבהן המסווג משתמש. לפיכך, העובדה שמסווג זה מגיע לרוויה בתוצאות הסיווג באמצעות כ-150 מסווגים בינאריים בלבד מעידה על כך שניתן לבצע סיווג עבור מרבית הקטגוריות בצורה טובה למדי באמצעות כ-150 תכונות בלבד (כ-20%!) מתוך 768 התכונות המרכיבות את הוקטורים שפולט BERT.

השפעת שיטות הוקטוריזציה

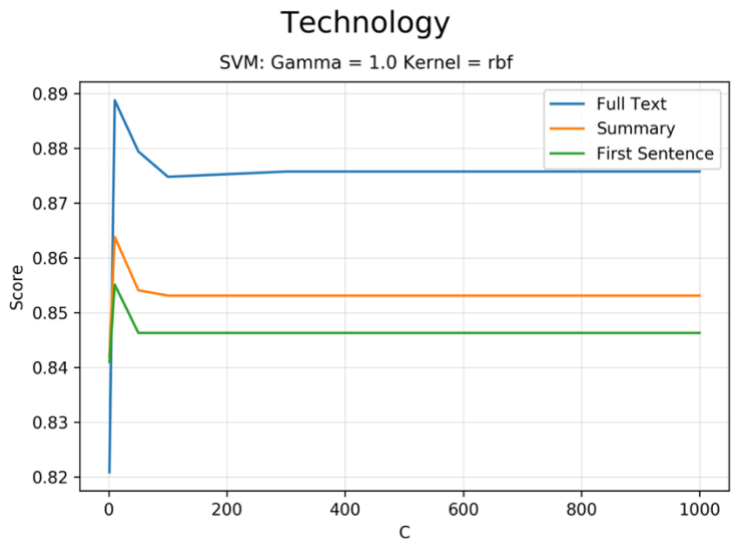


גם במקרה זה ניכר שבמרבית הקטגוריות (כולל בקטגוריות בגרפים המצורפים משמאל) קיימת קורלציה טובה בין רמת המיודעות של שיטת הוקטוריזציה לבין התוצאה שהושגה, כאשר בראש ובראשונה עומדת שיטת ה-Full Text.

עם זאת, ההפרשים בין התוצאות הטובות ביותר שהשיגו השיטות השונות השתנו מאוד מקטגוריה לקטגוריה: כ-12% עבור Crimes לעומת כ-2% בלבד עבור Entertainment (ההפרשים הם בין התוצאות הטובות ביותר של השיטות החזקה והחלשה ביותר בכל קטגוריה). תופעה זו ככה"נ נובעת מכך שבקטגוריות מסוימות משפטי הפתיחה (שעליהם מבוססת שיטת ה-First Sentence שהינה לרוב החלשה ביותר) ממצים טוב יותר את נושאי המאמרים מאשר משפטי הפתיחה בקטגוריות אחרות.

SVM (Support Vector Machine)

השפעת פרמטרי המסווג



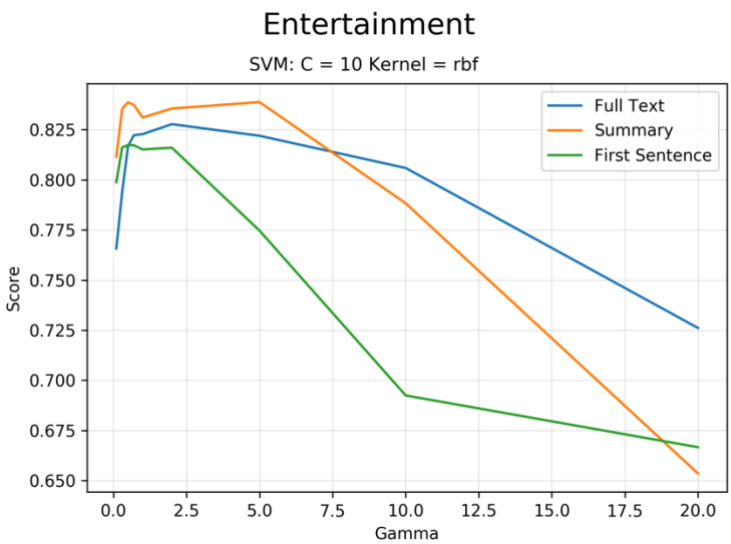
כפי שניתן לראות בגרפים משמאל, מסווג זה, שהשיג תוצאות טובות למדי כמעט בכל הקטגוריות, הצטיין בעיקר באמצעות שימוש ב-kernel ה-rbf תוך שימוש בערכים קטנים יחסית של C ו-gamma.

עם זאת, השיפור בתוצאות שהשיג המסווג באמצעות שימוש בקרנל ה-rbf (שאיננו לינארי) היה די זניח ביחס לתוצאות אותן השיג באמצעות הקרנל הליניארי (שיפור של כ-1-2% בממוצע בכל קטגוריה), מה שמעיד על כך שמרבית הקטגוריות (גם ברמה השנייה) ניתנות להפרדה ליניארית די טובה במרחב.

עובדה זו הינה משמעותית כיוון שזמן הריצה של המסווג תוך שימוש בקרנל הליניארי היה נמוך מבזה הלא לינארי, מה שמעיד על כך שניתן לקבל תוצאות סיווג טובות למדי ללא צורך בשימוש בקרנלים מסובכים.

הסיבה שהמסווג השיג תוצאות טובות יותר באמצעות ערכים קטנים יותר של C ו-gamma נובעת מהעובדה שכפי שהזכרנו קודם לכן ברקע על המסווג, ככל שערכם של פרמטרים אלה גדול יותר כך המסווג נוטה יותר לביצוע overfitting על קבוצת האימון, מה שבסופו של דבר מביא להשגת תוצאות טובות פחות במציאות.

השפעת שיטות הוקטוריזציה

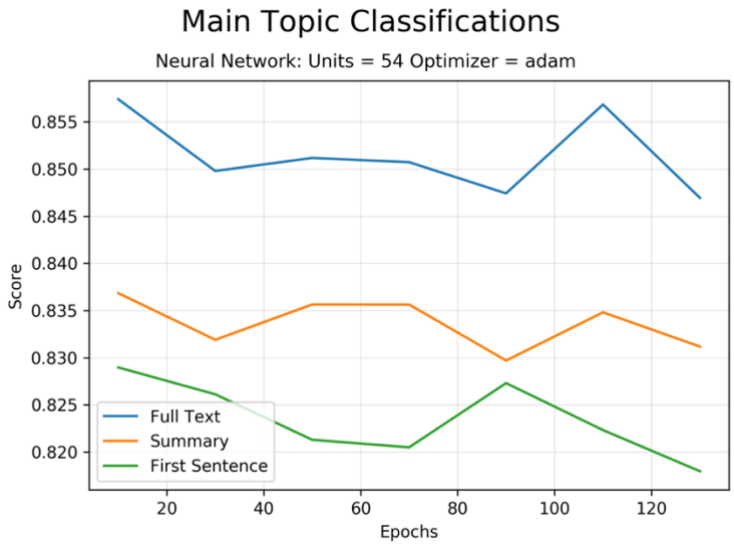


על אף שמסווג זה השיג תוצאות יוצאות מן הכלל במרבית הקטגוריות באמצעות כל שיטות הוקטוריזציה, ניכר שגם הפעם השיטות המיודעות יותר השיגו תוצאות מרשימות יותר.

גם כאן ברוב המקרים השיגה שיטת ה-Full Text את התוצאות המרשימות ביותר, אולם מעניין היה גם לגלות שבקטגוריה Entertainment השיטה שהשיגה את ה-score הגבוה ביותר הייתה דווקא שיטת ה-summary, כפי שניתן לראות בגרף משמאל. עם זאת, כפי שנראה בהמשך הצלחתה של שיטה זו (תוצאת recall גבוהה) במקרה זה לא באה לידי ביטוי גם בתוצאת הדיוק (accuracy) של המסווג בקטגוריה זו, שכן את תוצאת הדיוק הגבוהה ביותר השיגה שיטת ה-Full Text.

