**תיעוד**

* לא לשכוח שבצענו Project characterization

חומרי קריאה:

לקחת מהרפרנסים של המצגת אמצע

* [http://karpathy.github.io/2015/05/21/rnn-effectiveness/](https://eur01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fkarpathy.github.io%2F2015%2F05%2F21%2Frnn-effectiveness%2F&data=02%7C01%7Ckiassi%40campus.technion.ac.il%7C6f096115164e4e6887dc08d85ad18f0f%7Cf1502c4cee2e411c9715c855f6753b84%7C1%7C0%7C637359203385747454&sdata=RdvMHVLj%2FnJj%2Bm6nBpX5lMUNzUXvoxuQom%2BBrV%2B3Yv8%3D&reserved=0)
* [https://www.tensorflow.org/tutorials/text/text\_generation](https://eur01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.tensorflow.org%2Ftutorials%2Ftext%2Ftext_generation&data=02%7C01%7Ckiassi%40campus.technion.ac.il%7C6f096115164e4e6887dc08d85ad18f0f%7Cf1502c4cee2e411c9715c855f6753b84%7C1%7C0%7C637359203385747454&sdata=71oOqiFirVlavN5Hi5Ag9k0RLatRWJh4fWklEYNppCE%3D&reserved=0)
* [https://machinelearningmastery.com/how-to-develop-a-word-level-neural-language-model-in-keras/](https://eur01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fmachinelearningmastery.com%2Fhow-to-develop-a-word-level-neural-language-model-in-keras%2F&data=02%7C01%7Ckiassi%40campus.technion.ac.il%7C6f096115164e4e6887dc08d85ad18f0f%7Cf1502c4cee2e411c9715c855f6753b84%7C1%7C0%7C637359203385757447&sdata=ip%2BzON%2BMnZlEykBbrgXIoIqmD5QH9KldJiLFrMkqiPk%3D&reserved=0)
* <https://machinetalk.org/2019/02/08/text-generation-with-pytorch/>
* <https://www.kdnuggets.com/2020/07/pytorch-lstm-text-generation-tutorial.html>
* @article{radford2019language,
* title={Language Models are Unsupervised Multitask Learners},
* author={Radford, Alec and Wu, Jeff and Child, Rewon and Luan, David and Amodei, Dario and Sutskever, Ilya},
* year={2019}
* }

@article{Wolf2019HuggingFacesTS,

title={HuggingFace's Transformers: State-of-the-art Natural Language Processing},

author={Thomas Wolf and Lysandre Debut and Victor Sanh and Julien Chaumond and Clement Delangue and Anthony Moi and Pierric Cistac and Tim Rault and Rémi Louf and Morgan Funtowicz and Joe Davison and Sam Shleifer and Patrick von Platen and Clara Ma and Yacine Jernite and Julien Plu and Canwen Xu and Teven Le Scao and Sylvain Gugger and Mariama Drame and Quentin Lhoest and Alexander M. Rush},

journal={ArXiv},

year={2019},

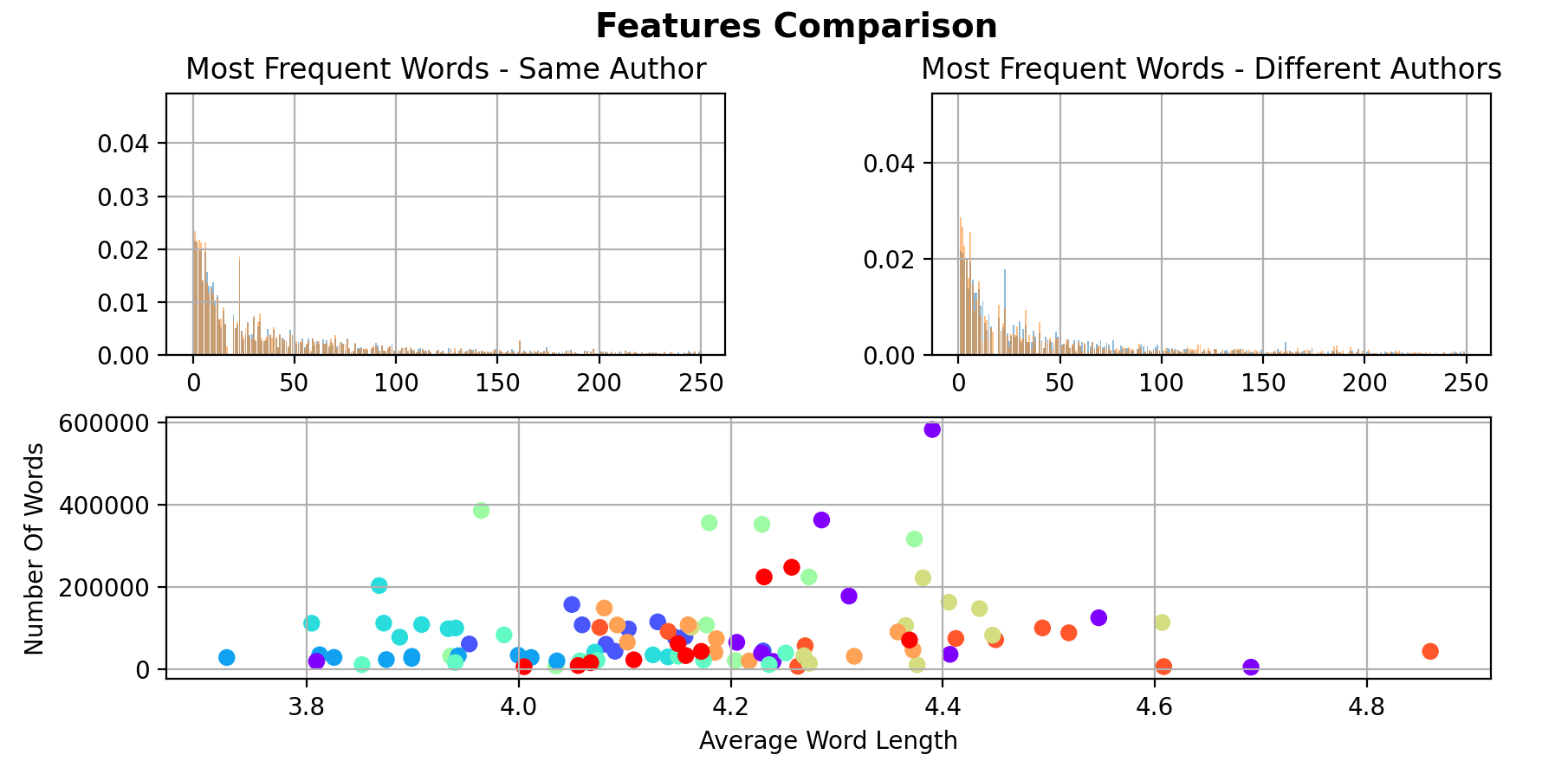
volume={abs/1910.03771}

}

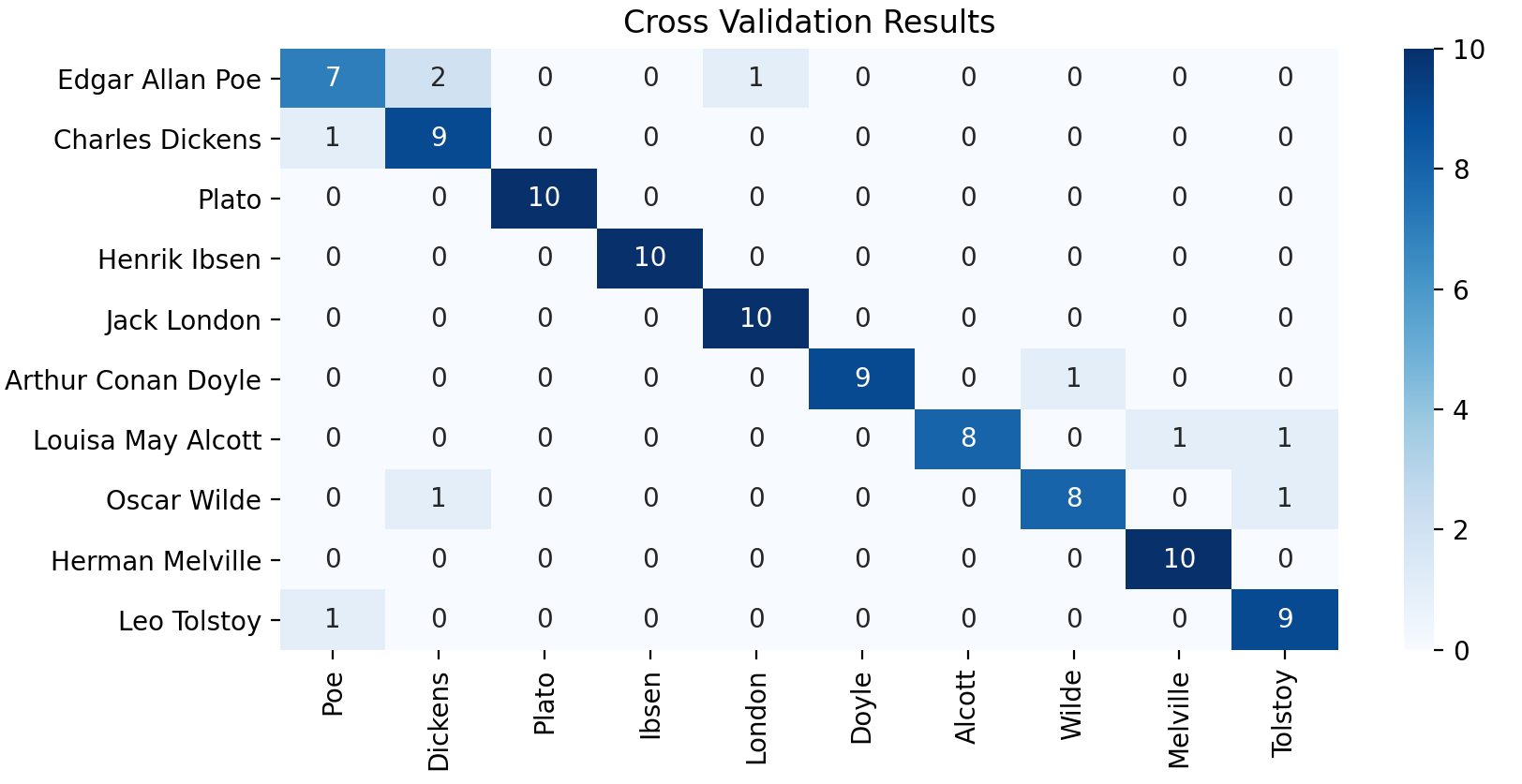
טוי פרובלם (הכל מופיע במצגת אמצע):

* משימת זיהוי כותבים פשוטה.
* דאטהסט קטן: 10 כותבים שלוקחים מכל אחד מהם 10 ספרים שכתב (באמצעות פרויקט Guttenberg). פיצול של 85/15% של טריינינג וטסט.
* השתמשנו במודל Scikit-learns SVC .
* מבחינת preprocessing:
  + “Cleaning” the texts
  + Tokenizing
  + Ignoring punctuation
* מבחינת פיצ'רים:
* Histogram of most frequent words
* Histogram of least frequent words
* Average word length
* A picture containing clock, meter, kitchen

  Description automatically generatedNumber of words

A screenshot of a cell phone

Description automatically generated

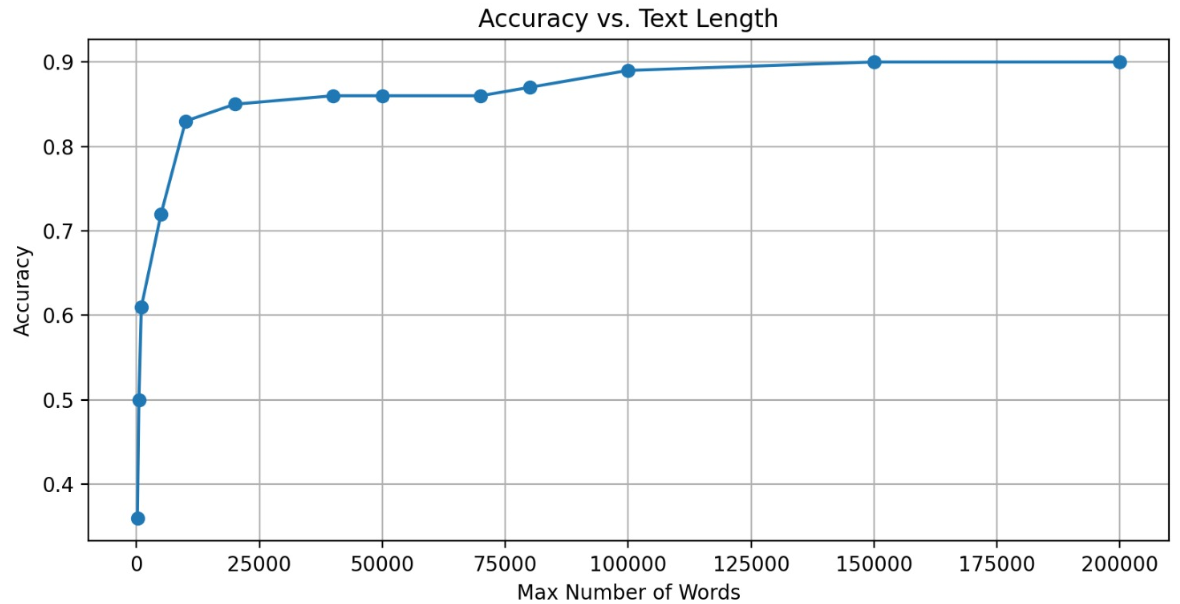
תוצאות:

דיוק ממוצע: 90%

על הפרויקט (גם במצגת אמצע):

* מטרה
* מוטיבציה
* (רקע על NLP ו-Authorship Attribution)
* אתגרים
* סקר ספרות
* סביבת עבודה - Python 3, Anaconda, pycharm, nltk ועוד.

