Course-ID:	MA-INF 4306
Course:	Lab Development and Application of Data Mining and Learning Systems: Data Science and Big Data
Term:	Winter 22-23
Supervisor(s):	Dr. Jörg Kindermann
Students:	Behnam Fanitabasi: 42424242

Clustering on text embeddings with various distance measures

Behnam Fanitabasi

August 30, 2023

Abstract

Neural natural language models are drawing more and more attention every day as they aim to give computers the ability to interpret human words and documents. Yet as there are more and more models available, it gets more challenging to select the optimal one for a certain task. So, in this project, we look for several features and evaluation methods that will aid in selecting a model. We begin the report by defining the terminology and approaches we employ for the project. We then discuss in detail the main idea and steps we implement. Afterward, we bring the plots and the results of our experiments. Finally, we conclude with discussions of which features have potential and could serve as building blocks for future developments.

Contents

List of Figures

1 Introduction

Natural language processing (NLP) is a branch of computer science, specifically artificial intelligence, which employs various techniques to help computers comprehend texts and speeches as people do. Neural language models are an approach that represents words and texts as vector embeddings in order to use the computing power of computers for various tasks involving datasets of words and texts, such as clustering documents according to their topics. Large language models are often trained on very large amounts of text data using different unsupervised methods. The performance of the machine learning (ML) model trained with these embeddings for a specific task may alter if multiple embeddings of the same dataset obtained by different language models are used.
Due to the nature of these datasets, attempting to obtain different embeddings and train ML models on them to choose the best one is memory and time insufficient. The solution to this issue is to develop a scoring method for neural language models based on the embedding space and derive an evaluation measure based on multilabel text categories. We look for numerous features to determine which of them can assist us in better distinguishing the embeddings given three alternative embeddings of the same dataset (Amazon Cat-13k (Bhatia et al., 2016)). We investigate, plot, and compare different features are statistical features such as the mean of cos-similarities and the number of neighbors in the nearest neighbor graphs created by the embeddings and structural features such as chosen subgraphs and dense clusters of the KNN graphs.

2 Definition

2.1 Neural Natural Language Models and Vector Embedding

Neural natural language models are a type of machine learning model used for natural language processing (NLP) tasks. These models are based on neural networks and are used to represent words or sentences as dense, high-dimensional vectors which are called word embeddings and sentence embeddings. Word embeddings are vector representations of words that capture the semantic and syntactic relationships between them and sentence embeddings are built by combining the word embeddings. Similar words and texts should be clustered together in the embedding space in order to achieve the purpose of these embeddings. Word2Vec (Mikolov et al., 2013) and Glove (Pennington, Socher, and Manning, 2014) are examples of famous embedding algorithms. These embeddings can be used in tasks such as question answering, text classification, and information retrieval. For instance, *Chatgpt* 2022 is a chatbot developed by OpenAI and was trained on *GPT-3.5* 2022 which is a generative language model that uses a transformer architecture to generate high-quality natural language text. BERT (Devlin et al., 2019) is another example of the neural language model.
In this project, we are using vector embeddings of Amazon descriptions of different products to classify them.

2.2 Amazon Cat-13k

The dataset we are using is called Amazon Cat-13k (Bhatia et al., 2016). It has 13,330 product labels and 1,186,239 product descriptions. We select a subset of this dataset with

5000 documents, around 3500 labels, and around 85000 unique words and numbers. For the purposes of this project, we employ three different text embeddings:

- Vectors computed on the original neural network from HuggingFace (Orig (Beltagy, Peters, and Cohan, 2020))
- Vectors computed on a retrained network with original vocabulary (Refined)
- Vectors computed on a retrained network with new vocabulary (New)

Where the dimension of each embedding for each text is 768.

2.3 Term Frequency–Inverse Document Frequency

TF-IDF, or Term Frequency-Inverse Document Frequency, is a widely used method in information retrieval and natural language processing. It is a statistical technique that measures the relevance of a word in a document, with respect to a larger corpus of documents. The relevance score of a word is calculated by multiplying its term frequency (number of times it appears in a document) by its inverse document frequency (logarithm of the number of documents in the corpus divided by the number of documents containing the word). High TF-IDF scores indicate words that are unique and informative to a document. This method has numerous applications, including search engine ranking, text classification, and document clustering.

$$TF - IDF(t, d) = TF(t, d) \cdot IDF(t)$$

$$TF(t, d) = \frac{f_{t,d}}{\sum_{k=1}^{n} f_{k,d}}$$

$$IDF(t) = \log \frac{N}{n_t}$$

Here, $f_{t,d}$ is the frequency of word "t" in document "d", n is the total number of words in document "d", N is the total number of documents in the corpus "C", and n_t is the number of documents in the corpus that contain the word "t".