Neural Network

השפעת פרמטרי המסווג

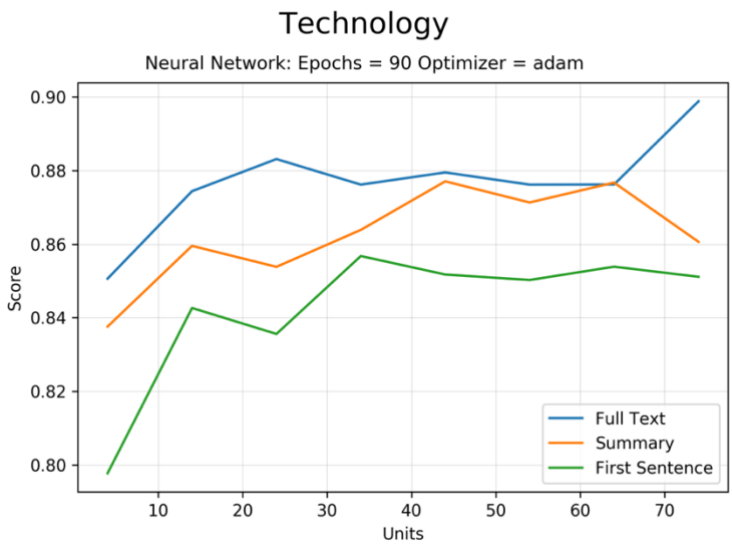


גם מסווג זה, בדומה למסווג ה-SVM, הצליח להשיג תוצאות מרשימות למדי בכל שיטות הוקטוריזציה וברוב הקטגוריות.

מבחינת פרמטר ה-Epochs (שכזכור מייצג את מספר המעברים שמתבצעים על קבוצת האימון בתהליך הלמידה של המסווג), ניכר שעבור קטגוריות שונות נדרשו ערכים שונים מאוד של הפרמטר על מנת להביא את המסווג לנקודת "רוויה" בתוצאות שהתקבלו, שלאחריה הוספת Epochs נוספים לעיתים דווקא פגעה בתוצאות, ככה"נ כתוצאה מביצוע overfitting על קבוצת האימון. כך לדוגמא בכדי לקבל את התוצאה הטובה ביותר בקטגוריה הראשית (בגרף משמאל) נדרשו 10 epochs בלבד (!), בעוד שבכדי לקבל את התוצאה המיטבית בקטגוריה Technology נדרשו 90 epochs.

עבור פרמטר ה-Units (בגרף משמאל למטה לדוגמא), הגרפים שהתקבלו עבור הקטגוריות השונות הזכירו מאוד במגמתם את הגרפים שהתקבלו עבור פרמטר ה-N-Estimators של מסווגי הוועדות שהצגנו (Random Forest, AdaBoost). גם במקרה זה, הגדלה בכמות ה-units הובילה תחילה לעלייה בתוצאות המסווג, אולם החל מערך מסוים המסווג הגיע לרוויה והתוצאות לא השתפרו עוד. תופעה זו הגיונית שכן כפי שהזכרנו ברקע על רשתות הנוירונים (עמ' 50) פרמטר ה-units קובע את מספר הנוירונים בכל שכבה חבויה, וציינו שיתכן שרשתות קטנות מדי יתקשו לקרב את המיפוי הנדרש בצורה מספקת. עם זאת, החל מכמות מסוימת ניכר גם שתוספת של נוירונים נוספים לשכבות כבר לא מובילה יותר לשיפור במיפוי המתקבל.

עבור פרמטר ה-optimizer נראה שבמרבית הקטגוריות שימוש באלגוריתם 'adam', שמבוסס על אלגוריתם ה-Stochastic Gradiant Decent, הביא לתוצאות טובות יותר מאשר שימוש באלגוריתם ה-sgd הרגיל. יתכן שהסיבה לכך היא שאלגוריתם זה משתמש בטכניקות שונות על מנת לשפר את זמן ההתכנסות של אלגוריתם ה-sgd הרגיל, ולכן השגנו באמצעותו תוצאות טובות יותר בכמות epochs זהה.



השפעת שיטות הוקטוריזציה

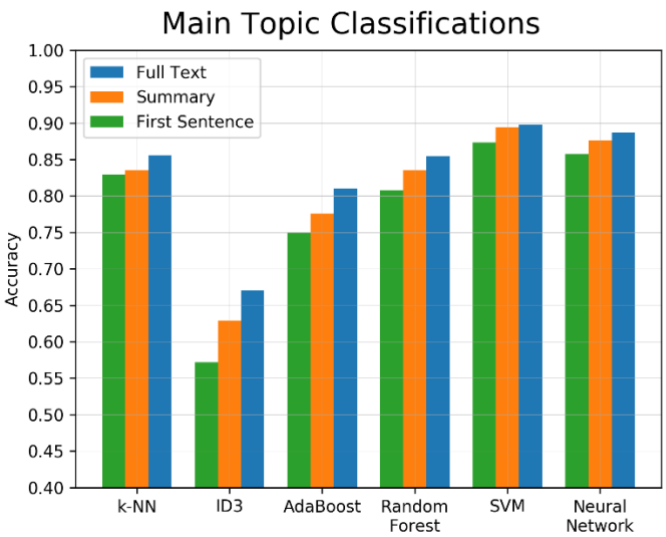
מסווג זה לא הציג הבדלים בביצועים של שיטות הוקטוריזציה השונות ביחס למסווגים הקודמים, כשגם כאן נראתה קורלציה ברורה בין מידת המיודעות של השיטה לבין רמת תוצאותיה.

השוואת ביצועי המסווגים והסקת מסקנות

כזכור, בתום שלב הכיוונון קיבלנו מאלגוריתם בחירת הפרמטרים מילון המשמש כפונקציה שבהינתן קטגוריה, מסווג, ושיטת וקטוריזציה מחזירה את סט הפרמטרים הטוב ביותר של המסווג עבור הקטגוריה ושיטת הוקטוריזציה, ואת תוצאת ה-accuracy שהשיג המסווג עם פרמטרים אלה על קבוצת האימות.
באמצעות המילון הנ"ל יצרנו היסטוגרמות המשוות לכל קטגוריה (מבין הקטגוריה הראשית וחמשת קטגוריות המשנה) את ביצועי המסווגים השונים על הקטגוריה באמצעות שיטות הוקטוריזציה השונות.
אנו מזכירים שוב שהציונים שנשמרו עבור המסווגים במילון הנ"ל (ועליהם מבוססים צירי ה-y של ההיסטוגרמות משמאל) הינם ציוני **accuracy**, כיוון שבשלב זה נרצה להשוות בין הדיוקים האבסולוטיים של המסווגים על כל קטגוריה וקטגוריה. זה גם מסביר את הסיבה לכך שציוני ההיסטוגרמות נראים גבוהים יותר מהציונים שהתקבלו בגרפים בתת הפרק הקודם (כזכור עבור כיוונון הפרמטרים השתמשנו בפונקציית score אשר מסתכלת על ממוצע שלושת הקטגוריות בעלות ה-recall הנמוך ביותר).

בתת-פרק זה נדון בהשפעותיהם השונות של שיטות הוקטוריזציה, המסווגים והקטגוריות השונות על התוצאות שהתקבלו.