בין לבין, בצענו בדיקה על הטוי פרובלם לבדוק דיוק למול אורך טקסט (לקחנו כל פעם מספר מסוים של מילים מטקסט ובדקנו):



כלומר, קיימת השפעה משמעותית של אורך הטקסט על הדיוק.

* הכנת ה-data בפרויקט (לפרט על כל פרסור: html לטקסט, פידיאף לטקסט, ניקיונות שעשינו, לחפור על זה שהשתמשנו בregex, הקודש כתבנו בפייתון, ניסיונות שעשינו ב-CMD, סידור ידני שעשינו לעמודות בפידיאף ועוד):
* BMJ:
  + הורדה של 190-200 reviews עם תיוג, עבור 9 כותבים, כדאטה לטסט.
* הורדה של 10 מאמרים לכל כותב כדאטה לאימון (הכותב לכל הפחות כותב מרכזי במאמר).
* עבודה ידנית של ניקוי ה-reviews (הורדה של דפי html וניקויים) והמאמרים (חיבור ידני של שתי עמודות לאחת מקבצי pdf, הרבה preprocessing שיפורט בהמשך ועוד).
* בינתיים: זה הדאטה איתו מתעסקים בפרויקט.
* Nips:
* הורדנו ופרסרנו (כולל ניקיון משמעותי של הטקסט) X reviews.
* F1000:
  + הורדנו מעל ל11 אלף reviews ופרסרנו אותם לקבצי טקסט נקיים.
* מאמרים של מרק:
  + הורדנו את המאמרים כpdfים בלבד.

Preprocessing:

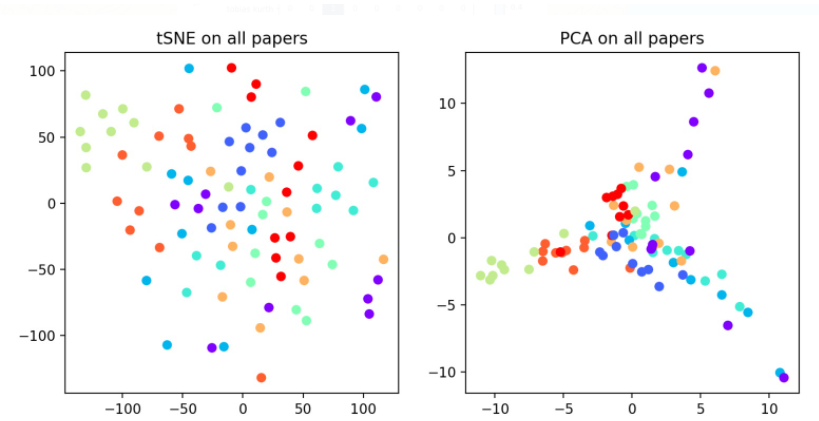
* ניקוי ידני של מידע לא רלוונטי ואף של מידע "מפליל" – שם הכותב, כתובת המייל שלו וכו'.
* עבודה ידנית של איחוד שתי עמודות לאחת.
* התעלמות מחלק מהניקוד.
* התעלמות מ-stopwords.
* Stemming.
* Lemmatization.

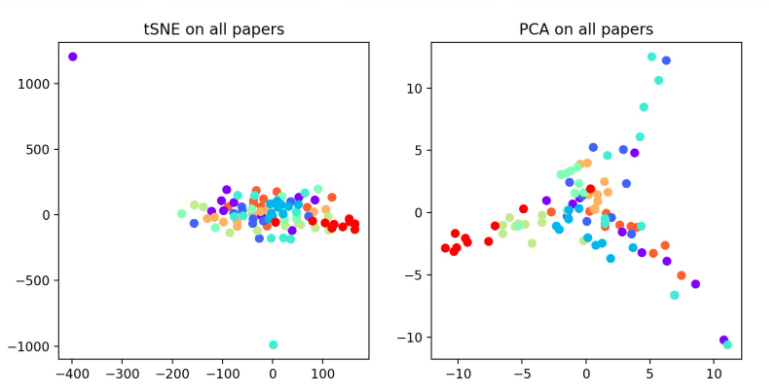
Features:

* היסטוגרמת 1-gram.
* היסטוגרמת 2-gram.
* היסטוגרמת 3-gram.
* TF-IDF.
* קלאסטרינג (בין כל המאמרים, בין מאמרים של אותו כותב, בין reviews(?)):
* מוטיבציה.
* שיטות שונות (PCA, tsne, dbscan).
* ניסוי וטעייה: שינויים שונים בפיצ'רים ובדיקת האשכול, שינויים שונים ב-preprocessing ובדיקה מתאימה.

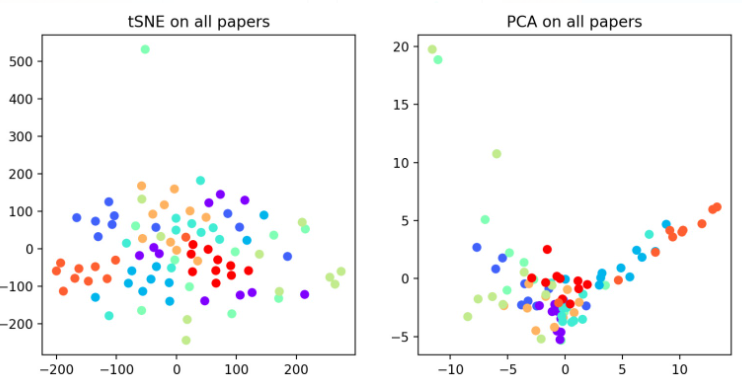
דוגמאות (בכל גרף – צבע זהה אומר שמדובר באותו כותב מאמר):

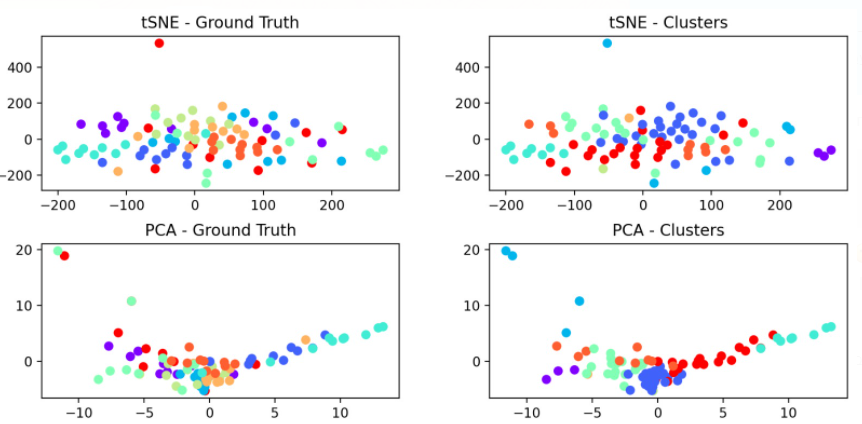
עם lemmatization + הורדת stopwords:



בלי ';' :

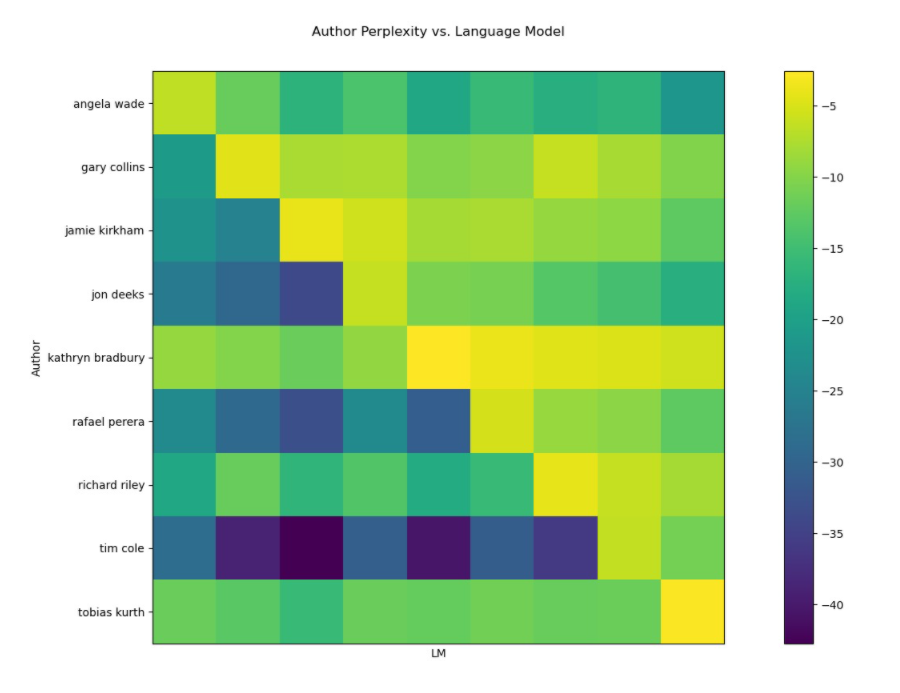
עם 4-grams ו- 5-grams:



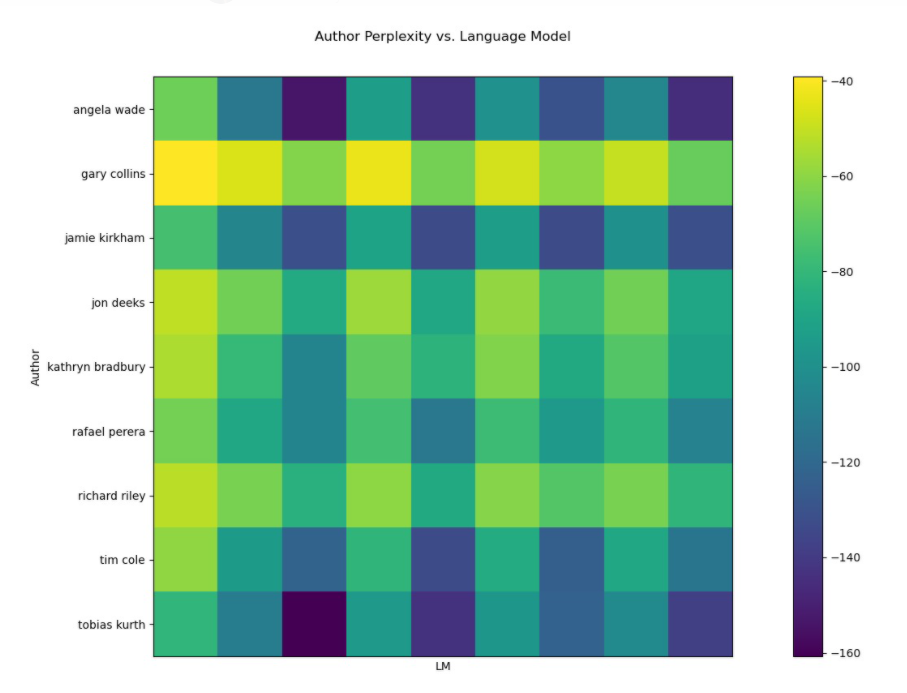
ובאופן כללי אשכול של המודל הנוכחי (אין קשר בין הצבעים בין גרפים שונים):