2.4 Frequent Itemset

Frequent Itemset is a collection of items that appear frequently together in a dataset. These itemsets are used in association rule mining, which is a technique used to uncover hidden relationships and patterns between items in a dataset. The dataset for our project is the collection of documents, and the items are the document categories.
The Apriori algorithm (Agrawal and Srikant, 1994) and the FP-Growth algorithm (Han, Pei, and Yin, 2000) are two popular algorithms for finding frequent itemsets in large datasets. The Apriori algorithm works by considering itemsets of increasing size, generating candidates, and using the frequent itemsets found at each step to prune the search space for the next step. On the other hand, the FP-Growth algorithm builds a compact data structure called a frequent pattern tree which is used to directly identify the frequent itemsets. The FP-Growth algorithm is more efficient than the Apriori algorithm in terms of memory usage and speed. We employ the FP-Growth algorithm because we are dealing with a large number of documents and categories.

2.5 Nearest Neighbor and Recursive Nearest Neighbor Graph

The nearest neighbor (KNN) (Eppstein, Paterson, and Yao, 1997) graph is a graph where each data point is connected to its k-nearest neighbors in a high-dimensional space. The high-dimensional space is typically represented using vector embeddings. Vector embeddings and nearest neighbor graphs are often used together in machine learning and data mining tasks to find similar items or data points. Vector embeddings can be used to create a nearest neighbor graph by placing them as nodes and connecting them to their k closest neighbors using a similarity metric like cosine similarity, which calculates the cosine of the angle between two vectors to determine which direction the vectors are pointing in. Recursive KNN graphs are created by using the KNN method recursively and finding the nearest neighbors of the neighbors of the original node and continuing the same process. A parameter that we call *nHops* in this project defines the number of times that this process is repeated (including the first KNN). The algorithm for creating a Recursive KNN consists of two parts which are creating a KNN (Algorithm 1) and recursively creating Recursive KNN (Algorithm 2)

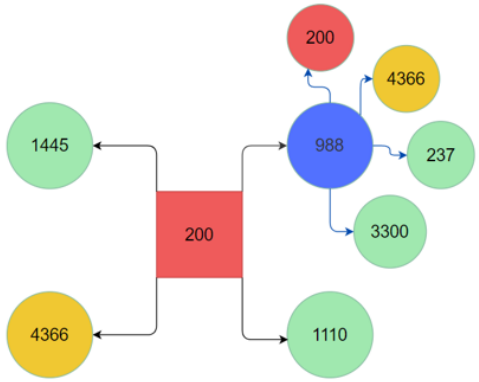


Figure 1: Example of a node in Recursive KNN with K=4
Each node is a document with the number as an identifier. Some Recursive KNN graph nodes are repeated, as indicated by their colors. We expect a decent model to contain a few unique nodes in the neighborhood of each node

Algorithm 1 Building KNN

Input: a matrix M storing the similarities between vector embeddings, nHops, K
Output: a dictionary D storing the closest neighbors

$D \leftarrow \emptyset$
for $node \in M.rows$ **do**
  $D \leftarrow D \cup \{node\}$
  $L \leftarrow$ sorted row M[node] decreasingly w.r.t similarity scores of the node and nodes excluding the node itself
  $D[node] \leftarrow L[0:K]$
  $Neighbors \leftarrow D[node]$
  $counter \leftarrow 1$
  $D \leftarrow$ Building Recursive KNN(M, D, node, Neighbors, nHops, K, ,counter+1)
end for

Algorithm 2 Building Recursive KNN

Input: M, D, node, Neighbors, nHops, K, counter
Output: D

```
if counter ≤ nHops then
    for n ∈ Neighbors do
        D[node][counter][n] ← ∅
        L ← sorted row M[n] decreasingly w.r.t similarity scores of the node and
              nodes excluding the node itself
        D[node][counter][n] ← L[0 : K]
        Neighbors ← D[node][counter][n]
        D ← Building Recursive KNN(M, D, node, Neighbors, nHops, K, ,counter+1)
    end for
end if
```

