

### למידה עמוקה עבודה 3

#### :Data exploration

הבעיה: הנתונים מתייחסים לנתוני חיפוש ותוצאות שלקוחות ביצעו ברשת Home Depot. עבור כל רשומה בנתונים הכוללת את הכותרת של המוצר אותו חיפש הלקוח ותיאור התוצאה שהתקבלה דירגו את הרלוונטיות של התוצאה לחיפוש ע"י ביצוע מבחן בו 3 שופטים התבקשו לדרג את הרלוונטיות של התוצאה לחיפוש ע"י אחד משלושת הדירוגים:

- Irrelevant (1)
- Partially or somewhat relevant (2)
- Perfect match (3)

לבסוף לקחו את הממוצע של שלושת הדירוגים. כך קיבלנו את ערך הרלוונטיות לכל רשומה שנע בטווח 1-3 כאשר יש בדיוק 13 ערכים אפשריים. ניתן להתייחס לבעיה כבעיית רגרסיה בכך שיש לחזות את הרלוונטיות של התוצאה לשאלה על ידי מספר בין 1 ל-3, לעומת זאת ניתן להתייחס לבעיה כבעיית סיווג בכך שיש לסווג כל זוג של חיפוש-תוצאה לאחד מ-13 ערכי הרלוונטיות השונים. אנו ניסינו להתייחס לבעיה ב-2 האופנים האלו כפי שנתאר בהמשך.

הנתונים מחולקים ל-2 קבצים עיקריים - train, test כך שtrain מכיל 74067 רשומות וtest מכיל 166693. בהסבר על הנתונים בkaggle היה נתון שיש חיפושים בtest שכבר המודל ראה בעבר בtrain אך גם כאלו שלא ראה בעבר. בדקנו מהו מספר הsearch\_items הייחודיים בtrain ובtest:

```
train unique search terms 11795  
test unique search terms 22427
```

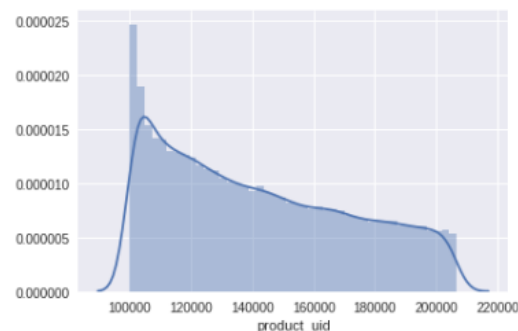
זאת אומרת שבוודאות נתקל בלפחות 10,632 ביטויי חיפוש ב-test שלא נתקלנו בהם בעבר ונצטרך לחזות בצורה הטובה ביותר את ההתאמה שלהם לתוצאת החיפוש שלהם.

ה-label של כל זוג (ביטויי חיפוש, תוצאות חיפוש) הוא בעצם מידת ההתאמה של תוצאת החיפוש לביטויי החיפוש המבוטאת במספר בין 1 ל-3 כאשר 3 מתאר את מידת ההתאמה הגבוהה ביותר ו-1 מתאר את מידת ההתאמה הנמוכה ביותר. כלומר מדובר במשימת **חיזוי**.

לא קיימים ערכים חסרים בנתונים.

התפלגות הנתונים:

- התפלגות המוצרים שהחיפוש מחזיר בקובץ train:



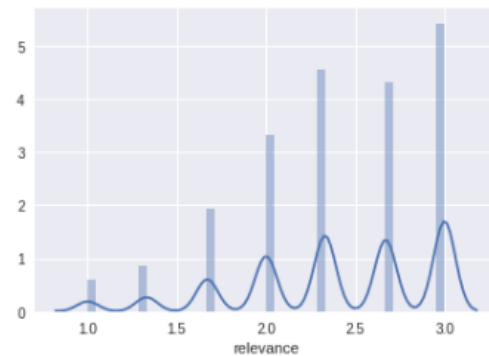
- התפלגות המוצרים שהחיפוש מחזיר בקובץ test:

רוני מינדלין מילר 302242870  
מיה קרמר 204219976



ניתן לראות שהתפלגות המוצרים שהחיפוש מחזיר בקבצי train וה-test דומה כלומר כנראה שאנשים מחפשים לרוב את אותם המוצרים או לחלופין שהמערכת מחזירה לרוב את אותם המוצרים גם עבור חיפושים שונים (פחות סביר מאחר וציון ההתאמה של רוב החיפושים הוא לרוב ציון טוב) או שילוב של השניים.

בדקנו כיצד מתפלגים ציוני ההתאמה של המוצרים:



כפי שציינו לעיל, רוב החיפושים מסתיימים בתוצאות מאוד טובות (ציון התאמה בטווח 2.5-3), חלק גדול מהחיפושים מסתיימים בתוצאה בינונית (ציון התאמה בטווח 2-2.5) ומעט חיפושים מסתיימים בתוצאות לא טובות (ציון התאמה בטווח 1-2). זאת אומרת שהנתונים הם **imbalanced** ולכן בהמשך נשתמש בשיטות להתמודד עם בעיה זו. כמות סיווגים לכל class:

רוני מינדלין מילר 302242870  
מיה קרמר 204219976

3.00	19125
2.33	16060
2.67	15202
2.00	11730
1.67	6780
1.33	3006
1.00	2105
2.50	19
2.25	11
2.75	11
1.75	9
1.50	5
1.25	4

ניתן לראות שמעט מאוד רשומות קיבלו את הדירוגים: 1.25, 1.5, 1.75, 2.25, 2.5, 2.75, 2.00

### חלק 1- Character level LSTM

הערה: בראשית הדרך חיברנו בין שדה title ושדה product\_description עבור כל מוצר על מנת לנסות לקבל כמה שיותר מידע על המוצר. עם זאת לאחר התייעצות עם נתי ומאחר והריצה הייתה מאוד ארוכה עד כדי כך שלא הצלחנו לקבל תוצאות בזמן סביר החלטנו לא להשתמש בשדה product\_description על מנת שנוכל לשפר את המודל ולנסות מודלים נוספים הרצים בזמן סביר מאשר הרצה של מודל עיקרי אחד בזמן לא סביר. בנוסף ניסינו להוסיף חלק מה-product\_description (לקחנו את כל 400 התווים הראשונים) אך זה לא שיפר את תוצאות המודל וגרם לכך שזמן האימון יהיה ארוך יותר. להלן דוגמה לאחד ה-epochs שהתקבלו:

```
Train on 59251 samples, validate on 14816 samples
Epoch 1/5
59251/59251 [=====] - 84s 1ms/step - loss: 0.9543 - val_loss: 0.2874
Epoch 2/5
59251/59251 [=====] - 82s 1ms/step - loss: 0.2857 - val_loss: 0.2851
Epoch 3/5
59251/59251 [=====] - 83s 1ms/step - loss: 0.2858 - val_loss: 0.2851
Epoch 4/5
59251/59251 [=====] - 82s 1ms/step - loss: 0.2860 - val_loss: 0.2851
Epoch 5/5
59251/59251 [=====] - 82s 1ms/step - loss: 0.2856 - val_loss: 0.2851
rmse 0.5339770017852502
```