* בדיקה על בעיה פשוטה יותר של אימון על reviews במקום על מאמרים – ורק אז ממשיכים ל-cross domain.
* יצירת Language Models:
  + למדנו מה זה מודל שפה ואיך יוצרים את זה, וגם מה זה LSTM.
  + המטרה – ליצור לכל כותב LM בהתבסס על הטקסטים שהוא כתב (מאמרים). להשתמש בזה כדי לחשב perplexity של טקסט מסוים לפי כל LM ולהכניס את זה כ-feature.
  + הרעיון – ה-perplexity יהיה הכי קטן עבור המודל של הכותב שהכי מתאים לטקסט הזה.
  + מה עשינו:
  + מודול vocabulary: שומר מילון שממיר ממילה למספר עבור כותב מסוים. משתמשים בזה כדי להפוך טקסט שכתוב כמילים לאות מספרי.
  + מודול LanguageModelNN: מודל שפה שמורכב משכבת embedding, שכבת LSTM, ושכבת FC (אולי ישתנה בהמשך). מממש את הרשת שמגדירה את ה-LM, כולל פונקציית forward.
  + מודול dataset שמאפשר לטעון בצורה נוחה את הטקסטים כ-batches של tokens.
  + פונקציית אימון ופונקציה שמגנרטת משפטים לפי ה-LM (בעיקר כדי לבדוק שהמודל למד בצורה הגיונית).
  + מודול LanguageModel שמאחד הכל (LanguageModelNN + vocabulary) ומאפשר שימוש נוח כדי להוציא features: מכיל פונקציית שמירה וטעינה של המודל + פונקציית חישוב perplexity – עבור טקסט מסוים מזינים את הטקסט לרשת ב-sliding window, ומקבלים וקטור הסתבורויות למילה הבאה, עבור המילה הבאה האמיתית לוקחים את ההסתברות וככה מחשבים perplexity על כל הטקסט. הנוסחא:
  + הכנסנו את המודול הזה כחלק מהמודול של Feature extraction והוצאנו פיצ'רים.
* יצירת Language Models עם GPT2:
  + בחירת המודל המתאים מ-Huggingface.
  + למידת ה-API.
  + הכנת הדאטה – איחוד כל הטקסטים של כל כותב למסמך יחיד.
  + כתיבת קוד לאימון מודל שפה לכל כותב: יצירת tokenizer, שימוש במודל GPT2 מאומן וביצוע fine-tuning לכל כותב.
  + כתיבת פונקציה לחישוב perplexity.
  + שימוש במודלי שפה ובפונקציה כדי להוציא features – perplexity לפי כל מודל שפה.
  + הרצת המודל על ה-toy problem כדי לראות שהתוצאות הגיוניות.
  + הצגה של סוג של confusion matrix שמציגה perplexity של כל סופר לפי כל מודל שפה – כדי לבדוק אם ה-feature החדש שימושי (גם למאמרים וגם ל-reviews).

LM שאומן על מאמרים, ונבדק על מאמרים:



LM שאומן על מאמרים ונבדק על reviews:



* הוספנו class weighting ל-SVM.
* כדי לבדוק עוד: הרצנו את הבדיקה לא ב-cross-domain: פעם אחת אימון על מאמרים ובדיקה על מאמרים, ופעם אחת אימון על reviews ובדיקה על reviews.
* ניקינו קצת את הדאטה: העפנו שני כותבים שהיו להם רק 5 reviews לכל אחד, כי הם קטנים מדי בדאטהסט. הורדנו reviews שהיו עם פחות מ-30 מילים כי אין שם מספיק בשר לסיווג. ניסינו גם להפוך את אורכי הטקסטים שאיתם אנחנו מאמנים את ה-LM לאחידים עבור כל הכותבים, כדי שלא יהיה מישהו שמתאמן יותר מאחרים.
* ניסינו גם לגרום לכך שסך כל הטקסטים של כל כותב יהיו באותו גודל, בעצם כדי שלא יהיה מצב שחלק מתאמנים על יותר טקסטים וחלק על פחות. בסוף זה הרס את התוצאות אז זנחנו את זה. (ההסבר שלנו: כי הורדנו דאטה וגם ככה הדאטהסט קטן).
* הוספנו אלגוריתם אשכול כדי למצוא outliers בסט האימון ולהתעלם מהם. השתמשנו ב-OPTICS (הרחבה של DBSCAN). המטרה היא בעצם לזהות דגימות שהן שונות מדי מהפילוג האמיתי של המידע כדי שלא "יבלבלו" את המסווג. עשינו את זה בגלל ההבנה שיש הרבה מאמרים שלא בהכרח בן אדם כתב, ואנחנו לא רוצים שהם יופיעו בסט האימון.
* ניסינו גם לבדוק הכנסת argmin על ה-ppl כפיצ'ר, במקום להכניס את כל ה-ppl כפיצ'ר וקטורי.
* בבדיקה ה-argmin כפיצ'ר, בדקנו את השמת ערך ה-perplexity ושינינו – לא לקחנו את המינימום בין הפרפלקסיטי ל5000 אלא הצבנו כמו שזה. כמו כן, הורדנו את ה-stride ל-32 וכך לא קבלנו יותר inf כערכים של הפרפלקסיטי על reviews.

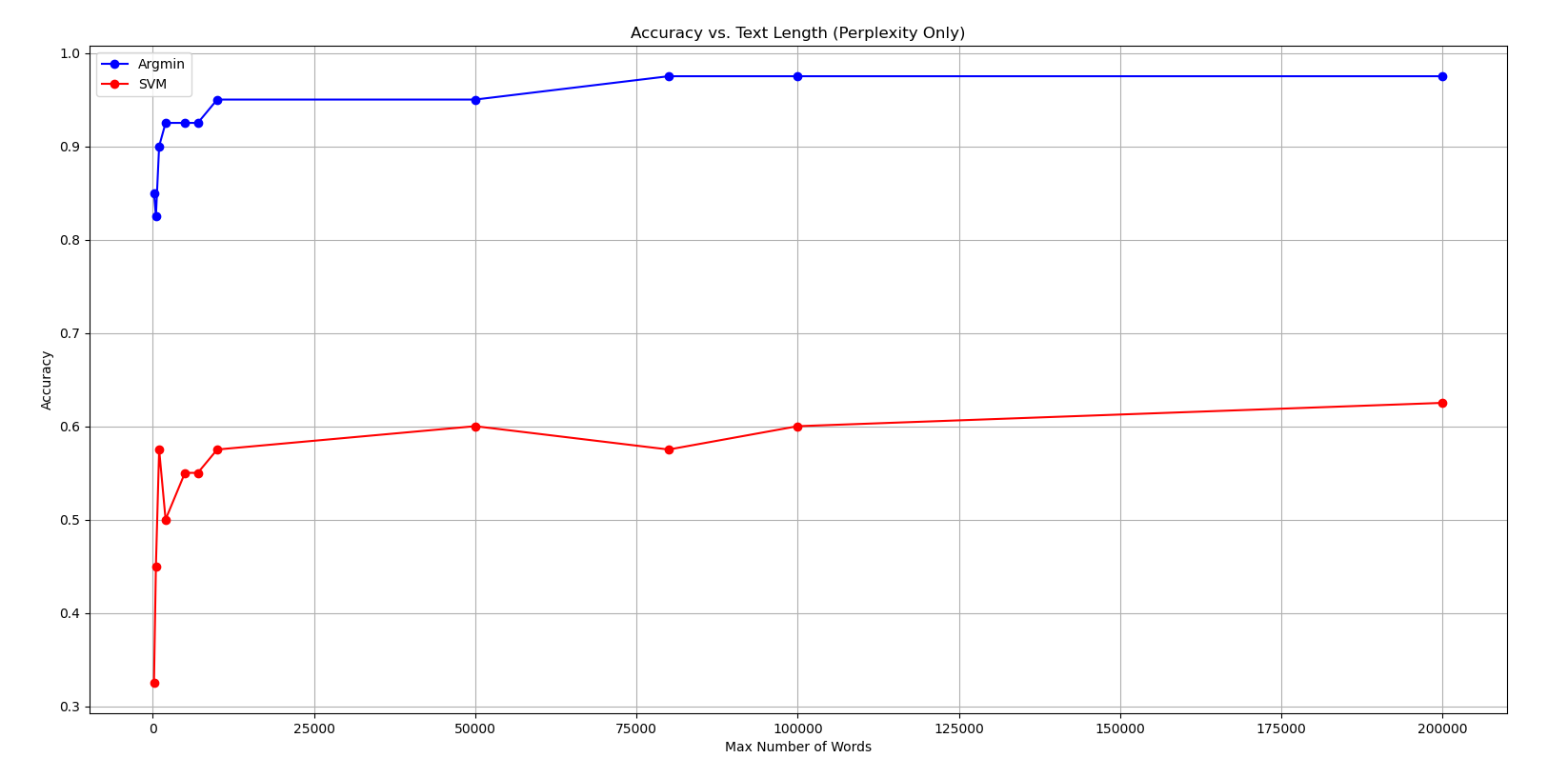
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **9 מחלקות** | | **7 מחלקות** | | |
|  | **בלי ppl** | **עם ppl** | **עם ppl** | **עם ppl + בלי חסם 5000** | **argmin על ppl כפיצ'ר במקום וקטור של כל ה-ppl** |
| **reviews-reviews** | 70% | 58% | 68% | 62% | 70% |
| **מאמרים-מאמרים** | 74% | 66% | 62% | 57% | 28.50% |
| **מאמרים-reviews** | 24% | 30% חזה על הכל richard חוץ מ-1 | 27% חזה על הרוב richard | 28.50% חזה על הרוב richard | 29.70% חזה על הרוב richard או jon |

Mean number of tokens: 527.6011904761905

STD number of tokens: 362.0160503525617

Median number of tokens: 458.0

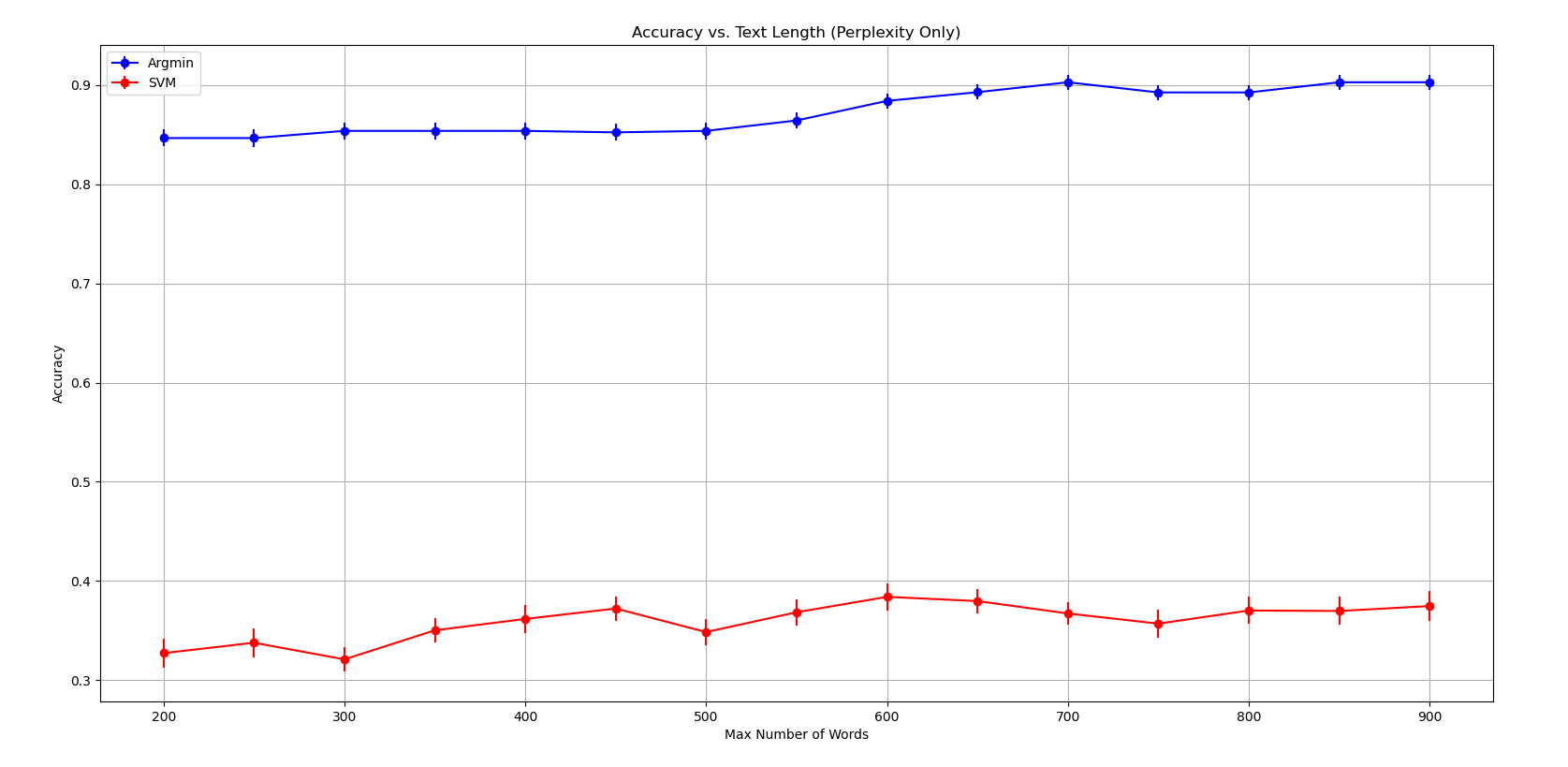
על ה-toy problem הרצנו סקריפט שמבצע חיזוי רק באמצעות ppl בשתי דרכים: באמצעות argmin על ה-ppl ובאמצעות אימון של SVM. בדקנו את ההשפעה של אורכי הטקסטים על הדיוק:



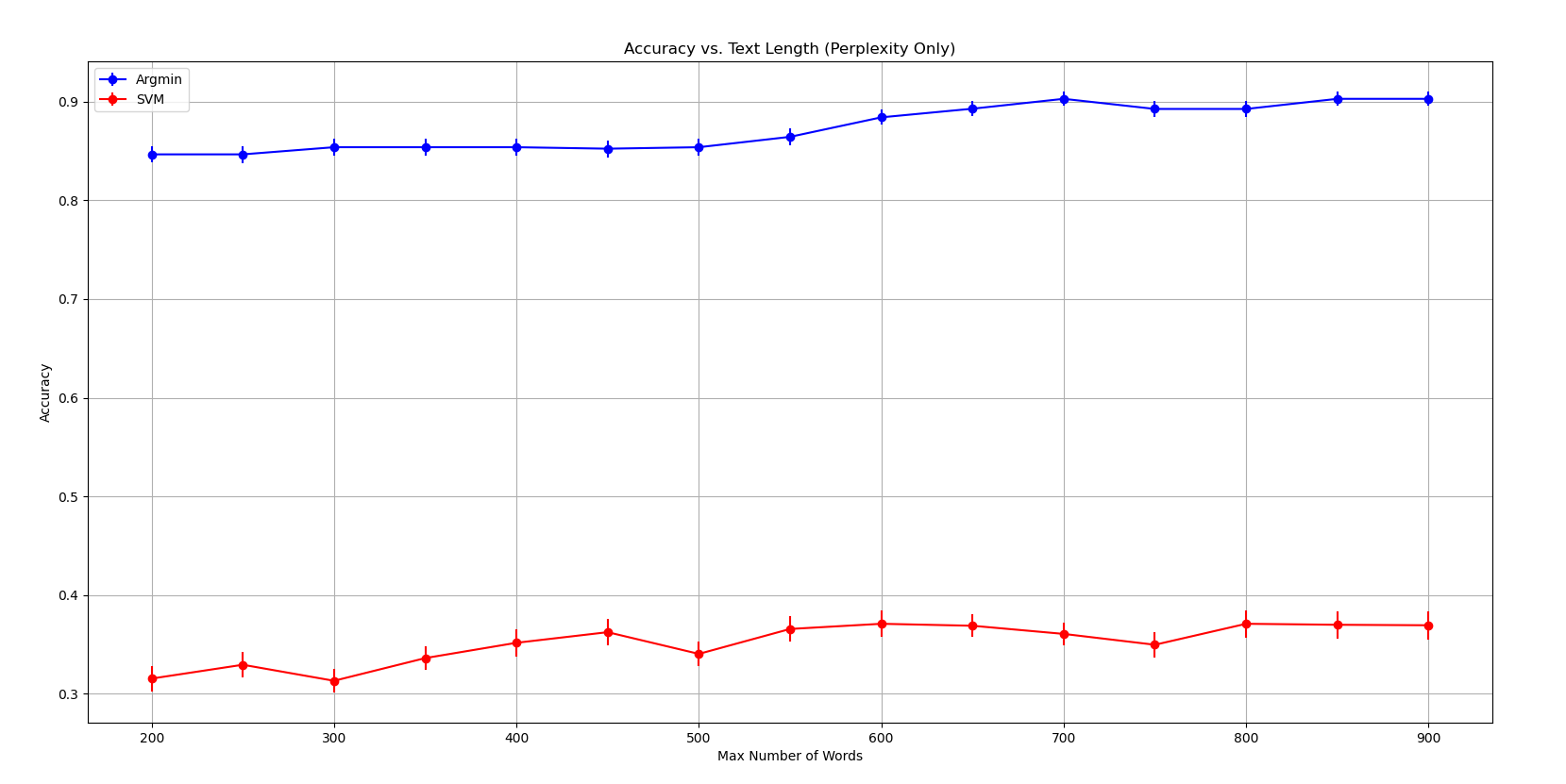
המסקנות הן ש-argmin עובד יותר טוב, שאורכי הטקסטים משפיעים (כמובן), ושמגיעים ל-plato כבר באזור 10000 מילים.

הוספנו גם דגימת bootstrapping כדי לקבל סטיית תקן וממוצע ולהציג error bars (100 דגימות).

בדקנו מה אורך ה-review הממוצע (וסטיית תקן), והגדרנו את זה כנקודת עבודה. עשינו את הבדיקה הנ"ל סביב נקודת העבודה הזו כדי לדמות איך זה יעבוד על reviews:

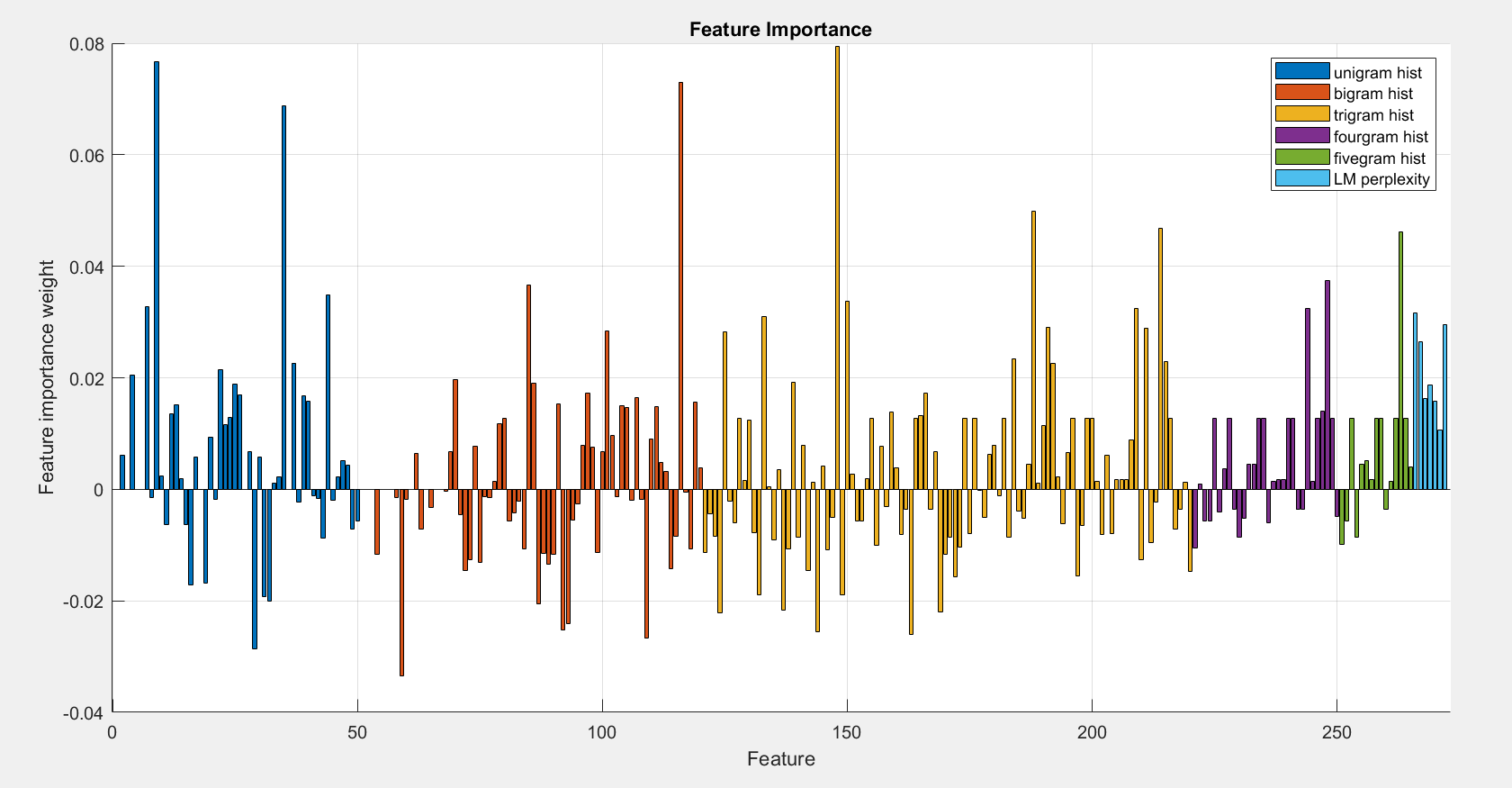


ניסינו גם את אותה ריצה רק בלי לעשות scaling לפני ה-SVM, כי פחדנו שה-scaling גורם לזה שהיחסים בין ה-ppl של טקסט מסוים נהרסים. בפועל התוצאות מראות שזה לא משנה:

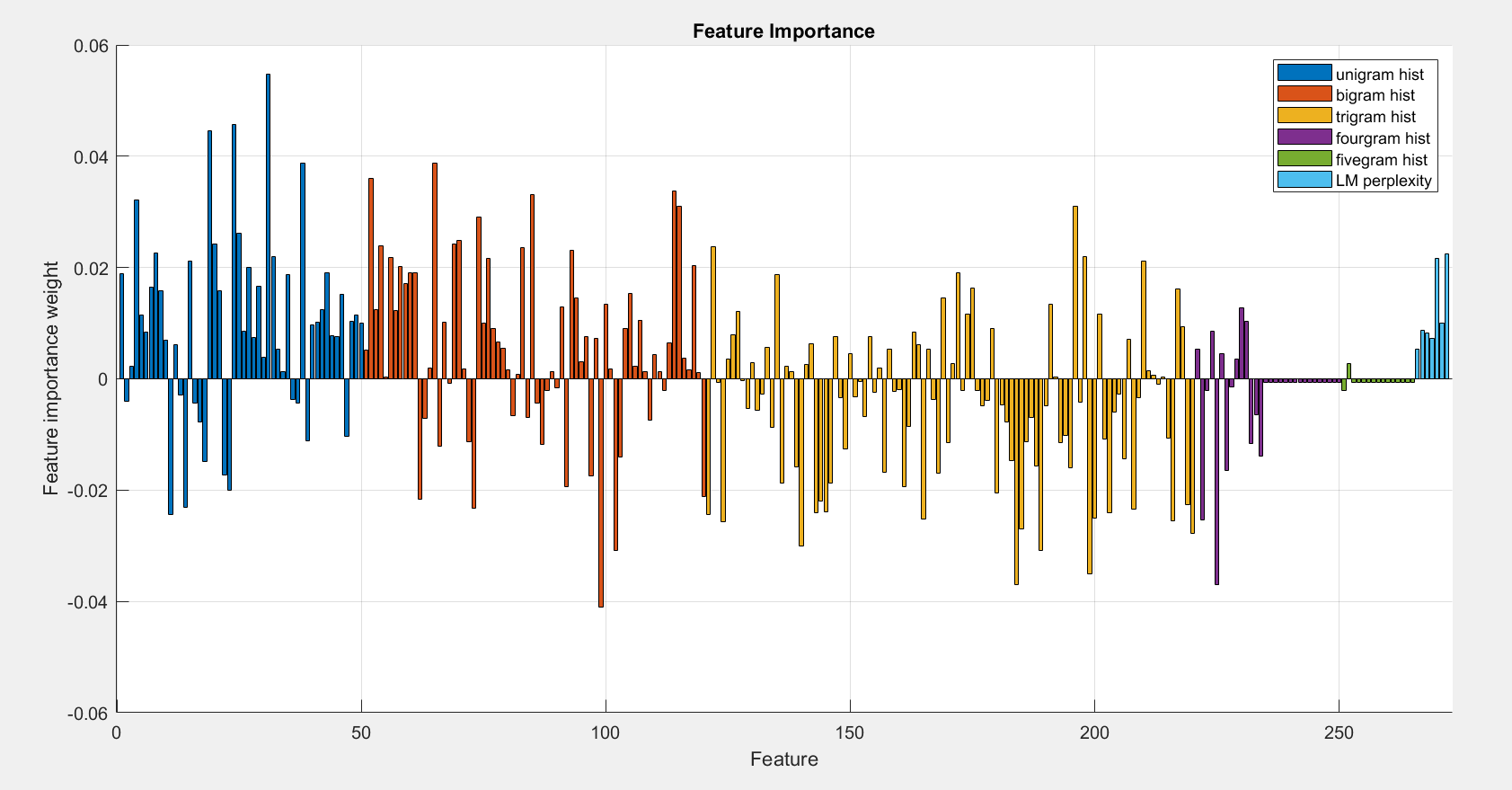


אלגוריתם relief למציאת feature importance (במטלב):