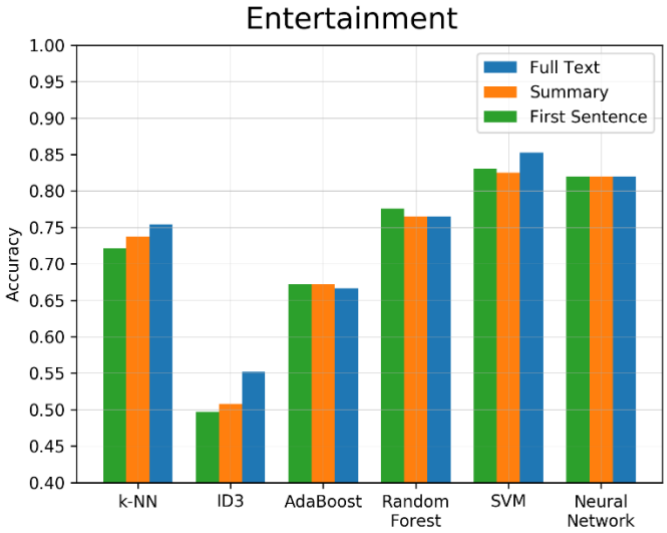
השפעת שיטת הוקטוריזציה

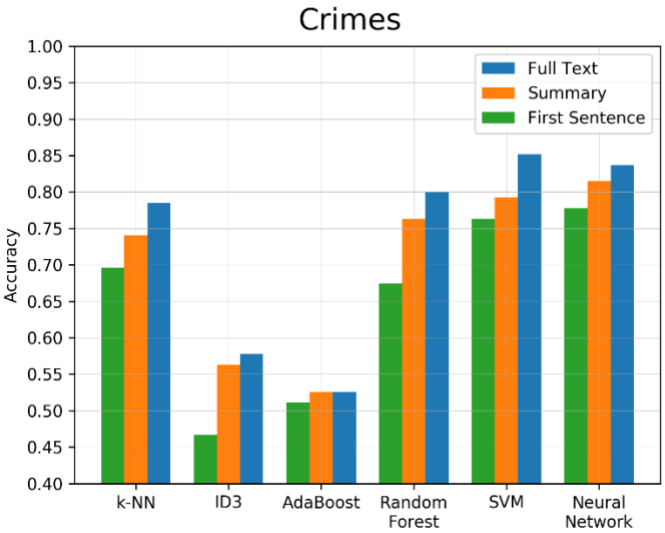


בדומה לגרפים שראינו עבור כיוונוני הפרמטרים, במקרה זה, כפי שניתן לראות בהיסטוגרמות, ככל ששיטת הוקטוריזציה הייתה **מיודעת** יותר (הסתמכה על כמות גדולה יותר של טקסט במאמר) היא הצליחה להשיג תוצאות טובות יותר ברוב המוחלט של המקרים, כל זה ללא תלות במסווג וללא תלות במיקום ההיררכי של הקטגוריה.
עם זאת, על אף העובדה ששיטת ה-Full Text השיגה את התוצאות הטובות ביותר ברוב המוחלט של המקרים, לרוב ההבדלים בין השיטות היו זניחים יחסית, והסתכמו בכ-3-5% בלבד. תופעה זו באה לידי ביטוי בעיקר בקרב המסווגים החזקים יותר (SVM ורשת הנוירונים) שהציגו תוצאות דומות מאוד עבור כל שיטות הוקטוריזציה.
המסקנה הנ"ל משמעותית כיוון שהיא מוכיחה אמפירית שבכדי לקבל תוצאות טובות מאוד בקלסיפיקציה של מאמרים מספיק להשתמש במשפט הראשון בלבד של כל מאמר – כלומר בשיטת ה-First Sentence.
ניכר ששיטה זו מציעה את האיזון הטוב ביותר בין דיוק וזמן ריצה (כזכור משפט פתיחה ממוצע במאגר הנתונים שלנו מכיל כ-1.8% בלבד מכלל המילים במאמר כולו, כלומר מצריך זמן וקטוריזציה נמוך פי 55 (!) ביחס לוקטוריזציה של המאמר כולו).

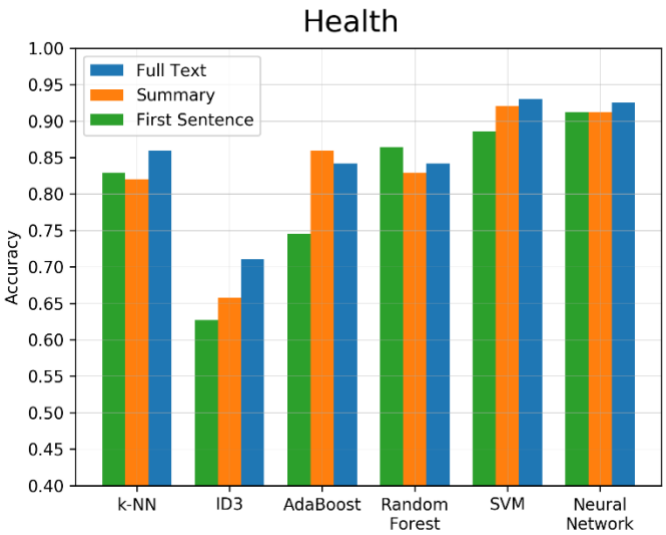
השפעת המסווג

ראשית, אנו סבורים שהצלחתו הטובה יחסית של מסווג ה-KNN (שהינו גם הפשוט ביותר מבחינה רעיונית), מוכיחה אמפירית בצורה טובה למדי שהרעיון ליצור וקטורים עבור מאמרים שלמים באמצעות BERT, תחת ההנחה שמאמרים הקשורים זה לזה מבחינה רעיונית יהיו קרובים זה לזה גם במרחב ה-n מימדי, אכן עובד.





עם זאת, מסווג ה-KNN כלל לא היה קרוב להשיג את התוצאות הטובות ביותר, מה שמעיד על כך ששימוש בשכנים הקרובים ביותר של הדגימה שאותה נרצה לסווג תוך הסתמכות רק על המרחקים בין הדגימות לצורך בחינת קרבתן מבחינה סמנטית איננה בהכרח השיטה האידיאלית להפרדה בין הקטגוריות במקרה זה. מנגד, ניתן ללמוד על רמת המורכבות הרבה של ההפרדה במרחב בין הקטגוריות השונות מהעובדה שהמסווגים כשהתגלו כמצליחים ביותר היו רשת הנוירונים ומסווג ה-SVM. מסווגים אלה ידועים ביכולתם להשתמש ב"מידע החבוי" המצוי בוקטורים (כלומר להבחין בקשרים הלאו דווקא ליניאריים הקיימים בין הדגימות השונות) בכדי לשפר את תוצאות הסיווג, והעובדה ששני המסווגים הנ"ל השיגו תוצאות טובות יותר משמעותית משאר המסווגים אכן מרמזת על הפרדה אי-ליניארית של הדגימות במרחב.

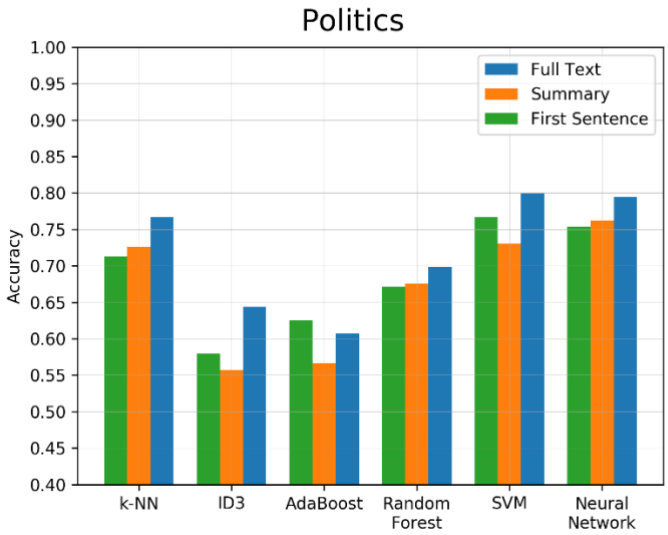


דבר נוסף שעולה מההיסטוגרמות הוא שכצפוי המסווגים המורכבים מוועדות של עצים (Random Forest, AdaBoost) השיגו תוצאות טובות יותר משמעותית מתוצאתו של מסווג ה-ID3 (שכאמור מבוסס על עץ בודד). תוצאה זו איששה את ציפיותינו שאלגוריתם המשיג ריבוי נקודות מבט על הדאטה באמצעות ועדה של מסווגים עולה בדיוקו על אלגוריתם המבוסס על מסווג יחיד.