### Preprocess

ראשית יצרנו data frame בשם all\_df שהיא בעצם חיבור של שתי ה-data frames של train וה-test על מנת להכין את הנתונים על בסיס תווים הנמצאים בכל הנתונים. חילקנו את הטקסט בשדות search\_trem ו-product\_title. והסרנו תווים שלאחר מעבר על הנתונים הגענו למסקנה שאינם מוסיפים מידע אלא עשויים לגרום לרעש. התווים אותם הסרנו:

```
char_to_remove = [' ', '{', '}', '"', '(', ')', '.', ',', '&', '[', ']', '!', '!', '~', '\\',
'\x80', '\x81', '\x84', '\x89', '\x8b', '\x90', '\x93', '\x95', '\x99', '\x9a',
'\x9d', '\xa0', 'ı', 'ç', '•', 'À', 'Â', 'É', 'Ê', 'Ë', 'İ', 'Ò', 'Û',
'Ü', 'à', 'ä', 'å', 'è', '÷']
```

התווים הייחודיים שנתרו:

```
array(['<uniq>', '!', '#', '$', '%', '*', '+', '-', '/', '0', '1', '2',  
      '3', '4', '5', '6', '7', '8', '9', ':', ';', '=', '?', '\\', 'a',  
      'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',  
      'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '\\x80',  
      '..', 'z'], dtype='<U6')

```

את התו '<uniq>' אנו הוספנו על מנת לתאר את ה-padding ב-0 שיתווסף בהמשך.

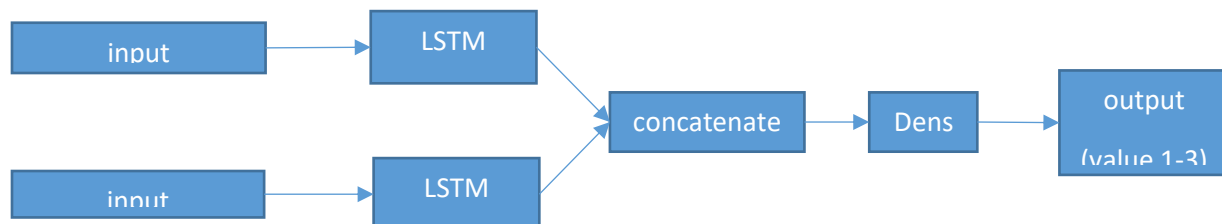
לאחר החלוקה לתווים המרנו כל תו למספר באמצעות label encoder ובכך קיבלנו עבור כל רשומה שתי מערכים של מספרים, אחד עבור search\_term ואחד עבור product\_title.

כעת בדקנו מהו האורך המקסימלי של כל המערכים שהתקבלו (מספר תווים מקסימלי) תחת הפיצ'רים search\_term ו-product\_title ובכך קיבלנו שתי ערכים prod\_max\_len=128 ו-search\_max\_len=58.

ביצענו padding של אפסים בתחילת כל מערך על מנת שכל המערכים השייכים לאותו פיצ'ר יהיו באותו האורך. קבענו שאורך המערכים יהיה כאורך המקסימלי (כלומר prod\_max\_len ו-search\_max\_len בהתאמה). יש לציין שגם ניסינו לבצע padding לאורך הממוצע של המערכים (וחיתוך למערכים הארוכים מהאורך הממוצע) אך קיבלנו תוצאות מעט פחות טובות ולכן בחרנו להשאיר את ה-padding לאורך המקסימלי.

כעת בנינו את המודל הראשון והבסיסי:

שיטת ולידיציה: חלוקה של קובץ הtrain הנתון לtrain ו-validation כאשר גודל הvalidation הוא 0.2 מגודל הtrain.

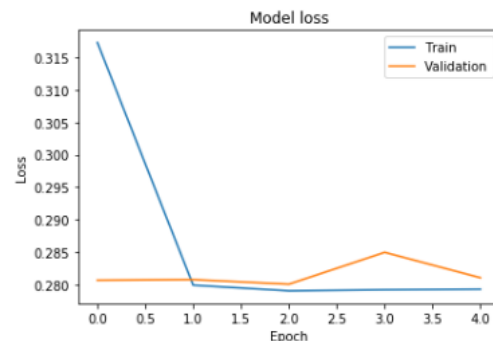


יש לציין שבמודל יש שכבת LSTM אחת אך היא מקבלת 2 inputs שונים ולכן גם יש לה 2 outputs שונים ולכן המודל משורטט כאילו מדובר ב-2 מודלים שונים שמתמזגים למודל אחד (בשלב ה-concatenate). אנו מציינות זאת מאחר וזה לא היה לנו ברור מאילו מהמאמר ולכן אנו חושבות שחשוב לציין זאת.

לאחר אימון הרשת על עם epoch=5 ו-batch\_size=72 קיבלנו:

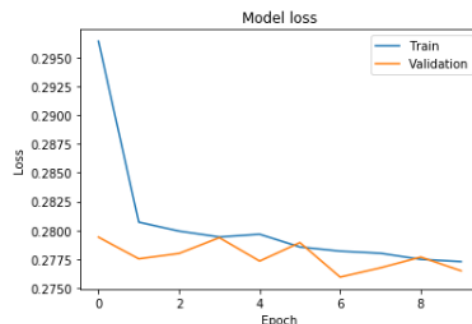
רוני מינדלין מילר 302242870  
מיה קרמר 204219976

```
Train on 59254 samples, validate on 14813 samples
Epoch 1/5
59254/59254 [=====] - 104s 2ms/step - loss: 0.3173 - mean_squared_error: 0.3173 - val_loss: 0.2806 - val_mean_squared_error: 0.2806
Epoch 2/5
59254/59254 [=====] - 102s 2ms/step - loss: 0.2799 - mean_squared_error: 0.2799 - val_loss: 0.2807 - val_mean_squared_error: 0.2807
Epoch 3/5
59254/59254 [=====] - 102s 2ms/step - loss: 0.2790 - mean_squared_error: 0.2790 - val_loss: 0.2800 - val_mean_squared_error: 0.2800
Epoch 4/5
59254/59254 [=====] - 102s 2ms/step - loss: 0.2792 - mean_squared_error: 0.2792 - val_loss: 0.2849 - val_mean_squared_error: 0.2849
Epoch 5/5
59254/59254 [=====] - 102s 2ms/step - loss: 0.2792 - mean_squared_error: 0.2792 - val_loss: 0.2810 - val_mean_squared_error: 0.2810
0:08:33.459564
```



ניתן לראות שיש שיפור בערך ה-loss של הtrain בתהליך הלמידה אך מה שחשוב הוא ה-loss של ה-validation והוא לא משתפר, ומתחיל מערך יחסית נמוך. ננסה לשפר את המודל.

