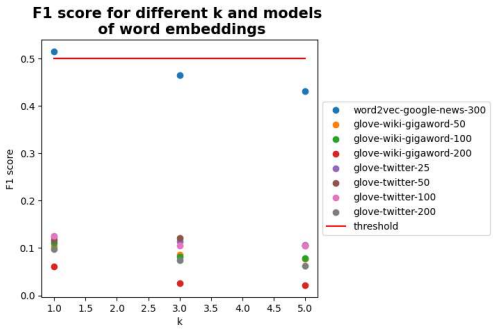
אימון

מודל 1

במודל זה מימשנו מודל KNN. השתמשנו במודלים המאומנים מראש של ספריית genism כדי לבנות ייצוג וקטורי לכל מילה. אם למילה לא היה ייצוג במודל, אתחלנו אותה להיות וקטור אפסים בגודל הווקטור של המודל הנוכחי שהשתמשנו בו (לדוגמא 200 במקרה של 'glove-twitter-200'). בחנו כמות שכנים שונה ביחד עם מודלים שונים מהספרייה. להלן תוצאות האקספלורציה:

	k = 1	k = 3	k = 5
word2vec-google-news-300	0.514	0.465	0.431
glove-wiki-gigaword-50	0.103	0.086	0.076
glove-wiki-gigaword-100	0.111	0.082	0.078
glove-wiki-gigaword-200	0.061	0.026	0.021
glove-twitter-25	0.124	0.114	0.107
glove-twitter-50	0.117	0.121	0.105
glove-twitter-100	0.125	0.105	0.105
glove-twitter-200	0.098	0.074	0.062



מודל 2

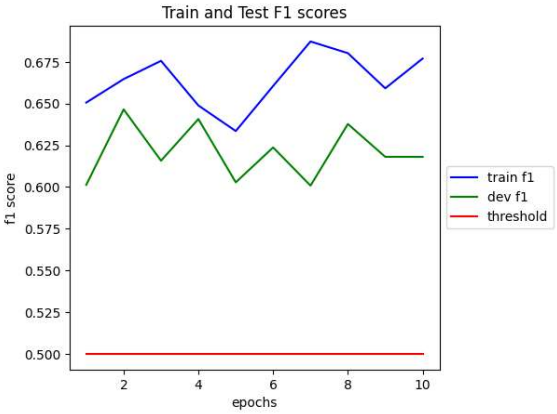
במודל זה בנינו רשת Feed Forward בעלת הארכיטקטורה הבאה:

3 שכבות ליניאריות עם 2 שכבות חבויות. כמות נוירונים בכל שכבה חבויה: 128

פונקציית אקטיבציה: Tanh , פונקציית loss: Cross entropy אופטימייזר: Adam עם קצב למידה של 0.01.

ניסנו מספר מודלים שונים לייצוג הוקטורי של המילה, ולאחר הרבה ניסיונות בחרנו בייצוג הבא:
השתמשנו ב2 מודלים מובנים: word2vec-google-news-300 ו glove-twitter-50. את 2 המודלים שרשרנו לכדי וקטור אחד וכך יצרנו ייצוג מכליל יותר. במידה ולמילה אין ייצוג באחד או יותר מהמודלים הנ"ל, שמנו ייצוג מובנה של oov (out of vocabulary) במקרה של glove ווקטור אפסים במקרה של word2vec.

בסך הכול, קיבלנו וקטור באורך 350.
המחשבה מאחורי השרשור הזה והשימוש ב2 מודלים שונים היא שאנו רוצים להימנע ככל הניתן מייצוג חסר משמעות במרחב של מילה (oov או אפסים). כאשר מילה לא מופיעה במודל אחד, ייתכן כי היא מופיעה במודל אחר ועל כן נצמצם את התופעה של חוסר הייצוג ונבנה נפח רחב יותר של וקטורים למודל.
להלן התוצאות שקיבלנו:



מודל 3

עקב התוצאות הטובות של ה embedding שתואר במודל 2, החלטנו לדבוק בו גם לחלק התחרותי. כדי לשפר את התוצאות שלנו, החלטנו לממש רשת LSTM. לפי הנלמד בהרצאה ובתרגול, רשת זו מסוגלת לתפוס קונטקסט של מילים במשפט. במשימת NER, הקונטקסט והסדר של המילים חשוב לסיווג מילה ב named entity שכן כך המודל יכול ללמוד את הקשרים בין המילים ובאילו מקרים הופיע שם כזה או אחר. הפעלנו את הרשת על כל משפט לחוד, כלומר כל משפט הוא batch של מילים.

הרשת הראשונית שבנינו הייתה רשת LSTM בעלת שכבה 1, מימד חבוי של 64, ואחרי זה שכבה של fully connected. התוצאות לא היו מספקות בעינינו. המרנו את הרשת לאחת עם מבנה מורכב יותר: bi-LSTM (LSTM דו כיווני) עם מימד חבוי של 64, רשת fully connected עם 2 שכבות וביניהן פונקציית האקטיבציה Tanh. אופטימייזר Adam עם קצב למידה 0.01 ופונקציית loss של Cross entropy loss.
גם התוצאות של רשת זו לא היו יציבות ומספקות לכן החלטנו לעבור לפן תיאורטי יותר.