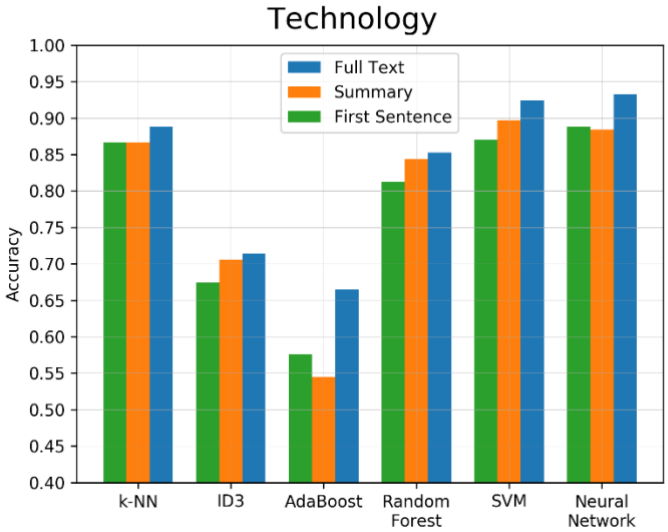
השפעת הקטגוריה

מההיסטוגרמות עולה שבקטגוריות שונות הצלחת המסווגים הייתה שונה, לעיתים אף באחוזים ניכרים.

ניקח לדוגמא את הקטגוריה Politics (משמאל):
אפילו מסווג ה-SVM - שקיבל את התוצאה הגבוהה ביותר עבור הקטגוריה, הצליח להשיג בה רק כ-80% דיוק.
לעומת זאת, בקטגוריה Technology (משמאל למטה) ניתן לראות שישנם ארבעה מסווגים אשר ללא תלות בשיטות הוקטוריזציה השיגו מעל 80% דיוק, כאשר הטוב מבינהם, רשת הנוירונים, אף הגיע לכ-94%.



הפערים באחוזי ההצלחה בין הקטגוריות השונות תלויים כמובן ברמת ההפרדה בין תתי-הקטגוריות של כל קטגוריה וקטגוריה.
לדוגמא, בכדי להסביר את ההפרש שהתקבל בדוגמא לעיל בין הקטגוריה Technology ל-Politics נתבונן רגע בתתי הקטגוריות של Technology:



- Mobile
- Engineering
- Computers
- Science and Technology Ministries

ברור מיד כי רמת ההפרדה בין כל תתי הקטגוריות הנ"ל הינה גבוהה יחסית.
לעומת זאת, תתי הקטגוריות של Politics הן:

- Voting
- Government
- Jewish Political Status
- Political Corruption

במקרה זה ברור כי רמת החפיפה בין שתיים מתתי הקטגוריות: Government ו-Voting עלולה להיות גבוהה יחסית, לדוגמא: מאמר המשתייך ל-Government אשר מדבר על הצבעה בכנסת ניתן גם לסווג "בטעות" לקטגוריה Voting.

חשוב אומנם גם לציין שבמציאות יתכן מאוד שמאמר מהסוג הנ״ל אכן היה מסווג בויקיפדיה תחת שתי תתי הקטגוריות המוזכרות לעיל, אך כאמור כיוון שמאגר המידע שלנו ממודל כעץ בעל מסלול יחיד לכל מאמר בהיררכיה רק אחד מבין הסיווגים האמיתיים יהיה קיים במאגר המידע שלנו. לכן יתכן מאוד שהיו מצבים שבהם על אף שמאמרים מסוימים בקבוצת המבחן שלנו סווגו על ידי מסווגים מסוימים נכון מבחינת היררכיית הקטגוריות האמיתית בויקיפדיה, הסיווגים הנ״ל סומנו כשגויים ע״י האלגוריתם שלנו כיוון שמאגר המידע שלנו כולל לכל היותר סיווג אחד לכל מאמר בכל רמה.

בניית המסווג ההיררכי והערכתו

בפרק זה נתאר את תהליך החשיבה שעברנו בבניית המסווג הסופי, את אופן בנייתו ואימונו ולבסוף אף נציג את התוצאות שהשיג בסיווג קבוצת המבחן.

כזכור, בהינתן מאמר חדש מטרתו של מסווג זה הינה לחזות את המסלול בהיררכיית הקטגוריות בו יש לסווג את המאמר.

כשהתחלנו לדון בינינו כיצד יש לבנות מסווג מהסוג הנ"ל עלו אצלנו המון שאלות שנוגעות לאופי המסווג: על איזה מהמסווגים שבחנו בפרק הקודם כדאי להתבסס? עם איזו שיטת וקטוריזציה כדאי לעבוד? וכיצד יעבוד אלגוריתם הסיווג במעבר בתוך היררכיית הקטגוריות?

לאחר דיונים ארוכים בנושא הגענו למסקנה שלא יהיה חכם מצדנו להתבסס בבניית המסווג רק על מסווג בסיסי טוב אחד (לדוגמא: SVM) או רק על שיטת וקטוריזציה אחת, שכן כפי שראינו בפרק הקודם עבור קטגוריות שונות במקרים מסוימים עדיפים מסווגים שונים עם קונפיגורציות שונות ושיטות וקטוריזציה שונות, ויתכנו גם כמובן מקרים שבהם מסווג טוב אחד ייתן סיווג נכון בעוד שמסווג טוב אחר ייתן סיווג לא נכון ולהיפך.

רצינו אם כך למצוא דרך שבה נוכל להשתמש ב"חוכמתם" של כל המסווגים הטובים ביותר שמצאנו עבור כל קטגוריה, כך שאלה ישלימו ויחפו על הפערים זה של זה.

כלומר – עבור כל קטגוריה החלטנו לבנות <u>ועדה</u> של המסווגים הטובים ביותר שמצאנו עבורה בפרק הקודם.

אימון המסווג

כזכור, אלגוריתם בחירת הפרמטרים (שהוצג בפרק הקודם) סיפק לנו מילון המהווה פונקציה שבהינתן קטגוריה, מסווג, ושיטת וקטוריזציה מחזירה את סט הפרמטרים הטוב ביותר של המסווג עבור הקטגוריה ושיטת הוקטוריזציה, ואת תוצאת ה-accuracy שהשיג המסווג עם פרמטרים אלה על קבוצת האימות.

באמצעות מילון זה מיינו לכל קטגוריה את <u>קומבינציות</u> המסווגים ושיטות הוקטוריזציה על פי התוצאות שהשיגו עבורה. מתוך הרשימה הממוינת הנ"ל בחרנו את 3 הקומבינציות הטובות ביותר עבור הועדה של אותה הקטגוריה.

חשוב אולם להדגיש כי בשיטה זו יתכנו חברים בוועדה המבוססים על אותו מסווג (אך עם שיטת וקטוריזציה שונה). בחרנו כך כי אנו מאמינים ששיטת וקטוריזציה נוספת עלולה להוסיף דיוק גם כאשר מדובר באותו מסווג.