ניסינו להגדיל את מספר הניורונים בשכבת ה-LSTM מ-50 ל-100 והגדלנו את מספר ה-epochs מ-5 ל-10 אך עדיין אין ירידה משמעותית ובאופן הדרגתי עבור ה-loss של ה-validation:



כעת הסתכלנו על הבעיה ב-2 דרכים שונות עם שיטות ולידציה שונות:

1. בעיית רגרסיה (כפי שהצגנו לעיל) עם k-fold cross validation.

2. בעיית סיווג עם StratifiedKFold - ניתן לעשות זאת מכיוון שמספר הערכים הניתנים במשתנה המטרה relevance שווה ל-13 ערכים שונים אפשריים. הסיבה לכך היא מאופן חישוב ה-relevance - נבחרו 3 שופטים לדירוג הרלוונטיות של המוצר לחיפוש שנעשה וה-relevance נקבע לפי ממוצע הדירוגים של 3 השופטים. במקרה זה נשתמש ב-StratifiedKFold כדי לאמוד את ביצועי המודל שלנו כיוון ששיטה זו מבצעת פיצול של הנתונים ל-folds כך שהתפלגות הנתונים ב-folds השונים תהיה זהה להתפלגות

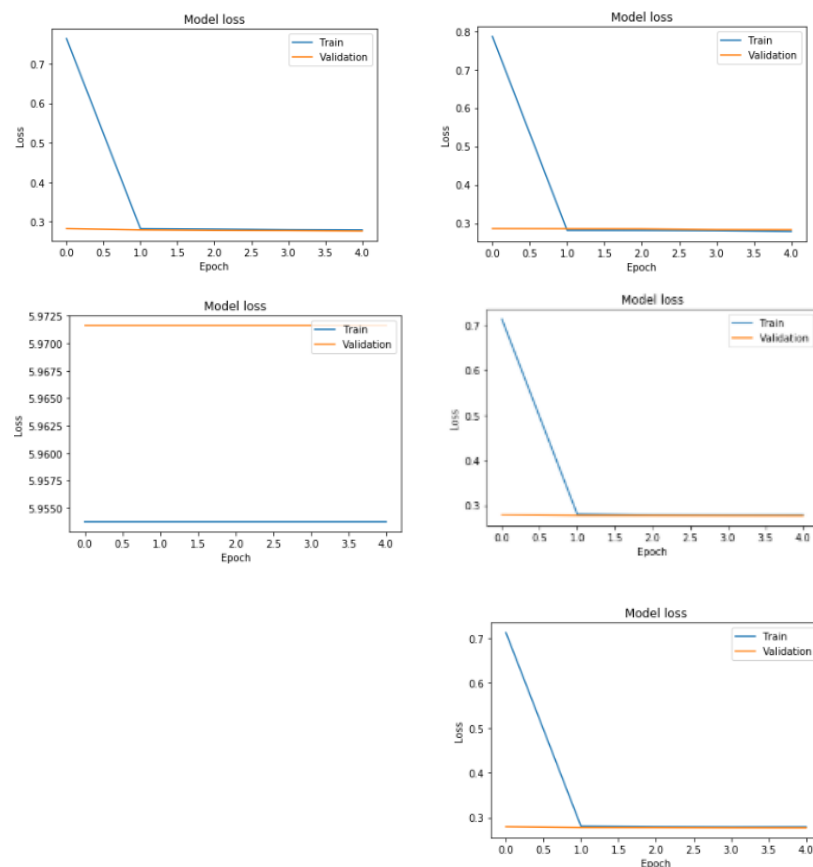
הנתונים ב-data המקורי ואכן זה מתאים לבעיה שלנו מאחר ויש imbalance בנתונים כפי שהצגנו בתחילת הדוח.

3. בעיית רגרסיה עם StratifiedKFold - הסיבה שלא השתמשנו ב-StratifiedKFold עבור בעיית הרגרסיה ב-1 היא ששיטת ולידציה זו דורשת שה-class לא יהיה מספר רציף. החלטנו לנסות להשתמש בשיטה זו גם עבור בעיית רגרסיה ע"י המרת  $\gamma$  למספרים באמצעות label\_encoder רק עבור הפיצול ע"י StratifiedKFold ושימוש ב- המקורי עבור אימון המודל.

תוצאות שיטה 1:

מודל כפי שהוסבר ושורטט לעיל רק שבשכבה האחרונה פונקציית האקטיבציה היא relu ופונקציית ההפסד היא mse.

להלן גרפים המתארים את loss בכל אחד מחמשת ה-epochs שביצענו:



ניתן לראות שבדף השלישי loss גרוע מאוד. ההנחה שלנו היא שמאחר והנתונים הם imbalanced אז ספציפית הרשומות שנבחרו עבור הfold הזה אינן טובות בכך שלוקח למודל יותר זמן לעבד אותן ואולי לא ניתן למצוא דפוסים ברשומות אלו בכלל (מאחר וה- loss של train גם לא טוב).

יש לציין כי ב-folds האחרים יש מעט שיפור ב- loss של validation אך לא רואים זאת בבירור בגרף מאחר ובהתחלה loss על train היה משמעותית יותר גבוה, לדוגמה פלט של אחד מה-folds שהצגנו:

רוני מינדלין מילר 302242870  
מיה קרמר 204219976

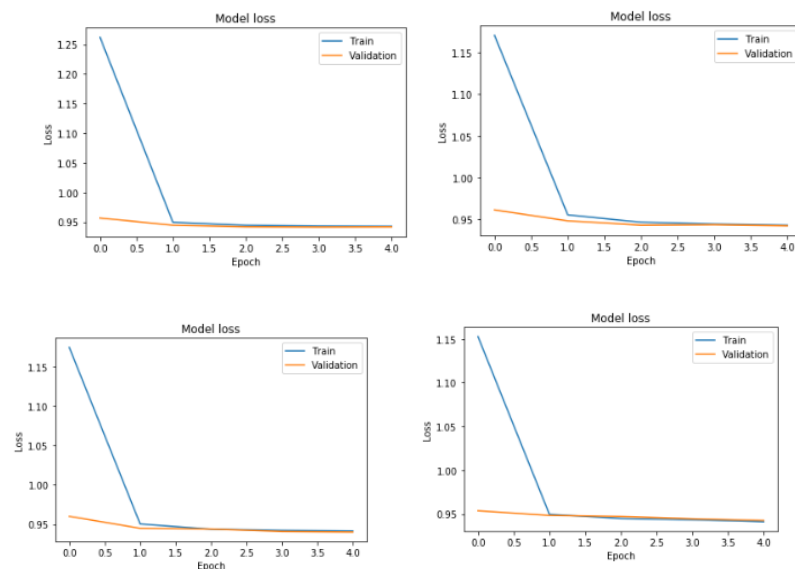
```
Train on 59253 samples, validate on 14814 samples
Epoch 1/5
59253/59253 [=====] - 38s 635us/step - loss: 0.7858 - mean_squared_error: 0.7858 - val_loss: 0.2859 - val_mean_squared_error: 0.2859
Epoch 2/5
59253/59253 [=====] - 36s 608us/step - loss: 0.2813 - mean_squared_error: 0.2813 - val_loss: 0.2855 - val_mean_squared_error: 0.2855
Epoch 3/5
59253/59253 [=====] - 36s 602us/step - loss: 0.2811 - mean_squared_error: 0.2811 - val_loss: 0.2852 - val_mean_squared_error: 0.2852
Epoch 4/5
59253/59253 [=====] - 36s 604us/step - loss: 0.2804 - mean_squared_error: 0.2804 - val_loss: 0.2832 - val_mean_squared_error: 0.2832
Epoch 5/5
59253/59253 [=====] - 36s 601us/step - loss: 0.2785 - mean_squared_error: 0.2785 - val_loss: 0.2829 - val_mean_squared_error: 0.2829
rmse 0.5319170518434212
```

ניתן לראות שקיים שיפור בloss של validation

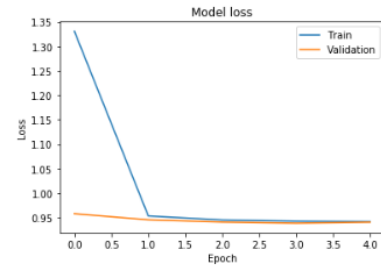
fold	rmse
1	0.531
2	0.525
3	2.443
4	0.526
5	0.525
mean	0.91

תוצאות שיטה 2:

מודל כפי שהוסבר ושורטט לעיל רק שבשכבה האחרונה פונקציית האקטיבציה היא softmax, פונקציית ההפסד היא categorical\_crossentropy וה- $\gamma$  נתון כ-one-hot:



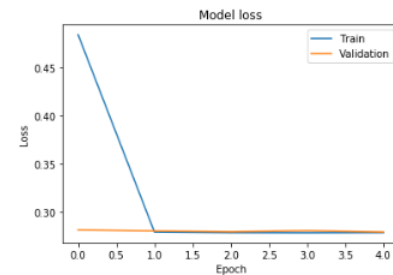
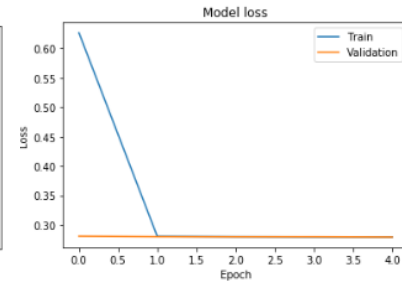
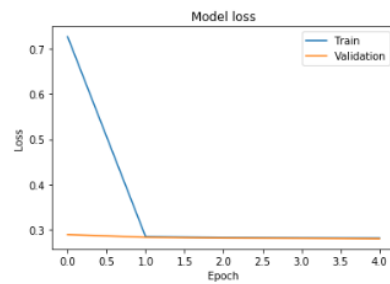
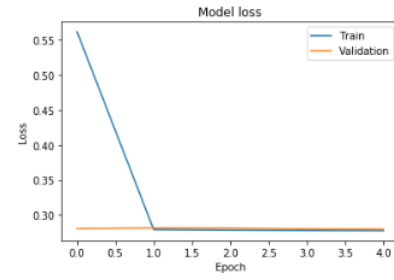
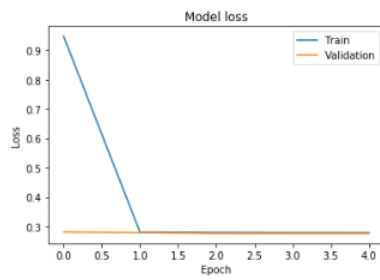
רוני מינדלין מילר 302242870  
מיה קרמר 204219976



fold	accuracy
1	0.581
2	0.581
3	0.581
4	0.580
5	0.581
mean	0.581

תוצאות שיטה 3:

מודל כפי שהוסבר ושורטט לעיל רק שבשכבה האחרונה פונקציית האקטיבציה היא relu ופונקציית ההפסד היא mse.





רוני מינדלין מילר 302242870  
מיה קרמר 204219976

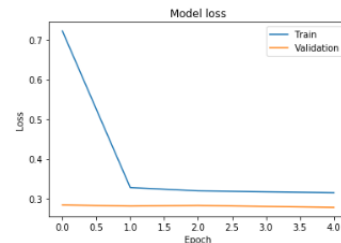
fold	rmse
1	0.528
2	0.529
3	0.528
4	0.528
5	0.526
mean	0.528

ניתן לראות שאכן בשימוש ב-StratifiedKFold עבור בעיית רגרסיה אנו כבר לא מקבלות את המקרה בו ה-loss היה קבוע וגבוה מאוד עבור הtrain והvalidation כפי שראינו במקרה 1 שבו השתמשנו בk-fold וחשדנו שמצב זה התקבל מהחלוקה הלא הוגנת מבחינת ערך המטרה בשימוש בk-fold.

**שיפורים:** נמשיך עם הגישה השלישית כלומר התייחסות לבעיה כבעיית רגרסיה ושימוש בשיטת ולידציה StratifiedKFold:

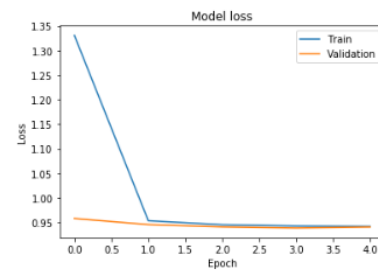
ניסנו לשפר את המודל ע"י הוספת שכבת dense עם 50 ניוונים לאחר שכבת concat שלאחריו מגיע גם שכבת dropout (עם הסתברות 0.3) על מנת למנוע over fitting ואחרי שכבת dropout שכבת dense אחרונה שהיא output ולכן היא עם ניוון אחד בלבד. בנוסף הוספנו אחרי שכבת ה-LSTM עוד שכבת של batch normalization

קיבלנו שה-rmse השתפר במעט. בנוסף הloss של הtrain פחות טוב מה-loss של הvalidation לאורך כל תהליך האימון כתוצאה מהוספת שכבת ה-dropout. דוגמה לepoch אחד בתהליך ה-cross validation:

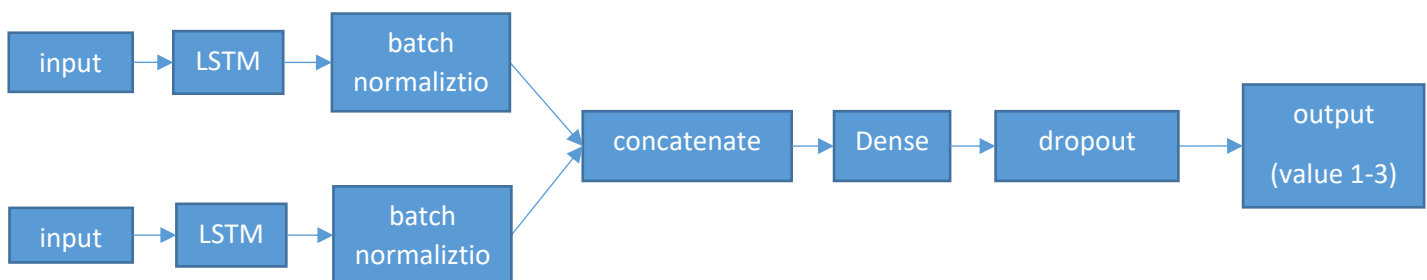


```
Train on 59251 samples, validate on 14816 samples
Epoch 1/5
59251/59251 [=====] - 32s 535us/step - loss: 0.7220 - val_loss: 0.2844
Epoch 2/5
59251/59251 [=====] - 27s 463us/step - loss: 0.3280 - val_loss: 0.2816
Epoch 3/5
59251/59251 [=====] - 27s 462us/step - loss: 0.3197 - val_loss: 0.2832
Epoch 4/5
59251/59251 [=====] - 28s 464us/step - loss: 0.3173 - val_loss: 0.2804
Epoch 5/5
59251/59251 [=====] - 27s 462us/step - loss: 0.3152 - val_loss: 0.2781
rmse 0.5273664109699436
```

לכן הקטנו את dropout ל-0.1 ואכן כעת הvalidation והtrain loss בעלי קרוב מאוד:

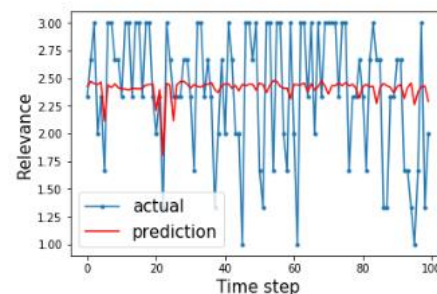


כלומר, המודל הנוכחי שהתקבל:

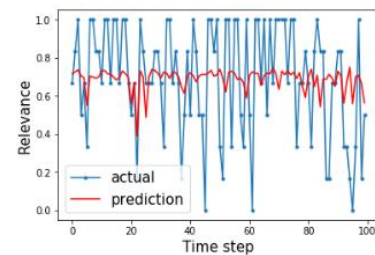
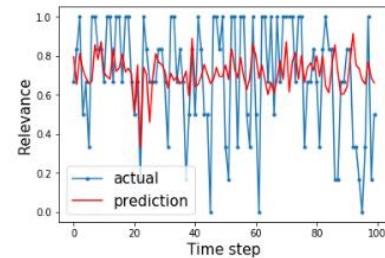


לאחר קריאת קובץ relevance\_instructions הנתון המכיל את ההוראות שניתנו לשופטים על מנת לדרג את הזוגות חיפוש-תוצאה, הבנו שניתן לשופטים דגש מיוחד להתייחס ל-Brand, Material, and Functionality של כל מוצר ולפי זה לדרג את הרלוונטיות תוצאות החיפוש למוצר. לכן החלטנו להוסיף רק את שדות אלו מקובץ attributes.csv אל שדה title של נתוני הtrain והtest. מאחר וכעת האורך המקסימלי של התווים שקיבלנו בשדה title הוא 303, החלטנו שכעת נבצע את padding ע"י הממוצע של אורך הtitle ולא ע"י הערך המקסימלי. האורך הממוצע הוא בגודל 66 תווים.

לאחר בחינת תוצאת המודל, הבנו שרוב החיזויים הניתנים הם בטווח קטן יחסית:



מפה הגענו למסקנה שהמודל לא מצליח לחזות ערכים גדולים/קטנים גם אם מופיעים בשכיחות גבוהה בנתונים כמו למשל דירוג 3- יש לדירוג זה הכי הרבה רשומות (מעל 19000) ועדיין המודל לא חוזה אותו אפילו פעם אחת. החלטנו שיש לנרמל את ערכי הרלוונטיות- נרמלנו ע"י min\_max normalization שמביא את ערכי הרלוונטיות לערכים בין 0 ל-1, ובדקנו את התוצאות החדשות:



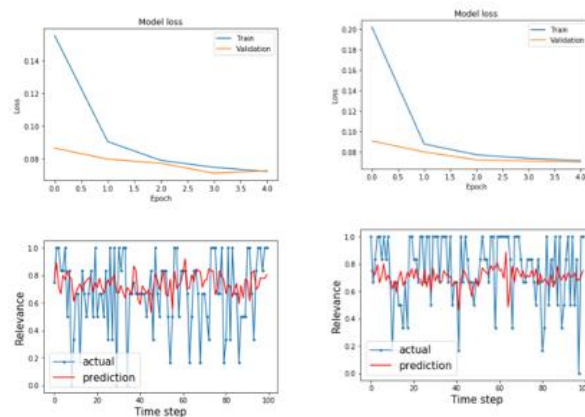
ניתן לראות שאכן טווח חיזוי המודל גדל! ואכן ה-rmse הממוצע השתפר: 0.271 (לא בהכרח שיפור כל כך משמעותי כי צמצמנו את ערכי המטרה ולכן גם טווח השגיאה קטן) **בטבלה הסופית התייחסנו לערכי המטריקות רק לאחר המרת משתנה המטרה חזרה לטווח המספרים המקורי שלהם.**

כעת הבנו דבר נוסף- המודל נותן חשיבות גדולה יותר לתווים בעלי קידוד מספרי בעל ערך גבוה יותר, למשל אם a מקודד ל-1 ו-z מקודד ל-24 אז z מקבל חשיבות גבוהה יותר בתהליך האימון! לכן שינונו את הקלט למודל כך שכל char ייצוג לא ע"י הקידוד המספרי שלו אלא ע"י one hot, ולכן אם עד עכשיו המודל היה מקבל קלט במימדים: (n\_samples, avg\_length, 1) אז עכשיו הקלט למודל יהיה במימדים:

(n\_samples, avg\_length, n\_unique\_char)

מאחר והמעבר ל-one-hot תופס המון מקום בזיכרון לא הצלחנו לבצע יותר מ-2 split עבור 5 epochs כאשר מתייחסים לpadding לאורך הממוצע (66 תווים). העדפנו לוותר על splits מאשר לחתוך יותר תווים מהקלט לאחר בדיקת שיפור rmse.