חקרנו רבות באינטרנט על משימת ה NER וראינו כי בהרבה מאמרים ואתרים מציינים כי במשימה זו ישנה בעיה של class imbalance. הכוונה היא שכמעט בכל dataset כמות השמות קטנה משמעותית מאשר שאר המילים שאינן שמות. כתוצאה מכך, כל לומד יתקשה לסווג שמות כבאלו שכן כמות המידע שיש לו עליהן קטנה מאוד לעומת המידע העצום שיש לו על ה class השני, מילים שאינן שמות. כדי להתמודד עם תופעה זו, נחשפנו ל2 שיטות שיוכלו לעזור למודל ללמוד בצורה טובה יותר ומימשנו אותן על מדגם האימון:

1. **Unser sampeling**: השמטת דאטה מהקלאס הדומיננטי. במשימה שלנו, נשמיט משפטים שיש להם מעט named entities ובכך נאזן את הדאטה שאנו מתאמנים עליו. הגדרנו סף לכמות ישויות מינימלית בכל משפט והורדנו מסט האימון משפטים שלא עמדו בסף.
2. **Over sampling**: הוספת דאטה מלאכותי. דגימה חוזרת של דאטה משמעותי ל class הרצסיבי (שנמצא במיעוט). במשימה שלנו, נדגום שוב משפטים שהיו בהם הרבה named entities. הגדרנו סף (אחר מהסף הקודם) לכמות ישויות מינימלית והוספנו לסט האימון משפטים שעמדו בסף זה.
3. **משקול של ה loss**: כל דוגמא בדאטה תקבל משקל. דוגמאות מה class הדומיננטי יקבלו משקל נמוך יותר מאשר דוגמאות מה class הרצסיבי. בכך המודל מעניש (במובני loss) על טעויות בקלאס הרצסיבי יותר מאשר על טעויות ב class הדומיננטי. לכל class הגדרנו את המשקל כהופכי ליחס שלו בדאטה. המשקל חושב בצורה הבאה:

$$w_i = \frac{num - samples}{2 * num - samples(i)}$$

כלומר, המשקל של ה class ה i הוא כמות הדוגמאות לחלק לכמות הדוגמאות מהקלאס ה i כפול כמות הקלאסים (במקרה שלנו 2). כך מחשבים משקל balanced בדומה למימוש הפנימי של ספרית sklearn. קראנו על שיטה זו במאמר[1] שמצורף.

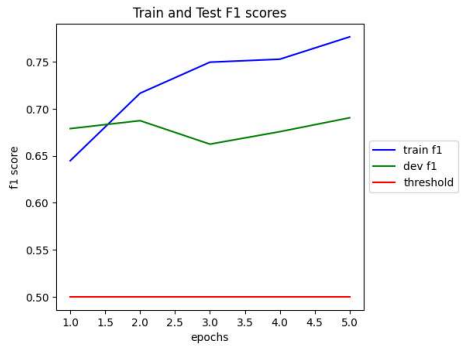
להלן כמה הרצות עם thresholds שונים לכמות מינימלית של ישויות עבור under sampling וכמות מינימלית של ישויות ל over sampling:

[1] https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/

```
underSample_threshold:  0 overSample_threshold:  8
Train Data:
dropped: 2166 sentences
added: 64 sentences
1 ratio: 0.13981650013001004
0 ratio: 0.86018349986999

Dev Data:
dropped: 0 sentences
added: 0 sentences
1 ratio: 0.07945083582279286
0 ratio: 0.9205491641772071

Classes weights: [0.58127132 3.57611583]
```

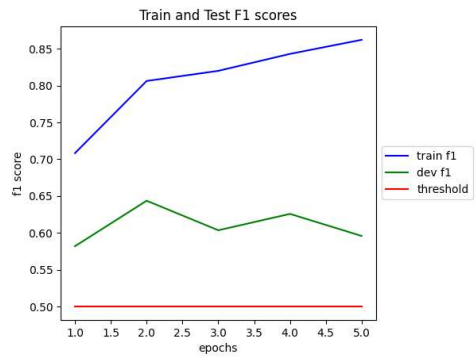


(1)

```
underSample_threshold:  2 overSample_threshold:  6
Train Data:
dropped: 2946 sentences
added: 228 sentences
1 ratio: 0.24849417748628028
0 ratio: 0.7515058225137197

Dev Data:
dropped: 0 sentences
added: 0 sentences
1 ratio: 0.07945083582279286
0 ratio: 0.9205491641772071

Classes weights: [0.66533084 2.01211958]
```



(2)

ניתן לראות כי שיטות אלו, בשימוש נכון (קרי דוגמה 1) העלו את ה F1 score. כאשר נעשה שימוש יתר בשיטות אלו (דוגמה 2) ה score יורד ואף משתווה לרשת פשוטה. לסיכום השתמשנו בארכיטקטורה ובשיטות שתוארו במודל התחרותי שלנו, כאשר השיטות יושמו לפי דוגמה 1. בחלק התחרותי השתמשנו בפונקציית ה sigmoid על ה output שלנו ותייגנו מילה כישות רק אם ההסתברות שלה להיות ישות, לפי הרשת, הייתה גבוה או שווה ל0.7. לאחר כמה הרצות וניסיונות תיוג ראינו כי ערך סף זה עובד הכי טוב.

מבחן

מודל 1

```
model: word2vec-google-news-300, k: 1,  F1 score: 0.514
```

מודל 2

```
Epoch 10 of 10
------------------------------
train F1: 0.676985121805003
dev F1: 0.6180639935196436

Max F1: 0.6465
```

מודל 3

```
Epoch 5 of 5
------------------------------
train F1: 0.7764786014303554
dev F1: 0.690406976744186

Max F1: 0.6904
```