לאחר הרכבת הוועדה לכל קטגוריה אימנו מראש את כל המסווגים בוועדות השונות על קבוצות הלימוד והאימות (כאשר בהמשך המסווג ייבחן לבסוף על קבוצת המבחן). בתום תהליך האימון שמרנו את מסווגי הוועדות השונות במילון שאליו ניתן לגשת לפי ה-ID של הקטגוריה המתאימה. מילון זה שימש אותנו בהמשך באלגוריתם הסיווג כך שלא היינו צריכים לאמן את המסווגים שוב מחדש עבור כל דוגמא בקבוצת המבחן.

בניית אלגוריתם הסיווג

הרעיון המקורי

טרם תחילת העבודה על הפרויקט הרעיון הראשון שעלה במוחנו עבור אלגוריתם הסיווג ההיררכי היה הרעיון הבא:

בהינתן מאמר חדש, נעבור רמה-רמה בהיררכיית הקטגוריות ובכל רמה ננסה לסווג את המאמר לכל אחת מן הקטגוריות הנמצאות באותה רמה.

לרעיון הנ"ל יש חיסרון ברור: בהינתן שברמה אחת סיווגנו את המאמר לקטגוריה X, לא מובטח לנו שברמה הבאה המאמר יסווג לתת קטגוריה של X, שכן בכל איטרציה האלגוריתם מנסה לסווג את המאמר לכל הקטגוריות הנמצאות באותה רמה - ללא תלות בקטגוריה שנבחרה ברמה הקודמת.

החיסרון הנ"ל עלול כמובן להוביל לחיזוי של מסלולים שכלל לא קיימים בהיררכיית הקטגוריות, מה שכמובן יפגע מאוד בדיוק המסווג.

חסרון נוסף הוא כמות הקטגוריות שאיתן יצטרך המסווג להתמודד בו זמנית בכל רמה: על אף שברמה הראשונה יצטרך המסווג להתמודד עם 5 קטגוריות בלבד, ברמה השנייה יגדל המספר כבר ל-23 קטגוריות, וברמה השלישית יצטרך המסווג להתמודד עם 52 (!) קטגוריות בו זמנית. אין ספק שהגידול הדרסטי בכמות הקטגוריות מרמה לרמה יפחית מאוד את דיוק המסווג בעומק ההיררכיה.

הפתרון המשופר

בכדי להתמודד עם הבעיות הנ"ל החלטנו לשפר את הרעיון המקורי ע"י הוספת תלות בין הקטגוריות שאליהן ננסה לסווג ברמה Z לבין הקטגוריה שאליה סיווגנו את המאמר ברמה Z-1. כלומר: בהינתן שאלגוריתם הסיווג סיווג את המאמר לקטגוריה Y ברמה Z, האלגוריתם יוכל לסווג את המאמר ברמה Z+1 רק לאחת מתתי הקטגוריות של Y.

כך נוכל להבטיח שכל מסלול הנחזה ע"י אלגוריתם הסיווג קיים בהכרח בהיררכיית הקטגוריות.

החיסרון בשיטה זו אומנם הוא שבהינתן שהאלגוריתם טעה בסיווג המאמר ברמה Z, החל מרמה Z ואילך שארית המסלול שיוחזר ע"י האלגוריתם יהיה שגוי בוודאות שכן גרף הקטגוריות שלנו הינו עץ - מה שמבטיח קיום של מסלול יחיד מהשורש אל כל קטגוריה (ובפרט אל כל קטגוריה במסלול הנכון).

בשיטה הקודמת לעומת, על אף שהמסווג טעה ברמה Z יתכן ששארית המסלול שהיה מחזיר הייתה נכונה.

עם זאת, אנו סבורים שכל מסלול חיזוי "טוב" חייב קודם כל להיות מסלול אמיתי בהיררכיית הקטגוריות, ולכן בעינינו הפתרון המשופר טוב יותר. בנוסף, העובדה שבפתרון החדש בכל רמה בהיררכיה יצטרך המסווג להתמודד בו זמנית עם 5 קטגוריות לכל היותר (מס' תתי הקטגוריות המקסימלי במאגר שלנו) מהווה גם היא יתרון משמעותי של הפתרון החדש על פני הישן.

סוגיה נוספת שעלתה לדיון בהקשר של אלגוריתם הסיווג הייתה השאלה: כיצד ידע האלגוריתם לעצור את תהליך הסיווג עבור מאמרים שהינם בנים ישירים של קטגוריות שאינן בתחתית ההיררכיה?

לדוגמא: בהינתן שקיים מאמר כללי כלשהו שהינו בן של קטגוריה ברמה 2, לא נרצה שאלגוריתם הסיווג יחזיר עבורו מסלול באורך הגדול מ-2.

לאחר דיון בנושא הגענו למסקנה שהסוגיה הנ"ל דורשת מחקר עצמאי נפרד, ולכן החלטנו שאלגוריתם הסיווג ההיררכי שנבנה במקרה זה יצפה לקבל מראש גם את מס' הרמות שיש לסווג כל דוגמא ודוגמא בקבוצת המבחן.

האלגוריתם הסופי

נציג כעת תיאור מילולי של אלגוריתם הסיווג ההיררכי הסופי שיצרנו:

בהינתן מאמר חדש, 3 הוקטורים המייצגים אותו (אחד עבור כל שיטת וקטוריזציה) ומספר הרמות שבהן יש לסווג את המאמר, בצע החל משורש היררכיית הקטגוריות את הצעדים הבאים:

1. אם הגעת למס' הרמות המקסימלי שיש לסווג את המאמר או שלקטגוריה הנוכחית אין תתי-קטגוריות, החזר את המסלול שנמצא עד כה וסיים.
2. הרץ את כל אחד ממסווגי הועדה של הקטגוריה הנוכחית על וקטור המאמר המתאים לשיטת הסיווג שעליה אומן המסווג ושמור את קולות המסווגים.
3. בצע הצבעה באמצעות קולות המסווגים ובחר את תת הקטגוריה שקיבלה את מס' הקולות הגבוה ביותר. במקרה של תיקו, הקול של המסווג החזק ביותר בוועדה (זה שקיבל את ציון ה-accuracy הגבוה ביותר על קבוצת האימות) הוא הקובע.
4. הפוך את תת הקטגוריה שנמצאה לקטגוריה הנוכחית, הוסף אותה למסלול וחזור לשלב 1.

- את מימוש המסווג הסופי בשלמותו הכולל את אלגוריתם הסיווג ההיררכי ניתן למצוא בקובץ: HierarchicalArticleClassifier.py. בנוסף, בפועל בכדי לייעל את האלגוריתם הנ"ל הגרסה שמימשנו בקובץ זה מעט שונה מהמתואר לעיל: בגרסה שמימשנו במציאות האלגוריתם בודק בכל ברמה בה הוא עובר האם הסיווג האחרון שביצע נכון, ואם לא, הוא עוצר מיד.

הערכת המסווג

בכדי להעריך את איכות המסווג שבנינו בחנו את המסווג על הדוגמאות בקבוצת המבחן ששמרנו בשלב הלמידה, שכאמור מהווה כ-15% מכלל המאמרים שכרינו עבור מאגר הנתונים שלנו.

בנוסף, דנו רבות בסוגיה כיצד יש להעריך את הביצועים שאותם יציג המסווג. שאלות רבות עלו בהקשר זה: מה נחשב מסלול חיזוי טוב? האם למאמרים בעומקים שונים צריכה להיות אותה ההשפעה על הציון הסופי שניתן למסווג? האם לדוגמא, למאמר בעומק 1 שמסלול החיזוי שלו היה מדויק צריכה להיות אותה השפעה על הציון הסופי שניתן למסווג כמו מאמר בעומק 3 שמסלול החיזוי שלו היה נכון רק ברמה הראשונה? (הרי בשני המקרים מספר הרמות שבהן נעשה סיווג נכון הייתה זהה).