תוצאות:



רוני מינדלין מילר 302242870  
מיה קרמר 204219976

```
mean rmse 0.2670321973538482
end time 246.814
mean train time 68.003
mean pred time 43.9555
```

מכיוון שהkernel נפל כל הזמן ולא הצלחנו לבצע commit במלואו, החלטנו לצמצם את רשימת התווים הייחודים (כמו למשל ?, =, \*, !, ועוד), ולבדוק אם הדבר פוגע ברmse ואם כעת יהיה ניתן להריץ את kernel במלואו:

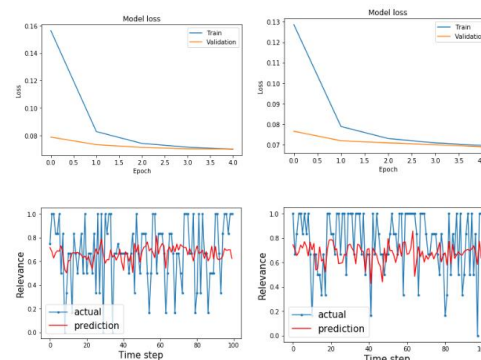
התווים הייחודים החדשים שהתקבלו:

```
array(['<uniq>', '#', '%', '+', '/', '0', '1', '2', '3', '4', '5', '6',
      '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
      'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
      'x', 'y', 'z', '-', 'z'], dtype='<U6')

```

כעת מספר התווים הייחודים הוא 43.

תוצאות:



```
mean rmse 0.2627451185252867
end time 347.59
mean train time 108.74549999999999
mean pred time 51.427
```

rmseנ אפילו מעט ירד! לכן נמשיך בשיטה זו.

יש לציין שניסינו להשתמש גם בSMOTE על מנת ליצור רשומות מלאכותיות להגדלת סט הנתונים על מנת שהמודל יוכל להתאמן על יותר דוגמאות בעלות דירוג נמוך יחסית (מאחר ויש סיווגים רק עם 4 רשומות למשל ולכן המודל לא לומד מהן בכלל) אך לא הצלחנו להגיע לתוצאות כתוצאה מעומס על הזיכרון מאחר והוספנו רשומות נוספות.

תוצאות הגשה לתחרות:

הגשנו את קובץ הsample submission שיצרנו ע"י קבלת ממוצע הprediction על test set מתהליך הcross validation וקיבלנו את התוצאות (בטבלה הסופית מופיעות כל המטריקות שבדקנו):

Siamese net rmse on test set: 0.529

## Feature extraction

מודלי classic ml שחברנו להשתמש בהם כקלט לתוצאות feature extraction הם XGBRegressor ו-LGBMRegressor. מדובר במודלים המבוססים על בניית ensembles של עצי החלטה בצורה איטרטיבית כך שבנייה של כל עץ תלויה בתוצאות של העצים הקודמים שנבנו. ההבדל בין 2 המודלים הנ"ל הוא ש-XGB נבנה בצורת רוחבית (פיתוח כל הקדקודים באותה השכבה ואז עובר לפיתוח השכבה הבאה) לעומת LGBM שנבנה לעומק כלומר פיתוח הקדקודים לעומק- לכן גם יותר מהיר משמעותית וצורך פחות זיכרון.

נחלק את הtrain ל-train ו-val. לאחר אימון המודל על הtrain שהתקבל, שלפנו את השכבה החמישית (שזוהי שכבת concat) ונתייחס אליה כפלט של המודל הנועד לקבלת הfeatures. את הקלט של המודל נשאיר כמו שהוא כלומר:

```
concat_layer = siamese_model.layers[5].output
feature_model = Model(siamese_model.input, concat_layer)
feature_model.compile(loss='mse', optimizer='adam')
```

כעת נחזה את ערך ה-train וערך ה-val ונבדוק את rmse שהתקבל עבור החיזוי ע"י כל אחד מהמודלים: XGBRegressor ו-LGBMRegressor. כאשר הקלט לאותם מודלים הוא הפלט שהתקבל מהמודל feature\_model שתיארנו לעיל וערכי ה-relevance המתאימים.

לאחר ביצוע tuning לכל אחד מהמודלים על-פי ערך rmse שהתקבל עבור ה-validation set, הרצנו את המודלים האלו על כל נתוני ה-train set וה-test set הנתונים, בתהליך כפי שתואר לעיל (קודם חילוף הפיצ'רים ממודל Siamese ולאחר מכן הכנסתם למודל ml) והגשנו submission של מודלים אלו והתוצאות הן (מופיעות גם בטבלה הסופית ביותר פירוט):

lgb rmse: 0.531

xgb rmse: 0.544

## חלק 2- word embeddings and word level LSTM

את תהליך עיבוד הנתונים ביצענו כפי שתיארנו לעיל (יצירת df\_all המכיל את titlen המורכבת מהשדות (title, Brand, Metrial).

כעת השתמשנו בTokenizer של keras על מנת לחלק את המשפטים לtokens, כאשר שינינו את הפילטר של tokenizer כך שישנן רק את התווים הבאים: ~{ } \_ ^ [ \ ] ? < = > ; : . , + \* ( ) & \$ % !

לאחר מכן בדקנו מהו אורך המשפט המקסימלי עבור השדות product\_title ו-serach\_term, לקחנו את המקסימום מבין השניים, ובציענו padding לערך המקסימלי של אפסים בתחילת כל טקסט על מנת שכל הטקסטים יהיו באותו האורך.