2.6 Community and Community Detection

A community in graph theory is a group of nodes that are densely connected to each other within the group and have fewer connections with nodes outside of the group. Communities can be considered as clusters or subgroups within a larger network. Thus, community detection is to divide nodes in a network (graph) into groups where the nodes in a group are densely connected whereas nodes in different groups are sparsely linked. Community detection (Jin et al., 2021) is a fundamental task in network analysis, and it has many practical applications in various fields, including social network analysis, computer science, biology, etc. There are several approaches to detect communities, such as Modularity-based methods which we utilize in this project, and these models aim to maximize the modularity of the graph, which measures the strength of the division of a network (graph) into modules.

3 Implementation steps

With the dataset in hand, we initially begin the project by identifying its key properties. The average, maximum, and minimum values for the defined properties distribution are shown below (we define the length of documents as the number of unique words and numbers):

Property	Mean	Min	Max
Distribution of the lengths	118	1	1932
Distribution of the lengths after removing outliers	78	1	287
Word frequencies	12	1	45786
Category frequency	7	1	1484

3.1 Document Selection

After determining these properties, we choose Categories as the criterion to select documents. For this purpose, we calculate a unique TF-IDF weight for each category by averaging the TF-IDF values of that category across all documents. Using the FP-Growth algorithm, we additionally discover common categories and a combination of categories

with at least 5% support. Then, using the union of frequent (top 13 ones obtained using FP-Growth Algorithm 1) and high TF-IDF (>= 0.012) value categories, we choose to select about 80% of our documents (4061). Tables of category selection are shown in the appendix (figure 5).

3.2 KNN and Recursive KNN creation

After selecting the documents, the next step is to build the KNN and Recursive KNN graphs of these documents. With the 3 different embeddings of these documents, we built 3 different KNN graphs using cosine similarity as a similarity measure and documents as vertices (nodes). For plausibility checking, we also build a KNN graph using a random matrix as cosine similarity. We next enlarge these KNN graphs using the aforementioned Algorithms 1 and 2 to create the equivalent Recursive KNN graphs. Having these Recursive KNN graphs and defining each nHops as a level for each node, we define and derive three features. They are as follows:

- Number of neighbors in the whole levels of each node
- Mean of cos-similarities in the whole levels of each node
- Number of categories in the whole levels of each node

We analyze these features in all nodes, the top 5% of the nodes with the most and least number of neighbors of these four Recursive KNN graphs. The histogram plots of these features are shown in the result section.

3.3 Community Detection

Investigating the chosen subgraphs and dense clusters of the KNN graphs, in addition to the previously mentioned properties, is our attempt to uncover additional features. For this purpose, we use the *Networkx* 2004 library in Python and the greedy_modularity_communities (Jin et al., 2021) method which Clauset-Newman-Moore greedy modularity maximization to detect communities in our graphs and find the community partition with the largest modularity. You can find the detected communities in Figure 2.

```
networkx.greedy_modularity_communities(G, weight=similarity, resolution=1, cutoff=1, best_n=None)
```