ככל שדנו יותר בשאלות הנ"ל הגענו למסקנה שפרמטר יחיד (כגון ציון accuracy או recall) אינו יוכל לשקף את התמונה בשלמותה עבור מסווג בכמה רמות.

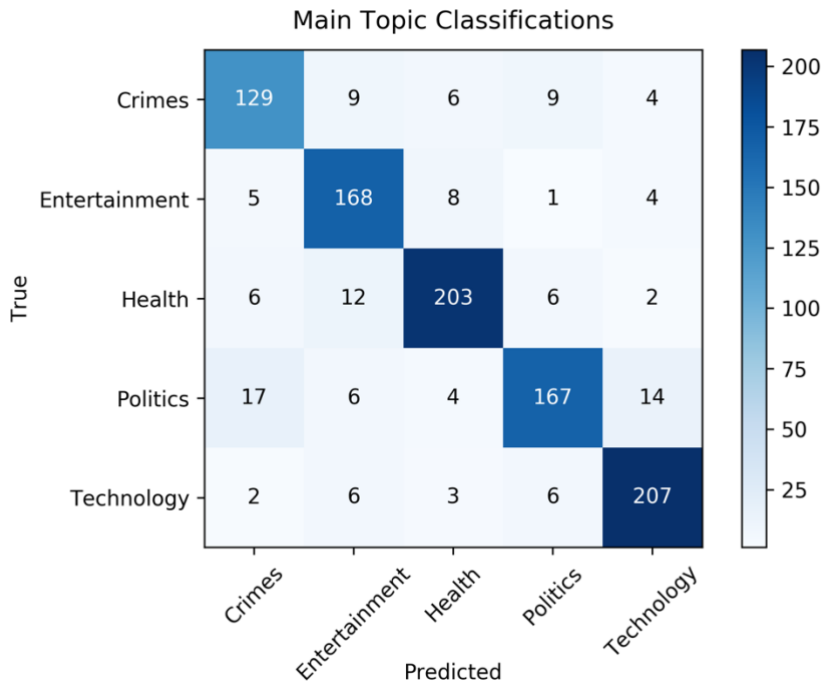
לכן, החלטנו להציג מס' סטטיסטיקות שונות של ביצועי המסווג בפני הקורא:

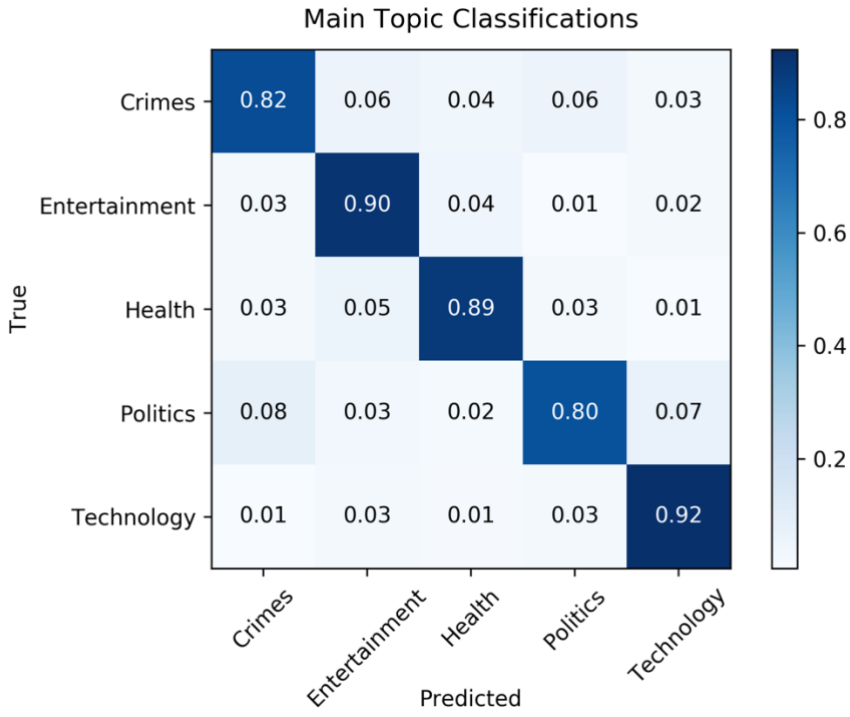
פרמטר	**אחוז דיוק**
היחס הממוצע בין אורך המסלול הנחזה לבין אורך המסלול האמיתי עבור מאמרים שאורך מסלולם האמיתי הוא 1	84.6
היחס הממוצע בין אורך המסלול הנחזה לבין אורך המסלול האמיתי עבור מאמרים שאורך מסלולם האמיתי הוא 2	78.5
היחס הממוצע בין אורך המסלול הנחזה לבין אורך המסלול האמיתי עבור מאמרים שאורך מסלולם האמיתי הוא 3	78.7
היחס הממוצע בין אורך המסלול הנחזה לבין אורך המסלול האמיתי עבור כלל המאמרים	78.8
ממוצע הסיווגים הנכונים ברמה 0	87.1
ממוצע הסיווגים הנכונים ברמה 1	84.4
ממוצע הסיווגים הנכונים ברמה 2	89.4

כפי שניתן לראות בטבלה לעיל המסווג הגיע לאחוזי דיוק גבוהים יחסים (כ-85%) בסיווג מסלולים של מאמרים הנמצאים בעומק 1, ובאורח מפתיע התוצאה הנ"ל לא השתנתה משמעותית במעבר מסיווג ברמה אחת לסיווג ב-2 או ב-3 רמות.

מבחינת התוצאות לפי רמות המסווג הגיע לתוצאות טובות מאוד ברמות 0 ו-1, ולתוצאה מצוינת (כ-89%) ברמה 2. עם זאת התוצאה ברמה 2 לא מפתיעה שכן ברמה זו לרוב לכל קטגוריה יש כ-2-3 תתי קטגוריות בלבד מה שיכול להקל על הסיווג. צריך גם להזכיר שהממוצעים המתייחסים לרמות 1-2 מתייחסים רק לדגימות שעבורן אכן בוצע סיווג ברמה זו, שכן במידה והאלגוריתם מזהה שהוא טעה ברמה 0 הוא אינו ממשיך לנסות לסווג את הדגימה ברמות הבאות.

כמו כן החלטנו גם להציג מטריצות בלבול המתארות את ביצועי המסווג בסיווג ברמה הראשונה לקטגוריות המרכזיות:





במטריצות לעיל ניתן לראות שהקטגוריות שבהן המסווג הצטיין במיוחד היו 'טכנולוגיה', 'בריאות' ו-'בידור', בעוד שהקטגוריות בהן הוא פחות הצליח היו 'פוליטיקה' ו-'פשע'. תוצאות אלה תואמות את התוצאות שקיבלנו עבור הקטגוריות הנ"ל גם באמצעות מסווגי הבסיס. עם זאת ניכר שבכל הקטגוריות לעיל אחוזי ההצלחה של המסווג היו סבירים (מעל 80%).

ממבט מעמיק בערכי הבלבול של המטריצה ניתן להסיק שהקטגוריות 'פוליטיקה' ו-'פשע' קיבלו שתיהן ציונים נמוכים יחסית בעיקר כתוצאה מפגיעה הדדית זו בזו: כ-8% ממאמרי הקטגוריה 'פוליטיקה' סווגו כ-'פשע', וכ-6% ממאמרי הקטגוריה 'פשע' סווגו כשייכים ל-'פוליטיקה'. אנו משערים שאחת הסיבות לתופעה הנ"ל הייתה הכללתה של תת-הקטגוריה 'שחיתות פוליטית' (Political Corruption) במאגר הקטגוריות שלנו תחת הקטגוריה 'פוליטיקה', כאשר אינטואטיבית המאמרים המשתייכים לקטגוריה זו מתקשרים גם לפוליטיקה וגם לפשע.