כעת השתמשנו ב-GoogleNews-vectors-negative300 כבסיס embedding שלנו. יצרנו מטריצה בגודל

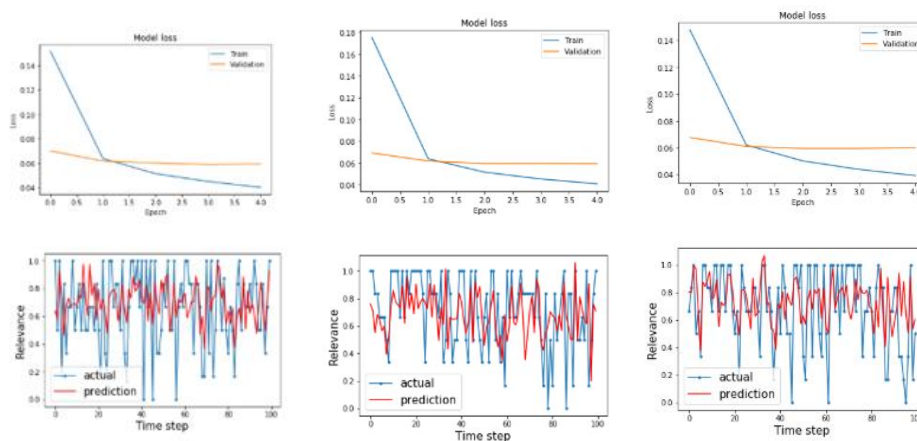
(n\_unique\_words, embedding\_size) המאותחלת בערכים רנדומליים, כך שכל שורה בה מייצגת מילה לפי האינדקס של אותה מילה הנוצר ע"י tokenizer. כעת החלפנו כל שורה במטריצה במשקלים של אותה המילה שקיבלנו מקובץ embedding של google. המילים הייחודיות שנמצאות במאגר המילים שלנו אך לא נמצאות במאגר המילים של google נותרו עם משקלים אקראיים ובתהליך הלמידה של המודל הם ישתנו ויקבלו ערכים מתאימים. יש לציין שהבחירה הטובה ביותר ליצירת embedding למילים שלנו תהיה לא להשתמש בembedding חיצוני בכלל וללמוד את embedding של המילים מהתחלה ע"י אימון המודל שלנו וכך נקבל embedding שמתאים לבעיה שלנו (קשרים שמופיעים אצלנו ולא הופיעו אצל google ולהפך) אך מפאת חוסר זמן החלטנו להשתמש בembedding של google ולאפשר למודל להתאמן גם על המשקלים (לא קיבענו את שכבת embedding ל-trainable=false) האלו כלומר נקודת ההתחלה שלנו לא אקראית לחלוטין וגם לא קבועה לפי google לאורך כל האימון.

כפי שהסברנו בחלק הראשון, נרמלנו את ערכי המטרה באמצעות min\_max normalization.

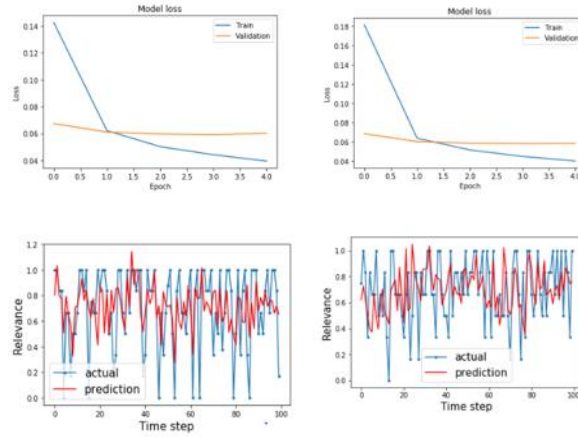
השתמשנו במודל זה למודל שבנינו בחלק בראשון רק ששינינו את גודל input ל-

(n\_sampels, max\_length, embedding size). גם פה השתמשנו בשיטת הוולידציה StratifiedKFold מאותן סיבות שכבר ציינו.

תוצאות המודל:

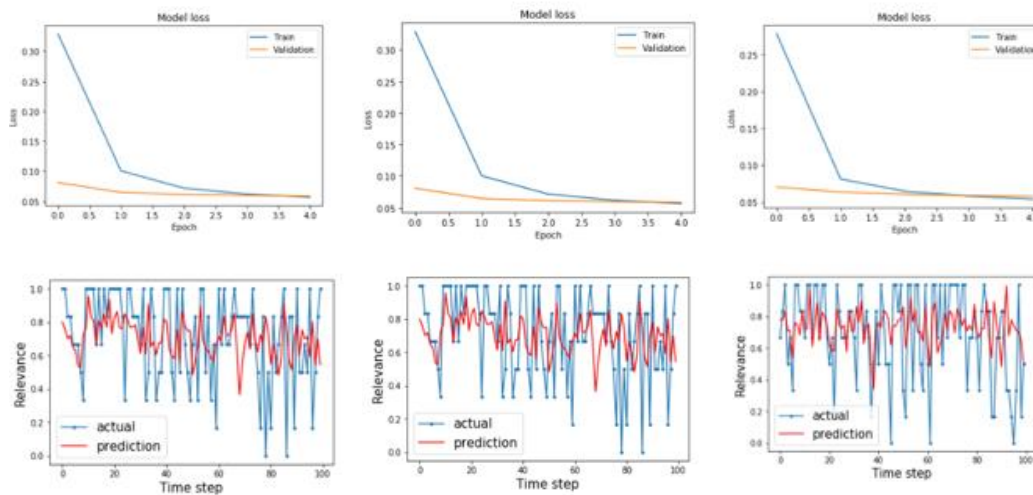


רוני מינדלין מילר 302242870  
מיה קרמר 204219976

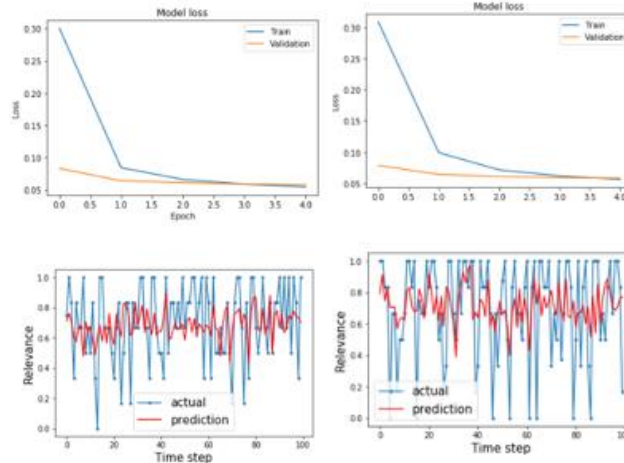


fold	Rmse
1	0.245
2	0.243
3	0.243
4	0.245
5	0.241
mean	0.243

ניתן לראות שה- $\text{loss}$  של הvalidation גבוה יותר מה- $\text{loss}$  של train. אנו חושדות שיש  $\text{over fit}$ , לכן הגדלנו את הdropout ל-0.5.



רוני מינדלין מילר 302242870  
מיה קרמר 204219976



fold	Rmse
1	0.241
2	0.241
3	0.242
4	0.240
5	0.241
mean	0.241

ניתן לראות שכבר אין over fitting וגם ה-rmse מעט עלה.

כעת כשהמודל הגיע לתוצאות הכי טובות שהצלחנו על validation, הגשנו הגשה של sample submission על ממוצע ההצאות של test set שהתקבל לאורך ה-cross validation שביצענו, קיבלנו:

rmse: 0.50774

התוצאה הטובה ביותר שקיבלנו עד כה!!!