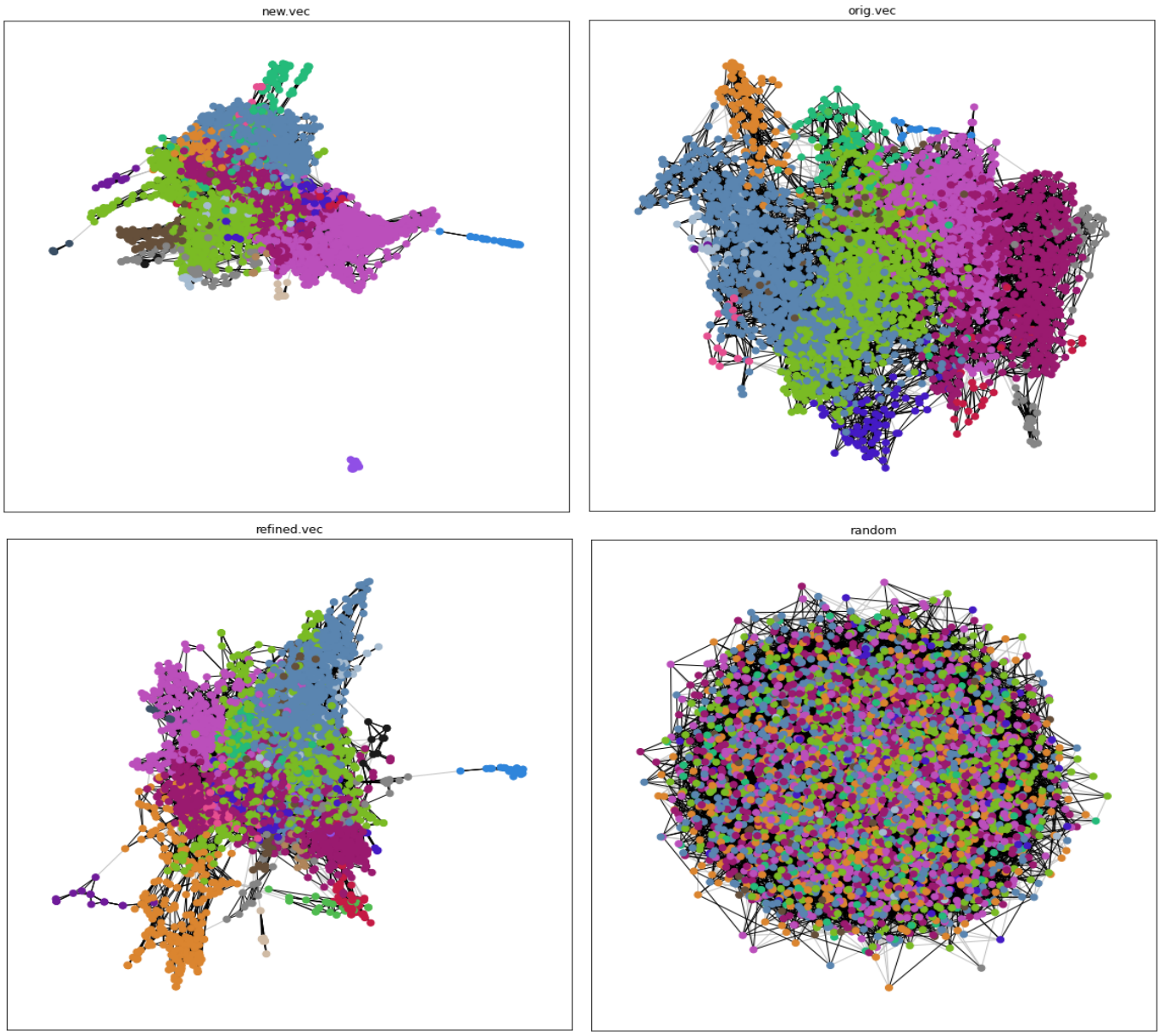


Figure 2: Communities obtain by the algorithm

Looking at the parameters, we pass cosine similarity as edges' weights. The resolution parameter sets an arbitrary tradeoff between intra-group edges and inter-group edges. If the resolution is less than 1, modularity favors larger communities. Greater than 1 favors smaller communities. The cutoff is the minimum number of communities below which the merging process stops and best_n is the maximum number of communities above which the merging process will not stop

4 Results

In the previous section, we derived three features, namely the number of neighbors, the number of categories, and the mean of cos-similarities. We plot the distribution of the features on the entire nodes (the top 5% of the nodes with the most and least number of neighbors are shown in the appendix (figure 6)). The plots on the left-hand side include

and the ones on the right-hand side exclude the KNN graph made using a random matrix. Plots are displayed in figure 3 and they show that the features are appropriate for our goal based on the distance between the random matrix and others. Also, we can discover that the mean of cosine similarity distinguishes between various embeddings more effectively than the other two.