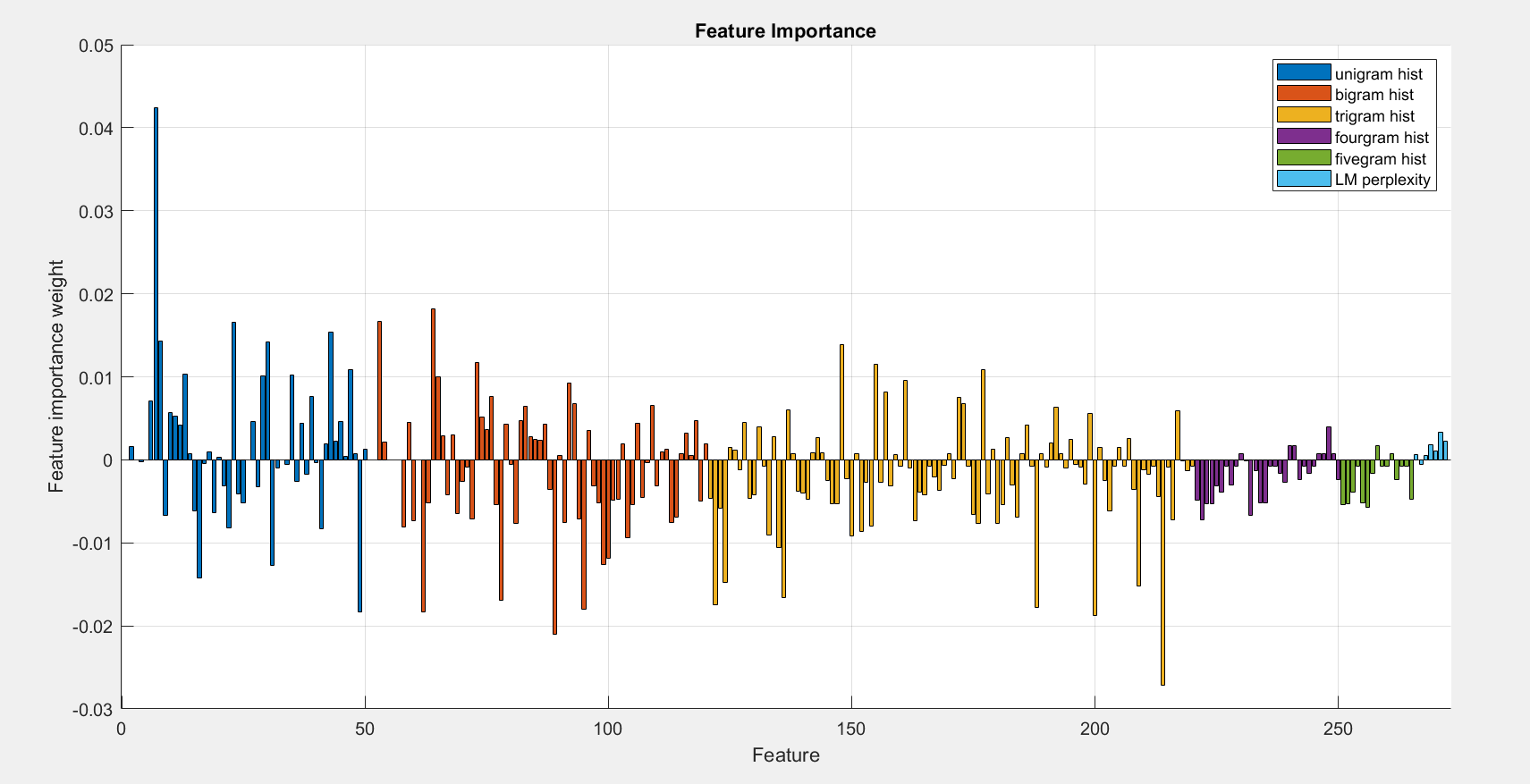
* על מאמרים-מאמרים:



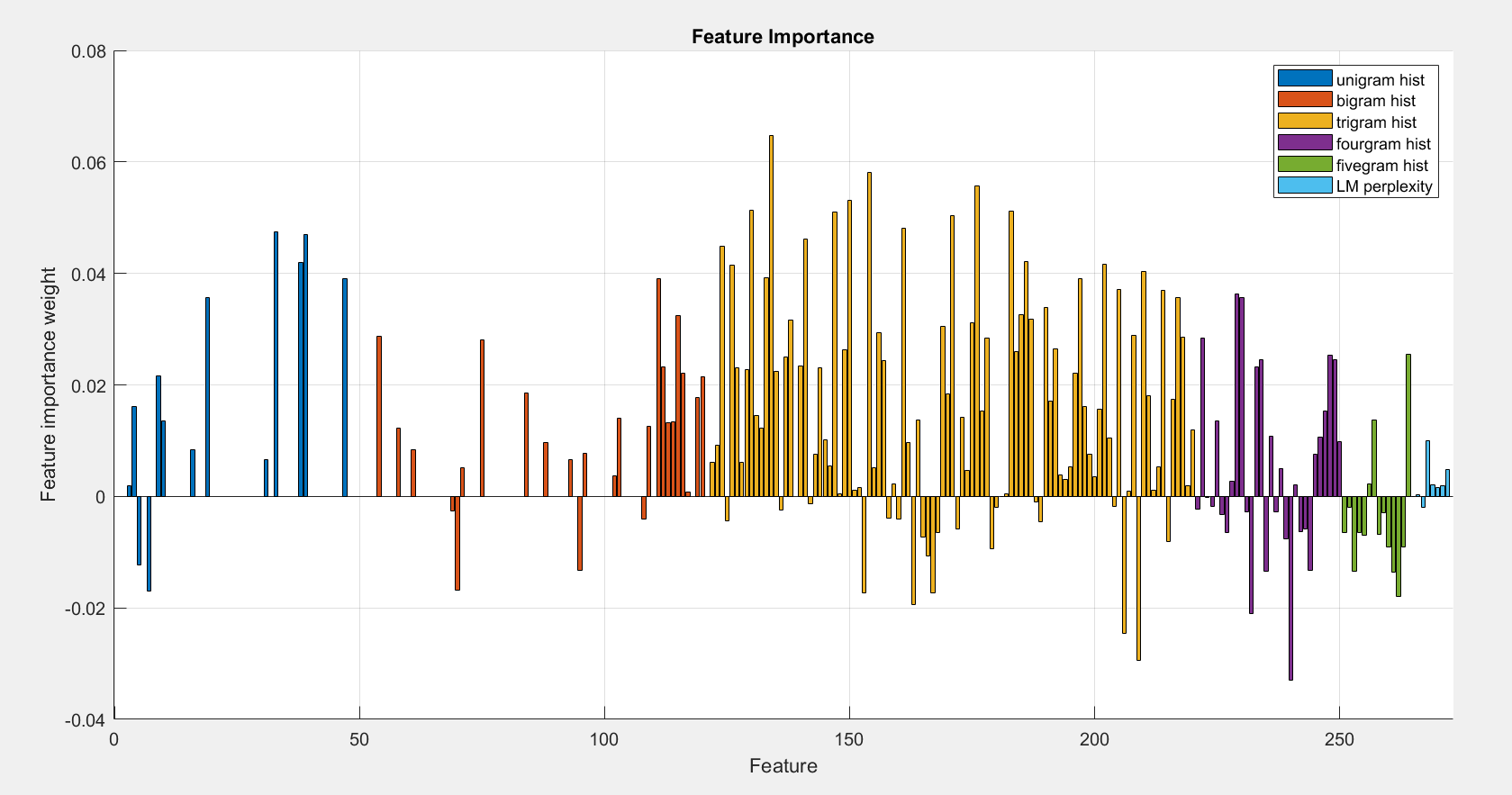
* על reviews-reviews:



* על מאמרים-reviews:



* על toy-problem:

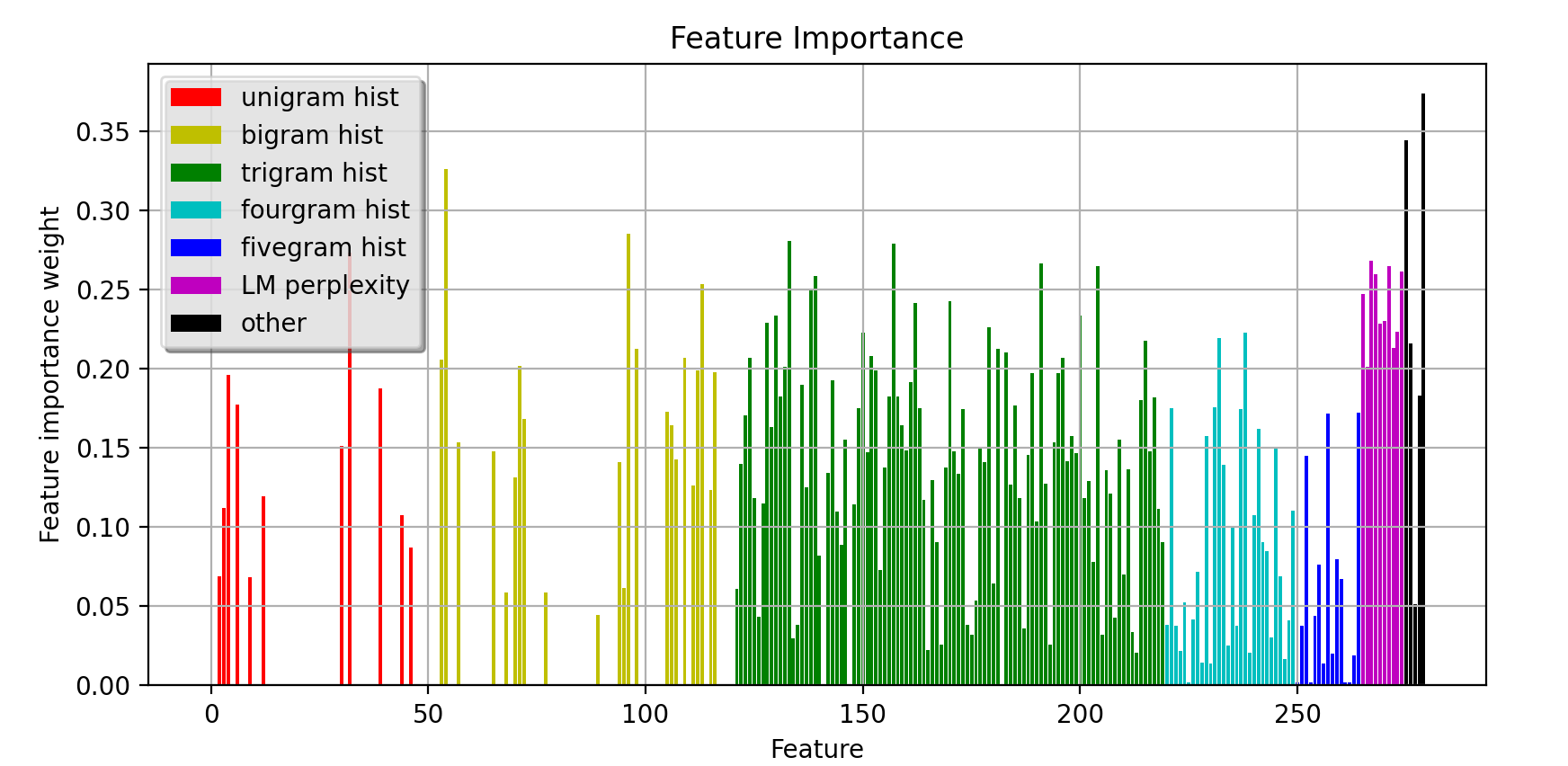


* הכנסנו את ה-feature selector כחלק מה-pipeline על ה-train set:
  + toy problem – 70%
  + מאמרים-מאמרים – בלי: 66.6%, עם 0.5: 66.6%, עם 0.67: 71.4%
  + reviews-reviews – בלי: 60.3%, עם 0.5: 58.5%, עם 0.67: 62.2%
  + מאמרים-reviews – בלי: 26.7%, עם כל דבר מתחת ל-0.9: +-17% (חוזה gary collins על הרוב)