### Feature extraction

תהליך האימון של מודלי ה-feature extraction זהה לתהליך כפי שהצגנו בחלק 1 של העבודה. ביצענו מעט tuning למודלים תוצאות (תוצאות סופיות נוספות ו-plots בטבלה הסופית):



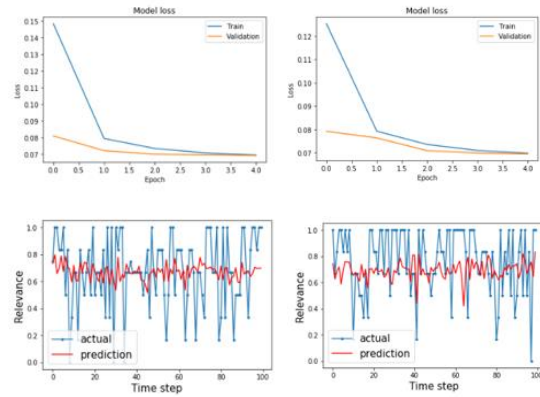
מִיָּה קָרְמֵר 204219976

סיכום ומסקנות:

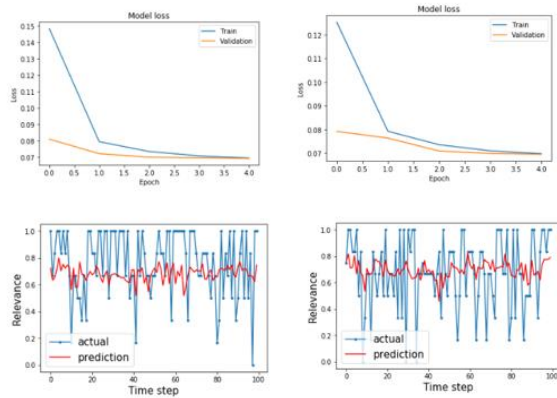
[illegible]

▪ Char level:

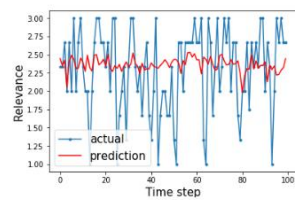
○ Siamese net train (2-stratified folds)



○ Siamese net validation (2-stratified folds)



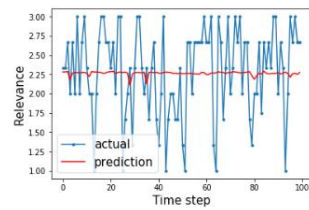
○ Siamese net test



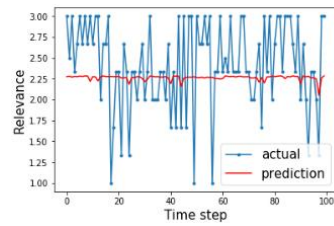
רוני מינדלין מילר 302242870  
מיה קרמר 204219976

- feature extraction – XGB

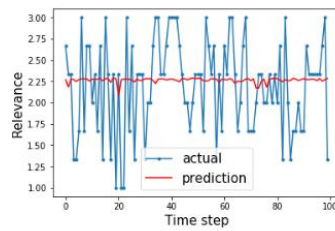
- test set:



- train set:

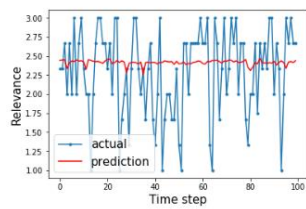


- validation set:

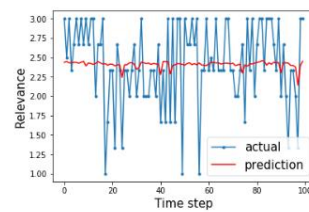


- feature extraction – LGB

- test set:

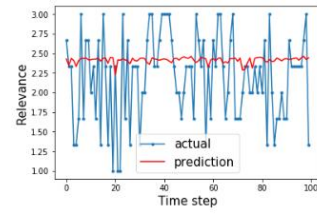


- train set:



- validation set:

רוני מינדלין מילר 302242870  
מיה קרמר 204219976



רעיונות לפיתוח אפשרי (חלקם ניסינו אך כתוצאה מהגבלה בזיכרון לא הצלחנו לממש):

- ביצוע embedding מאפס (ללא שימוש בembedding מוכן חיצוני) על מנת להתאים כמה שיותר את embedding לבעיה שלנו.
- שימוש בSMOTE או יוצר רשומות מלאכותיות אחר כלשהו על מנת לאזן את המחלקות בבעיה.
- הוספת המידע הנתון בטבלת product description על מנת לתת יותר מידע למודל על כל מוצר.

מסקנות:

השימוש בword level טוב יותר עבור המשימה הנוכחית מאשר השימוש בchar level. אנו סבורות כי ניתן לשפר את התוצאות ע"י הרעיונות שהצענו לעיל ולא מימשנו כתוצאה של חוסר זמן אך בעיקר חוסר באמצעים (זיכרון). בעבודה הזאת ההבנה המשמעותית ביותר הייתה בנושא של פונקציות האקטיבציה בשכבות המודל - גם בעבודות אחרות היינו צריכות לשנות את פונקציית האקטיבציה בהתאם למשימה אך פה מאחר וניסינו את 2 האפשרויות (חיזוי או סיווג) ממש ראינו כיצד טעות בבחירת פונקציות האקטיבציה של שכבות המודל עשויה להשפיע על כל התוצאות והיינו צריכות להתאים כל פעם את המודל לסוג המשימה. בנוסף הבנה משמעותית נוספת שקיבלנו היא התאמת מבנה הקלט והפלט לסוג המשימה. ביצענו reshape לא מעט פעמים לקלט המודל על מנת להתאימו למודל שלנו - בהתחלה הכנסנו את הקלט ללא one hot ולכן המימד השלישי של הקלט היה צריך להיות 1 ולקח לנו זמן להבין זאת (למרות שזה ברור מאילו - לא הבנו למה המודל לא מצליח לקבל את מבנה הקלט שהזנו ללא ביצוע reshape) ורק לאחר שהמודל זרק שגיאה הבנו שיש לבצע reshape לממדים:

(1, avg\_n\_characters\_in\_text, n\_samples). לאחר ביצוע one hot או embedding גם היינו צריכות לשנות את מבנה הקלט למודל אך כעת כבר היה לנו יותר ברור מהם המימדים שצריכים להזין בשכבת הinput.

לסיכום, ניסינו מודלים שונים ומגוונים וניסינו להתייחס לבעיה מזוויות שונות על מנת לבחון כיצד ניתן להגיע לתוצאות הטובות ביותר מבלי לקבל over fitting ואכן בחלק מהניסיונות שלנו הצלחנו לשפר את תוצאות המודל.