סיכום ומסקנות

התחלנו את הפרויקט עם רקע בסיסי בלבד בלמידת מכונה, ובפרט בעיבוד שפות טבעיות, ועכשיו כשהגענו לסוף הדרך אנו מרגישים שהפרויקט העמיק מאוד את הניסיון והידע שלנו בתחומים אלה.
המטרה הראשונה של הפרויקט - בחינת שיטות וקטוריזציה לביצוע document embedding באמצעות פלט הכלי BERT, לימדה אותנו כי גם שיטות פשוטות יחסית של ביצוע וקטוריזציה, כמו וקטוריזציה של המשפט הראשון בלבד, יכולות להניב תוצאות יפות מאוד בבעיות סיווג מורכבות ובנוסף דורשות הרבה פחות זמן ריצה ביחס לוקטוריזציה של המאמר בשלמותו.
המטרה השנייה של הפרויקט – בניית המסווג ההיררכי, תרמה לנו בכך שהכרנו באמצעותה מודלים חדשים בלמידת מכונה, כמו רשתות נוירונים, ואף התנסנו בבניית מודלים מורכבים יותר בעצמנו (בניית המסווג ההיררכי ותכנון האלגוריתם שלו).
בשורה התחתונה אנו סבורים שהפרויקט השיג את מטרותיו והוכיח אמפירית שאכן ניתן לבצע הכללה ולהשתמש במודל של BERT לצורך קלסיפיקציה של מסמכים שלמים, ואף להגיע בעזרתו לתוצאות טובות מאוד אף ביותר מרמת קלסיפיקציה אחת.
בנוסף לכל אלו למדנו גם ששלב איסוף הנתונים, אשר נראה לנו במבט ראשון טכני ופשוט למדי, התגלה בסוף כבעיה לא פשוטה כלל, ולמדנו על משמעותו של מאגר הנתונים כאבן בניין משמעותית מאוד בתהליך הקמת פרויקט בבינה מלאכותית.

נקודות למחקר עתידי

על אף שהפרויקט שלנו הפך בסוף לרחב יותר מכפי שציפינו תחילה, אנו עדיין מרגישים שיש עוד המון נקודות לבחון במחקר עתידי בתחום זה. נציג שתי נקודות עיקריות שאנו חושבים שכדאי לבחון בעתיד:

משקלים למשפטים

בשיטות הוקטוריזציה המיודעות שאותן בחנו (Full Text, Summary) הענקנו משקל שווה לכל משפט בתהליך הוקטוריזציה (כזכור בשיטות אלה הוקטור שהתקבל עבור המאמר היה ממוצע וקטורי המשפטים המרכיבים את החלק שעליו מסתמכת שיטת הוקטוריזציה). הבעיה שנוצרת בגישה זו הינה שהיא מעניקה משקל שווה למשפטים עיקריים ותפלים ביצירת הוקטור הסופי. בעוד שמשפטים עיקריים במאמר (כגון המשפט הראשון) מכילים לרוב מידע שמגדיר את המאמר בצורה טובה, קיימים גם משפטים תפלים שמתן משקל גבוה מדי עבורם עלול להקשות על תהליך הסיווג. לכן אנו סבורים שיכול להיות מעניין האם ניתן לגלות מהם המשפטים העיקריים/התפלים במאמר ולתת להם משקלים שונים בהתאם לחשיבותם בתהליך יצירת הוקטור.

ניצול רב יותר של יכולות זיהוי ההקשר של BERT

כפי שהזכרנו כשהצגנו את המודל של BERT, החידוש העיקרי שהמודל מציג על פני מודלי word embedding קודמים הינה היכולת של המודל לזהות את המשמעות הסמנטית של מילים כתלות בהקשר שלהן בתוך משפט. אנו חושבים שיכול להיות מעניין לחקור כיצד ניתן לנצל את היכולת הזו של BERT באופן מעמיק יותר בכדי לשפר את תהליך הקלסיפיקציה.

ביבליוגרפיה

[1] M.-W. C. K. L. K. T. Jacob Devlin, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 11 October 2018. [Online]. Available: https://arxiv.org/pdf/1810.04805.pdf】.

[2] M. S. Z. RIZVI, "Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework," 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/.

[3] C. McCormick, "BERT Word Embeddings Tutorial," 2019. [Online]. Available: https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/.

[4] Q. Zhao, "Mapping Marine Ecosystems Of The World," [Online]. Available: http://proceedings.esri.com/library/userconf/oceans16/papers/oceans_14.pdf.

[5] L. Chen, "Support Vector Machine — Simply Explained," 2019. [Online]. Available: https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496.

ספריות Python מרכזיות בהן השתמשנו

1. Wikipedia-API – ממשק המאפשר לכרות מאמרים מויקיפדיה בקלות
2. pytorch_pretrained_bert – מימוש BERT עבור Pytorch, שימש לתהליך הוקטוריזציה
3. Numpy, Pytorch – פעולות על וקטורים ורשימות
4. NLTK – ספריית ה-NLP של Python, שימשה לפירוק המאמרים למשפטים
5. scikit-learn – אלגוריתמי הלמידה של המסווגים הבסיסיים
6. TensorFlow – backend עבור רשתות הנוירונים
7. Keras – ה-API לבניית רשתות הנוירונים
8. jsonpickle – שמירת מאגר המידע והתוצאות בקבצי JSON לשימוש חוזר

נספח – רשתות עצביות – Artificial Neural Networks

אז מהי רשת עצבית?

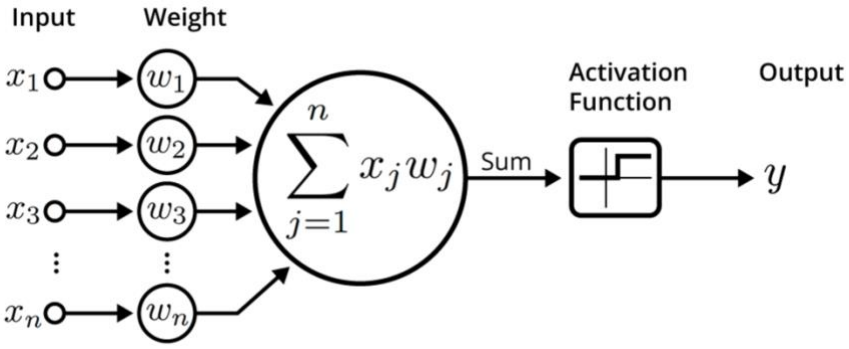
רשת עצבית (הידועה גם בכינוי: רשת נוירונים) הינה מודל מתמטי חישובי שפותח בהשראת התהליכים הקוגניטיביים המתרחשים ברשתות העצביות הטבעיות במוחם של אורגניזמים. במוח, מערכת העצבים מורכבת ממספר רב של תאי עצב, המכונים גם נוירונים, כאשר כל תא עצב כזה מורכב ממספר חלקים:

דנדריטים – שלוחות קצרות המוליכות את האותות הנשלחים לתא מתאים אחרים. הדנדריט ממיר את המידע שנקלט לאות חשמלי בעוצמות שונות ומוליך אותו לגוף התא.

גוף התא – קולט את האותות החשמליים מהדנדריטים, מעבד אותם, ומפיק פלט.

אקסונים – שלוחות ארוכות האחראיות על שידור הפלט לנוירונים אחרים.