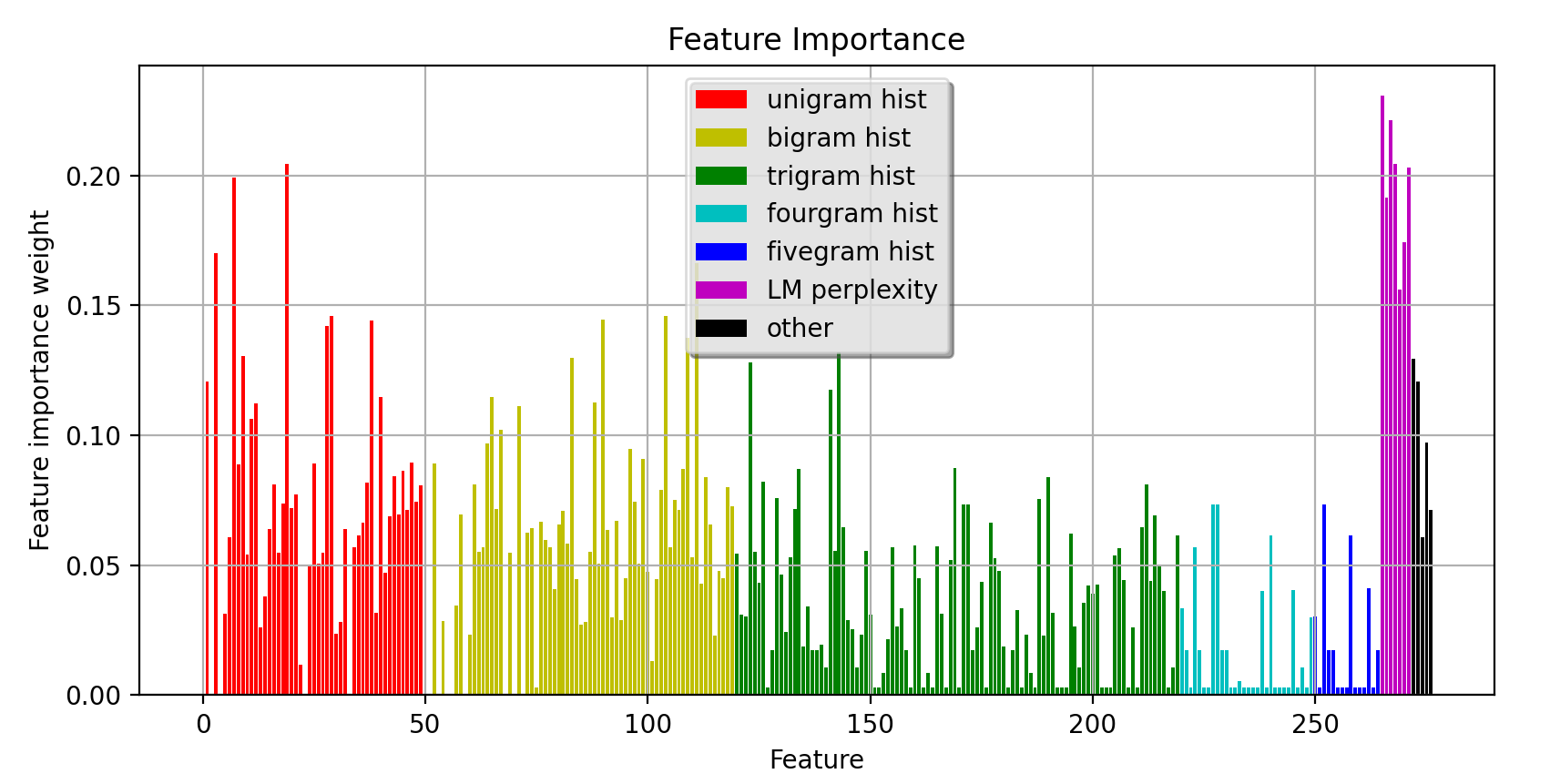
הוספנו פיצ'רים קטנים נוספים: היסטוגרמה של סימני פיסוק, אורך מילה ממוצע, מספר מילים ממוצע במשפט.

feature selection עם פיצ'רים חדשים:

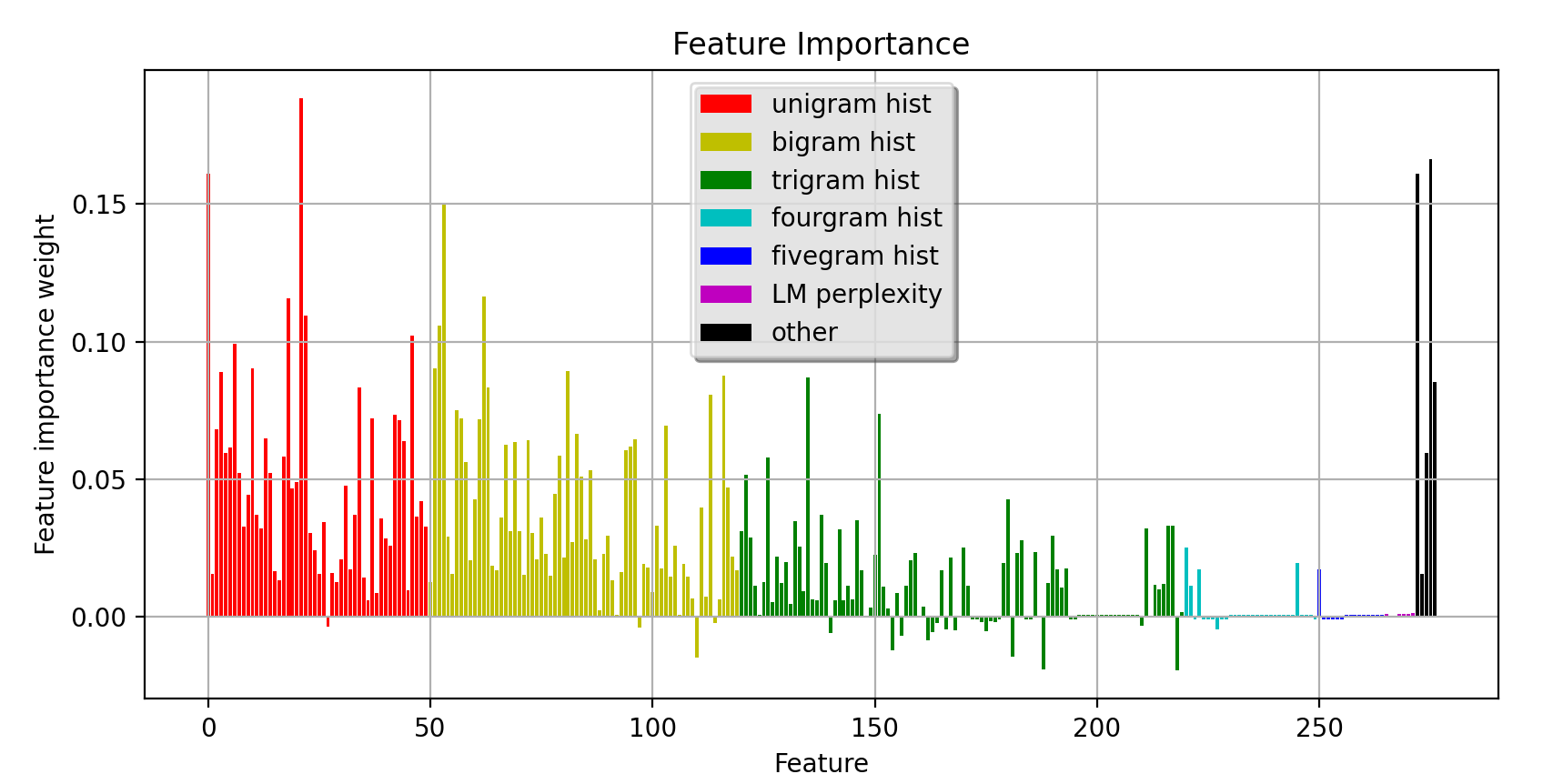
* טוי פרובלם:



* מאמרים-מאמרים:



* reviews-reviews:



הוספנו פיצ'רים נוספים: היסטוגרמה של סימני פיסוק, אורך מילה ממוצעת, אורך משפט ממוצע (במילים).

תוצאות אחרי הוספת הפיצ'רים + feature selection:

* טוי פרובלם: בלי – 76.7%, עם כל דילול בין 0.5 ל-0.8 – 76.7%
* מאמרים-מאמרים: בלי – 52.3%, עם דילול 0.7-0.8 – 57.1%
* reviews-reviews: בלי – 69.8%, עם דילול 0.5 – 73.5%
* מאמרים-reviews: בלי – 25.5% (כמעט הכל RR כמובן), עם כל דילול - +- 17%

הערה: לשים לב שלפי הגרפים דווקא נראה שבמאמרים-מאמרים הפיצ'רים החדשים אמורים ממש לעזור, אבל בפועל בתוצאות נראה שהם מורידים אחוזי הצלחה.

* עשינו בדיקה של relief והכנסנו את הדגימות בסדר שונה כמה פעמים כדי לוודא שיוצאות אותן תוצאות. אכן יצאו אותן תוצאות וזה אומר שאין תלות בסדר הדגימות.
* הוספנו שוב cross validation (עם feature selection):