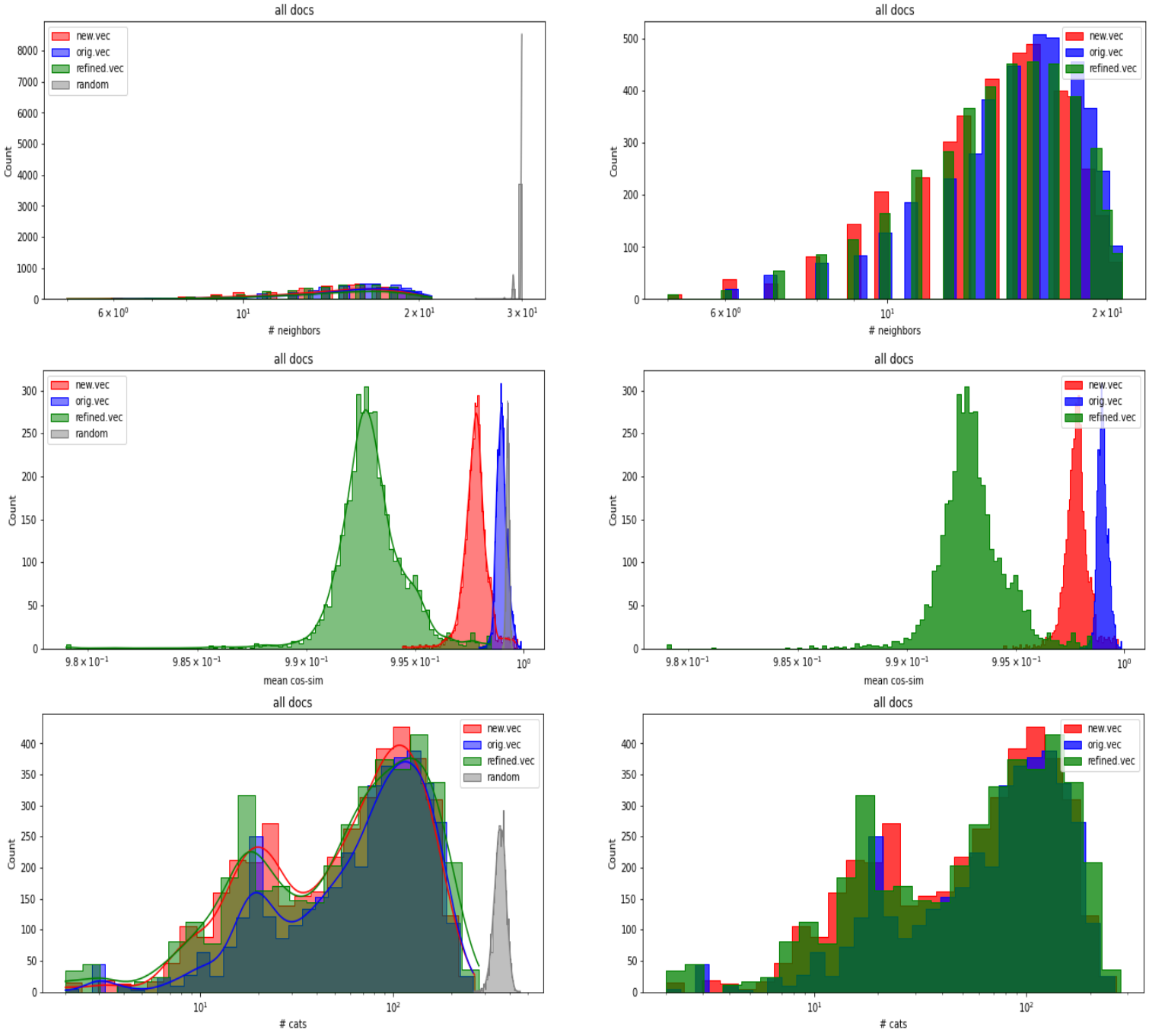


Figure 3: Feature distributions on entire nodes

We then create a scoring technique using the characteristic vectors of each node (document) according to their categories in the KNN graphs by pairwise summing these vectors

and counting the incidence of 2s which shows the shared categories between two documents. Although there is a gap between the models and the random matrix in its results, because the scores of the three models are so close together, we are not able to accomplish much with this technique, and future research should focus on alternative approaches. Finally, we uncover further features by discovering the communities of the graphs. Figure 4 contains the scatter plots of the detected communities and the same features as before.