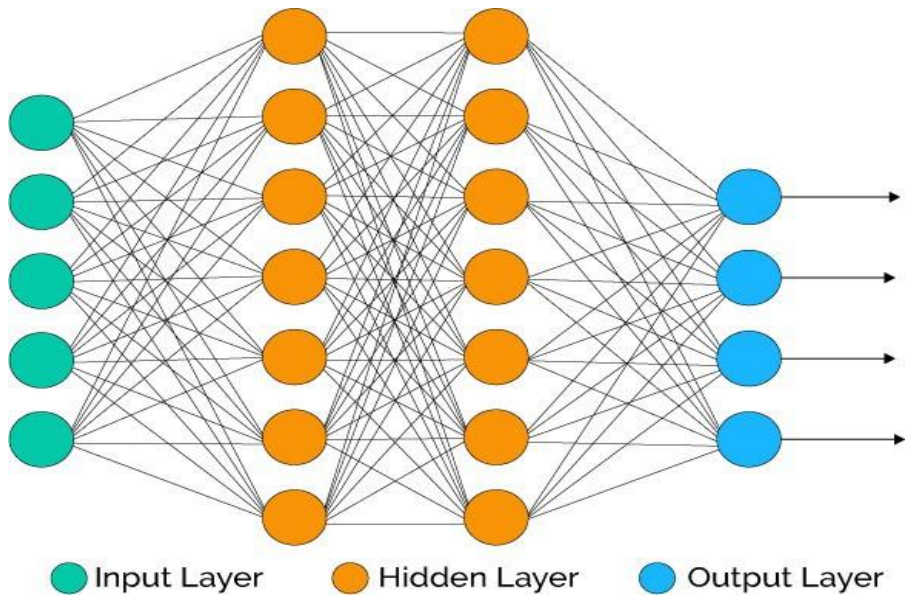
באופן דומה, רשתות עצביות מלאכותיות ממודלות גם הן באמצעות מספר רב של יחידות מידע (המכונות גם כאן, באופן לא מקרי – נוירונים) המקושרות זו לזו ומעבירות מידע אחת לשנייה. כל נוירון בנוי כך שהוא מקבל n קלטים מספריים, מכפיל אותם בסט של n משקלות, סוכם את המכפלות, מפעיל על התוצאה המתקבלת פונקציית אקטיבציה מסוימת, ולבסוף מעביר את הפלט לנוירונים הבאים ברשת.



פונקציית האקטיבציה בה משתמשים משמשת (בין היתר) בהפיכת פלטים בטווחים משתנים לפלטים בטווח חסום וידוע, ובנוסף בהוספת 'עומק' ליכולות רשת, שלולא פונקציית האקטיבציה הייתה פשוט אוסף של פולינומים ממעלה ראשונה (הכפלות של קלטים במשקלים וסכימה) בעלת יכולות לימוד מוגבלות.

ישנן מספר רב של פונקציות אקטיבציה אפשרויות: Sigmoid, Tanh, ReLU וכו', שאותן נחקור ועליהן נפרט עוד בהמשך בשלב הניסויים של הפרויקט.

רשת נוירונים בסיסית (סדרתית) עלולה להיראות כך:



ברשת ישנן 3 סוגי שכבות:

- שכבת כניסה (input layer) – לכל נוירון בשכבה זו כניסה אחת, כאשר מספר הנוירונים בשכבה הוא כמספר התכונות (features) ב-dataset.
- שכבות חבויות (hidden layers) – לכל נוירון בשכבה יש מספר כניסות אפשריות, לכל היותר כמספר הנוירונים בשכבה שלפניו.
- שכבת יציאה (output layer) – מספר הנוירונים בשכבה זו הוא כמספר ה-classes שאליהן רוצים לסווג, כאשר לכל נוירון מספר כניסות שהוא לכל היותר כמספר הנוירונים בשכבה החבויה האחרונה.

מספר הנוירונים בכל שכבה חבויה ומספר השכבות החבויות הינם פרמטרים הניתנים לכיוון (tuning). המספרים הללו קובעים את גודל הרשת, כאשר בשימוש ברשת קטנה מדי יש סיכון שהרשת לא תוכל לקרב את המיפוי הנדרש בצורה מספקת, ואילו אם משתמשים ברשת גדולה מדי מסתכנים בזמן לימוד ארוך למדי וב-overfitting. אנו נדבר עוד על כיוון הפרמטרים הללו בשלב הניסויים של הפרוייקט.

כיצד מתבצע תהליך הלמידה ברשת עצבית (פשוטה)?

ברשת נוירונים בסיסית, בהינתן dataset חדש, בכדי "ללמוד" את ה-data מאותחלים משקלי הרשת (המשקלים של כל נוירון) לערכים כלשהם, ואז מתבצעים בלולאה שני תהליכים מרכזיים:

זרימה קדימה (Feed-Forward) – בהינתן סט המשקלים הנוכחי, הקלטים מוזנים לשכבת הקלט ואז מתבצעים החישובים בנוירונים בשכבות השונות עד שלבסוף התוצאה מפעפעת לשכבת הפלט. לדוגמא, כיוון שבפרויקט זה אנו נתעסק בסיווג למספר קטגוריות של מאמרים שייוצגו ע"י וקטורים, בשלב זה יוזן הוקטור המייצג מאמר כלשהו לשכבת הקלט של הרשת (כזכור מס' הנוירונים בשכבה זו זהה למס' הכניסות בוקטור) ואז יתבצעו חישובים בשכבות הרשת השונות עד שהתוצאה תפעפע לשכבת הקלט. בשכבה זו, אם לדוגמא

נשתמש בפונקציית אקטיבציה מסוג softmax, יתקבל וקטור תוצאות שאורכו כמספר הקטגוריות, ולכל קטגוריה נקבל את ההסתברות (מספר בין 0 ל-1) שהמאמר משתייך לאותה הקטגוריה.
פעפוע לאחור (Back-Propagation) – בשלב זה מתבצעות האופטימיזציות: בכדי למזער את פונקציית ה-loss, הרשת מחשבת עבור כל משקל של כל נוירון את ההשפעה של שינוי בו על ה-loss. כלומר, עבור כל משקל מחושבת הנגזרת של ה-loss כתלות באותו המשקל, ולבסוף, כתלות בערך הנגזרת הרשת בוחרת האם להגדיל/להקטין את אותו המשקל בהתאם. שיטת אופטימיזציה אפשרית שבאמצעותה אפשר להחליט כיצד לשנות את ערך המשקלים לפי אופי הנגזרת היא Stochastic Gradient Descent, שאותה הכרנו כבר בעבר בקורס 'אלגוריתמים נומריים'. שיטת אופטימיזציה פופולארית נוספת מכונה "adam". גם את פרמטר שיטת האופטימיזציה (optimizer) של הרשתות נחקור בהמשך.

שני השלבים הנ"ל מתבצעים שוב ושוב באופן סדרתי על כל דגימה ב-dataset. כל מעבר שלם כזה על הדגימות ב-dataset מכונה 'epoch', וגם הוא פרמטר חשוב לו השפעה ישירה על ה-overfitting של המסווג המתקבל. גם את הפרמטר הזה נחקור בהמשך.

כיצד מתבצע תהליך הסיווג ברשת עצבית?
בתום תהליך הלמידה המשקולות שהתקבלו ברשת נשמרים על מנת שנוכל להשתמש בהם בעתיד לצורך סיווג. כך, בהינתן דגימה חדשה אותה נרצה לסווג, נטען לרשת את המשקולות ששמרנו, נזין את וקטור הדגימה לשכבת הקלט של הרשת ו"נריץ" את הרשת על הדגימה עם אותם המשקולות. לבסוף, בהינתן וקטור הפלט של הרשת (שאורכו כאמור כמס' הקטגוריות שאליהן נרצה לסווג) נבחר בקטגוריה שקיבלה את הציון הגבוה ביותר.