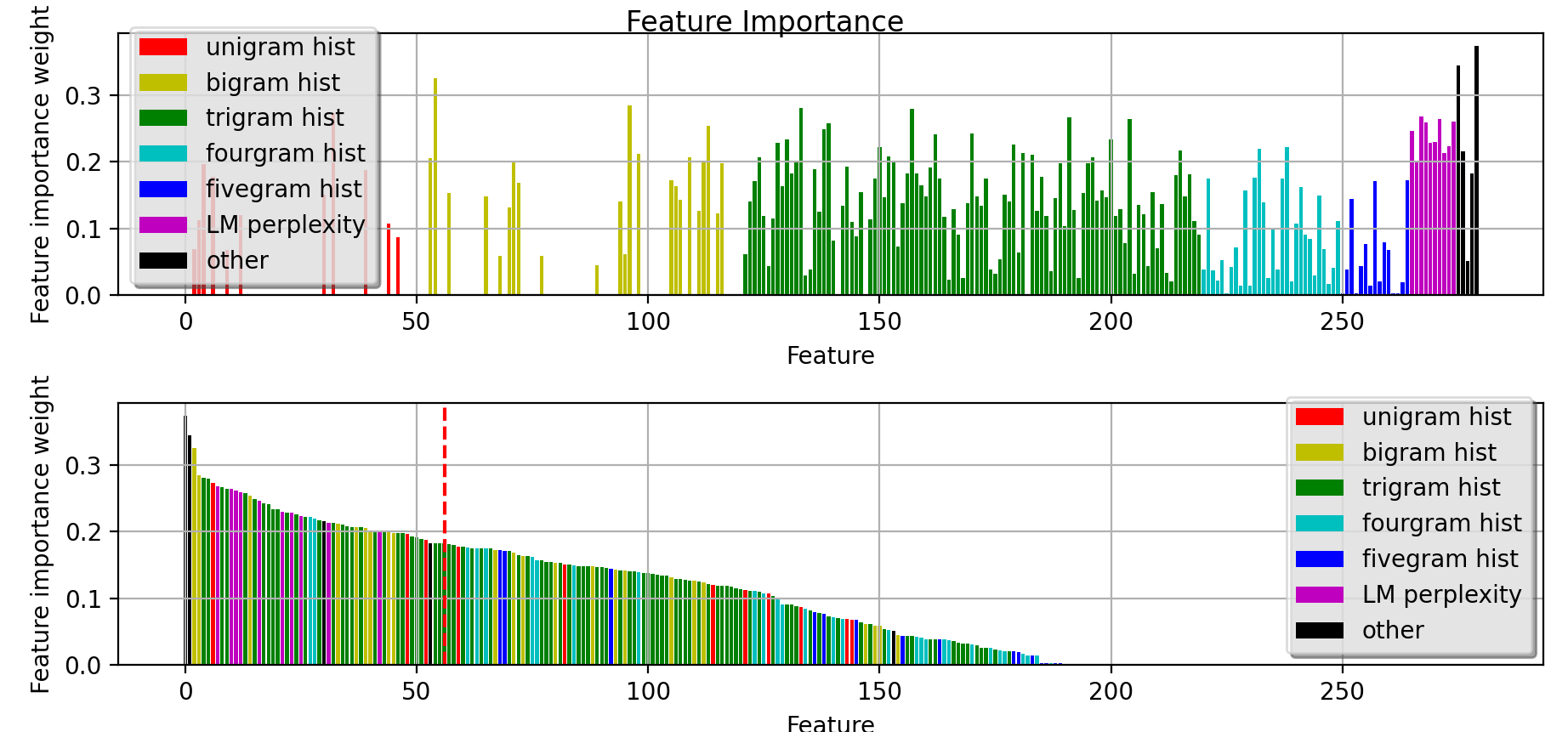
טוי פרובלם – עם 0.2 – 87%

מאמרים-מאמרים – עם 0.5 – 77.1%

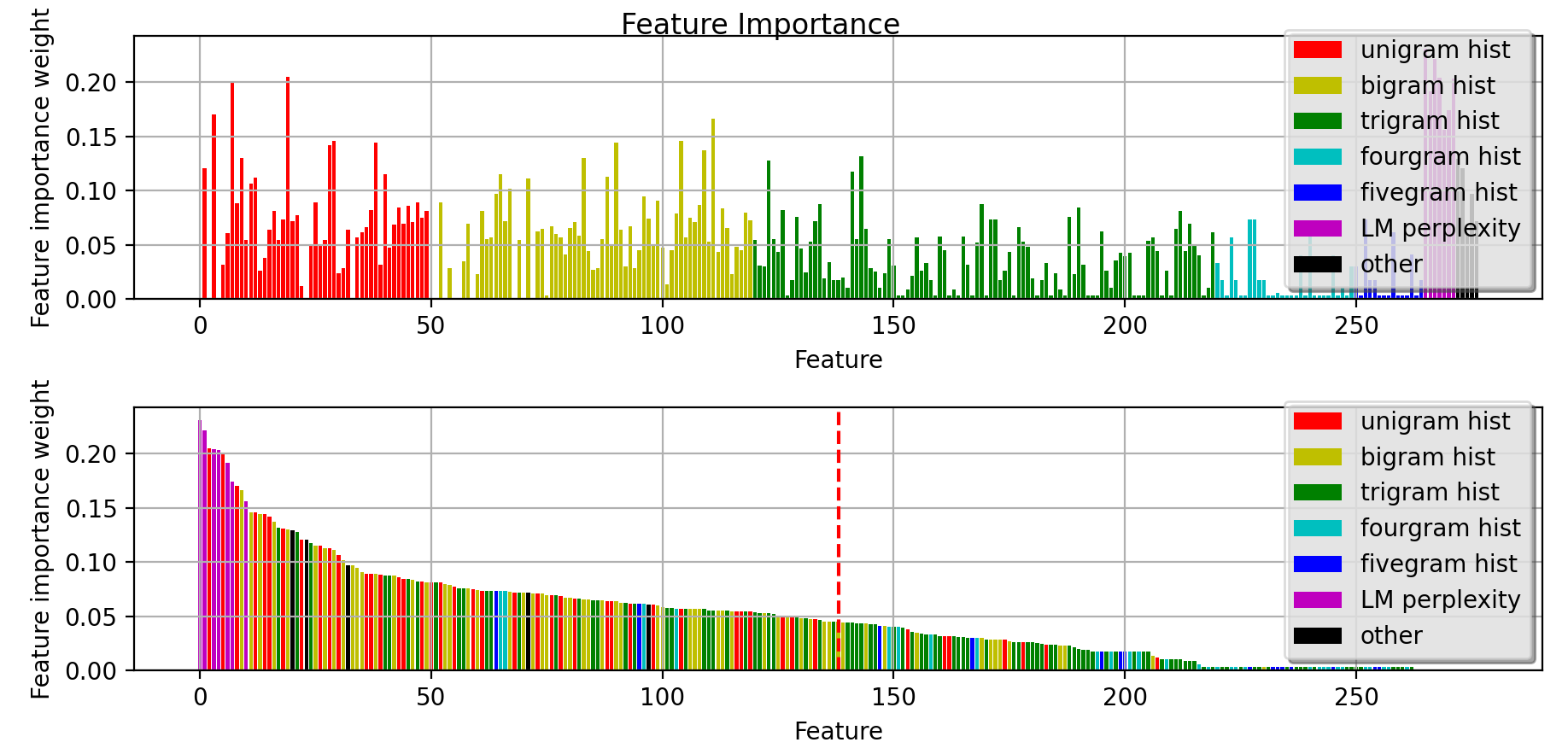
reviews-reviews – עם 0.3 דילול – 76.9%

הוספנו גם הצגה אחרת של ה-feature selection שמראה יותר טוב מה הפיצ'רים שנבחרו:

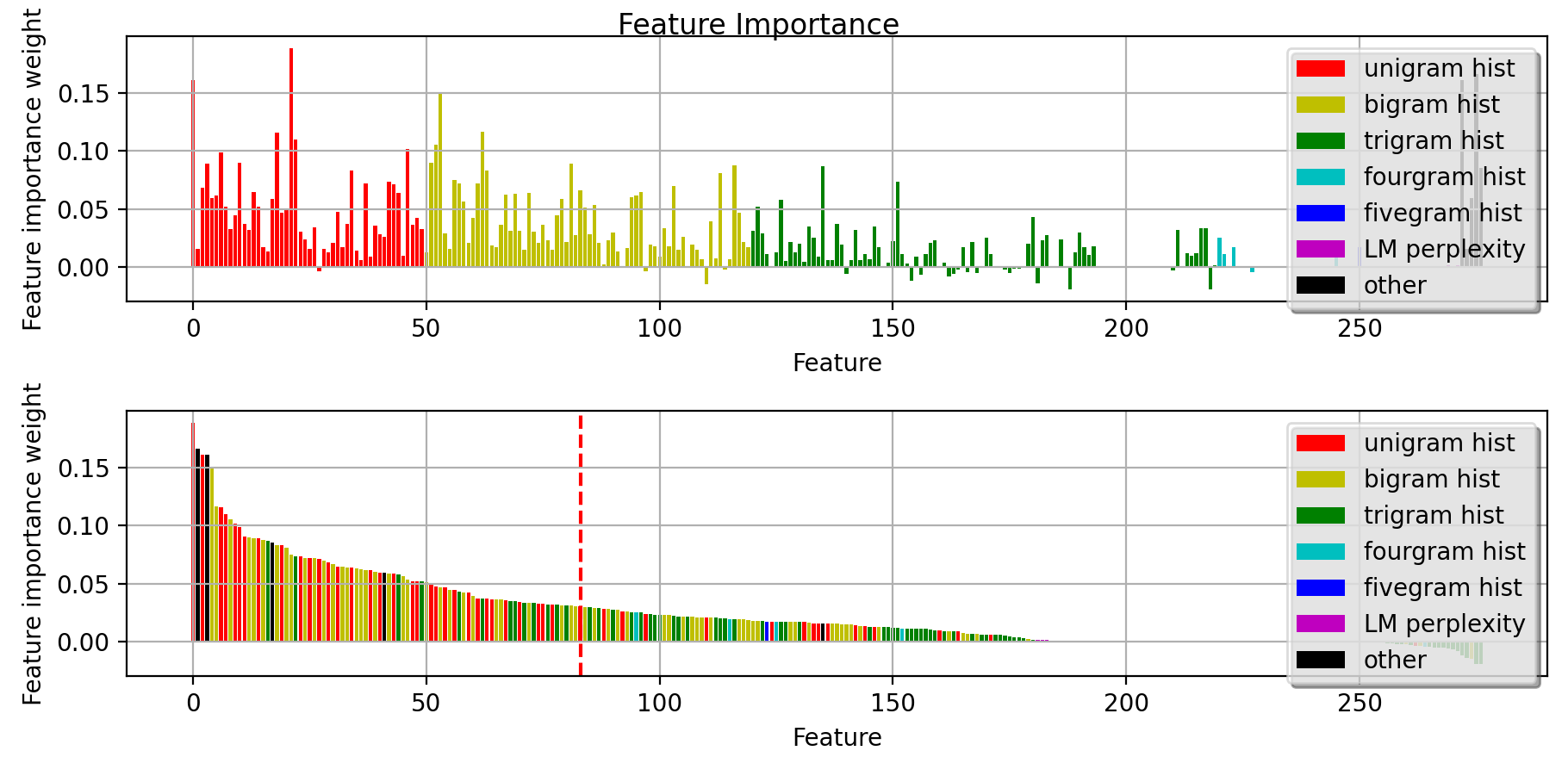
toy-problem:



מאמרים-מאמרים:



reviews-reviews:



* רק ב-reviews-reviews ה-ppl לא נכנס ב-feature selection, זה אומר שבאחרים יש מצב שקצת רימינו ב-CV (כי משתמשים כ-test set באותם טקסטים ששומשו לאימון ה-LM).
* שינינו את ה-kernel של ה-SVM כדי לנסות לתפוס מרחבי מטריקות שונים. הגרעין שהביא לתוצאות הכי טובות היה sigmoid:

toy problem – 83.3%, ו-87% ב-CV

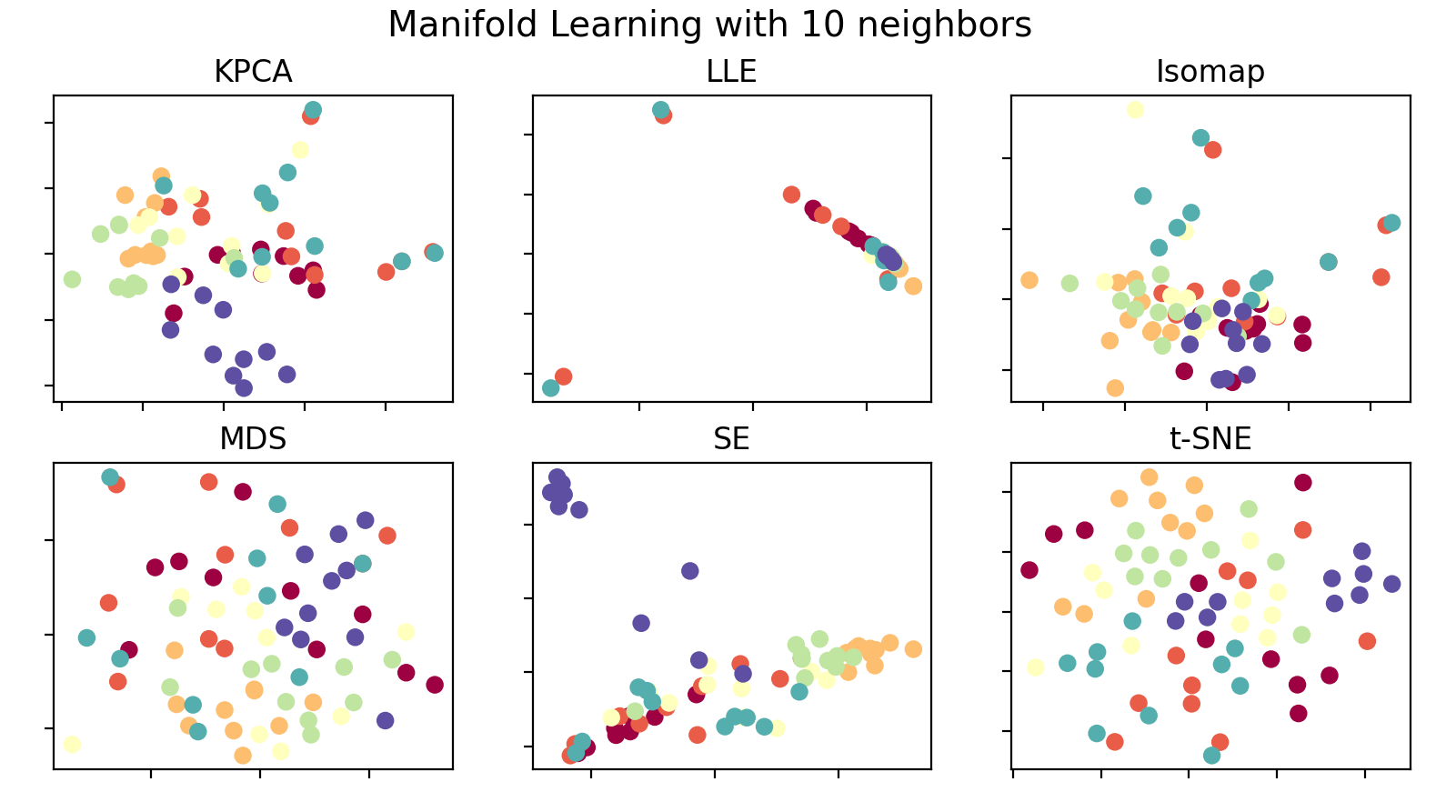
מאמרים-מאמרים – 76.2%, ו-78.5% ב-CV

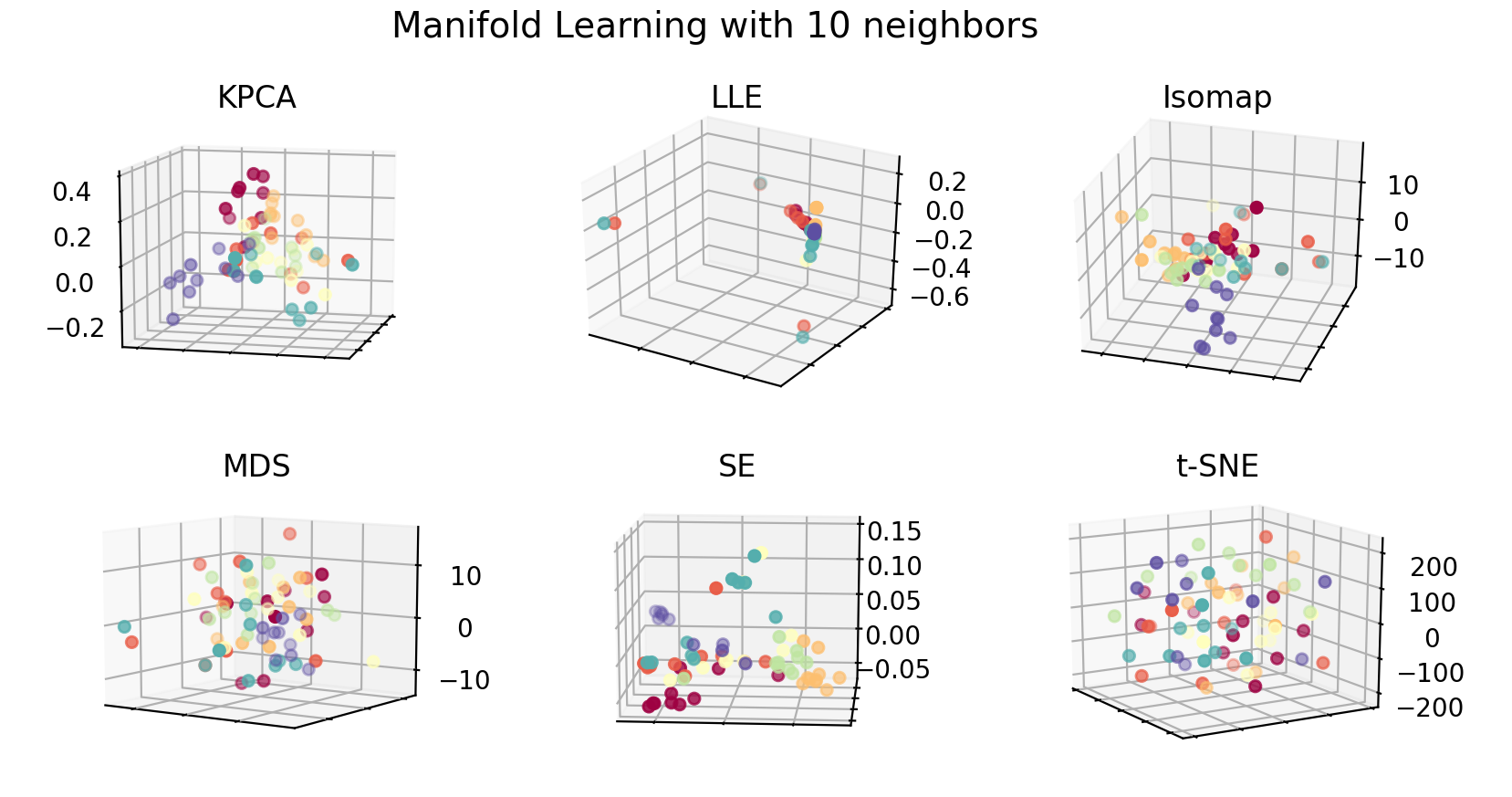
reviews-reviews – 83%, ו-80.2% ב-CV (הצליח להגיע ל-84% עם גרעין לינארי, אבל הגרעין הזה הביא לתוצאות פחות טובות במאמרים-מאמרים)

עוד הערה: התוצאות הנ"ל הן עבור דילול פיצ'רים של 0.5 לכולם.

הוספנו manifold learning כדי לבצע המרה ממרחב הפיצ'רים למרחב שיותר קל ללמוד עליו.

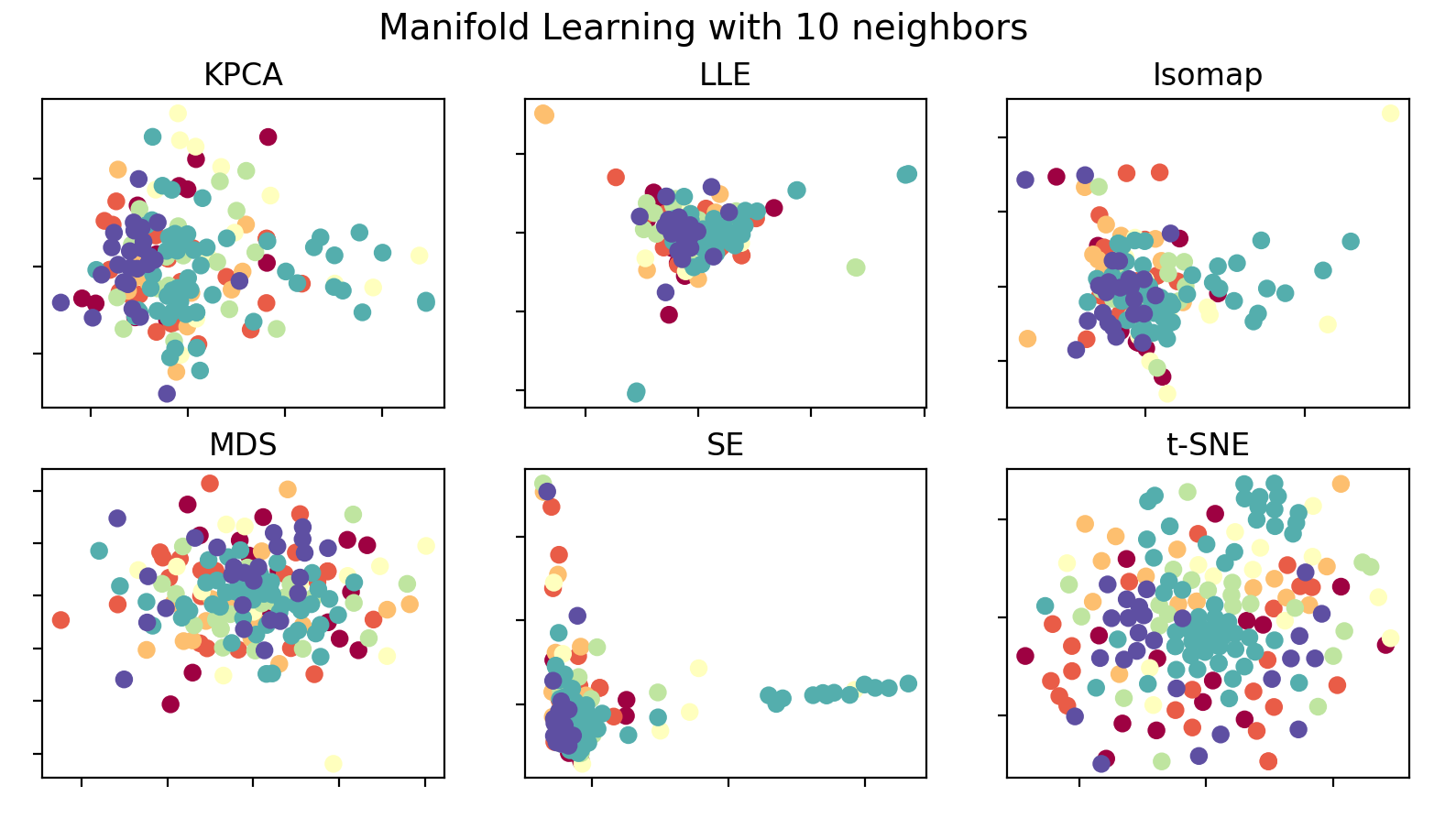
מאמרים:

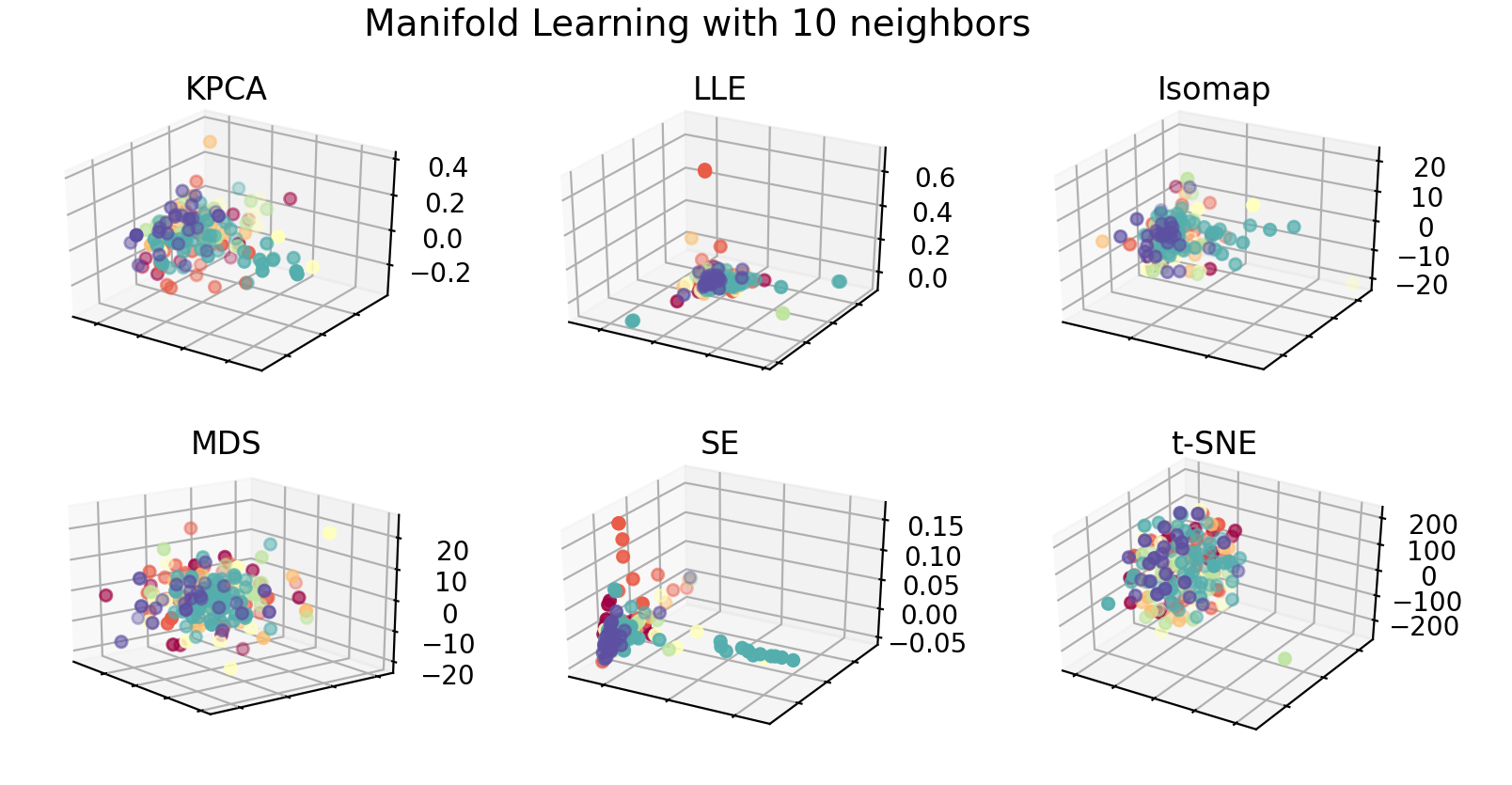




המסקנות כרגע הן ש-KPCA (עם sigmoid) נותן הפרדה יחסית טובה בין ה-classes. tSNE נותן פיזור טוב אבל נראה שההפרדה לא מספיק טובה. גם Isomap נראה די טוב.

reviews:





שיחה עם פבל:

* פיצ'רים שאפשר לבדוק: אוצר מילים (כמות מילים שונות), שגיאות כתיב (לבדוק אם יש משהו של NLTK), character n-grams.
* parallel transport להמרה ממרחב המאמרים למרחב ה-reviews