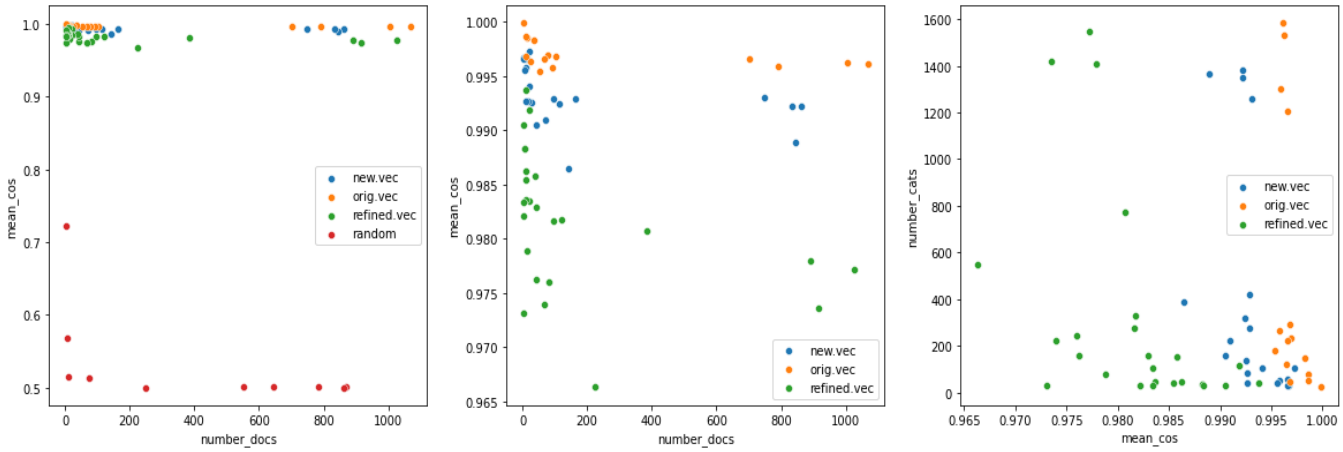


Figure 4: Community detection

The points in the scatter plots represent the communities of the KNN graphs. It is possible to see the distinction between the random matrix's communities and those of the other models. The scatter plots of the combination of each of these features with the mean of cosine similarity are shown here as the combination of the number of categories and the number of documents could not clearly separate the models

Moreover, we can observe that cosine similarity can still be used to discriminate between the embeddings of the communities and subgraphs. Also, we obtain the following table by changing the resolution parameter in the community detection algorithm:

Resolution	new.vec	refined.vec	orig.vec	random
1	21	20	17	15
2	32	32	22	57
5	60	64	50	165
8.5	94	99	86	258
10	108	112	107	295

An interesting observation of this table is that the numbers of created communities using three models are closer together than the random matrix. Also, another interesting discovery is that the number of communities in new.vec and refined.vec are close and they have more communities than the orig.vec for resolution $\in [1, 15]$. To get at a consistent result for the correlation of resolution and language models, more detailed investigations must be conducted which will be part of future work. However, this is a crucial discovery that aids us in future works and scoring techniques on the background of more detailed knowledge of the model training.

5 Conclusion

This project presents useful statistical and structural features that could aid us in distinguishing several language models within the same corpus. A subset of the Amazon Cat-13k dataset and embeddings from three different language models were used in our experiments. With these embeddings, we constructed KNN and Recursive KNNs graphs and examined them. Results demonstrate that the mean of cos-similarity is more effective than features derived from number neighbors or common categories at differentiating between different vector embeddings. Finding the communities of the graphs also shows that different models produce varying numbers of communities, which can also help us with our objective. We also put our model to the test using two embeddings from another Amazon Cat-13k subset that were given to us by Dr. Brito and Ms. Mirhaj. One embedding was achieved by the SBERT (Reimers and Gurevych, 2019) model and the other one by fine-tuning the same model using the categories as a similar indicator of the embeddings. The appendix contains the plots (figure 7).
Future works on enhancing and developing more precise evaluation models will be guided by examining the correlation between our results and models' performances on downstream tasks as well as combining the results with additional analysis methodologies like semantic analysis.
The KNN diagnosis tool is available via *GitHub* 2023.

References

Agrawal, Rakesh and Ramakrishnan Srikant (1994). "Fast algorithms for mining association rules in large databases". In: *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB Endowment, pp. 487–499.

Beltagy, Iz, Matthew E. Peters, and Arman Cohan (2020). "Longformer: The Long-Document Transformer". In: *arXiv:2004.05150*.

Bhatia, K. et al. (2016). *The extreme classification repository: Multi-label datasets and code*. URL: http://manikvarma.org/downloads/XC/XMLRepository.html.

Chatgpt (2022). https://icy-flower-0734d3310.2.azurestaticapps.net/blog/chatgpt. Accessed on November 30, 2022.

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186.

Eppstein, D., M.S. Paterson, and F.F Yao (1997). "On Nearest-Neighbor Graphs". In: *Discrete Comput Geom 17*. Springer, pp. 263–282.

GitHub (2023). https://github.com/behnamfani/KNN-diagnosis-tool.

GPT-3.5 (2022). https://platform.openai.com/docs/model-index-for-researchers. Accessed on February 10, 2023.

Han, Jiawei, Jian Pei, and Yiwen Yin (2000). "Mining frequent patterns without candidate generation: A frequent-pattern tree approach". In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. ACM, pp. 1–12.

Jin, Di et al. (2021). "A Survey of Community Detection Approaches: From Statistical Modeling to Deep Learning". In: *CoRR* abs/2101.01669.

Mikolov, Tomas et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Networkx (2004). https://github.com/networkx/networkx. Accessed on January 7, 2023.
Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543.
Reimers, Nils and Iryna Gurevych (2019). "Sentence-bert: Sentence embeddings using siamese bert-networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992.

5.1 Appendix

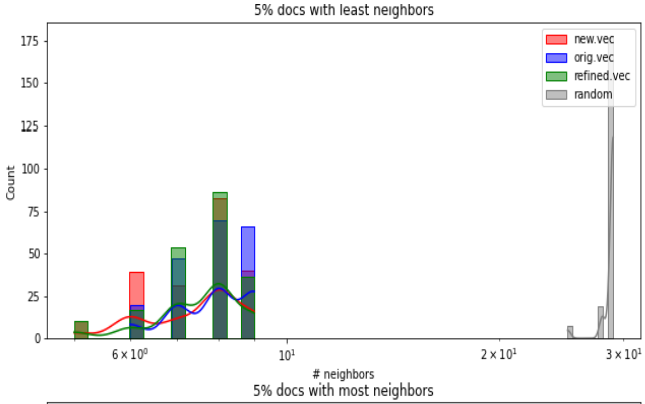
Category	tf-idf
books	0.065052
movies&tv	0.064402
music	0.061101
movies	0.051342
pop	0.039792
rock	0.028509
tv	0.025157
literature&fiction	0.023185
home&kitchen	0.016371
worldmusic	0.016097

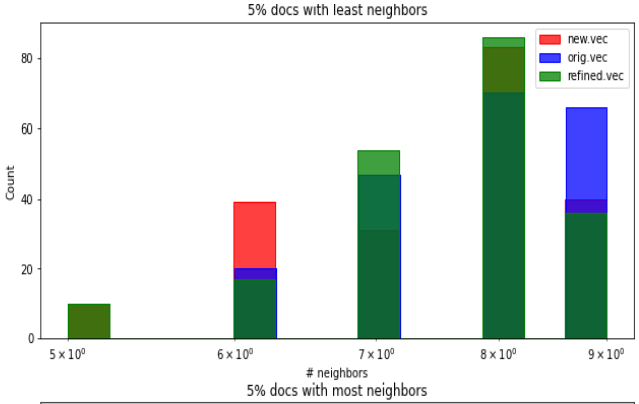
(a) high TF-IDF value categories

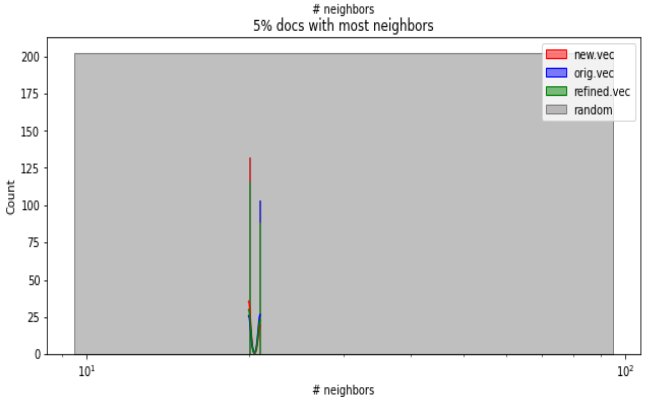
Category	tf-idf
0.2968	(books)
0.1560	(music)
0.1026	(movies&tv)
0.0972	(pop)
0.0972	(pop, music)
0.0790	(literature&fiction, books)
0.0790	(literature&fiction)
0.0718	(movies)
0.0692	(movies, movies&tv)
0.0652	(education&reference)
0.0646	(new)
0.0646	(used&rentaltextbooks)
0.0646	(books, used&rentaltextbooks)
0.0646	(new, used&rentaltextbooks)
0.0646	(books, new)
0.0646	(books, new, used&rentaltextbooks)
0.0638	(education&reference, books)
0.0620	(rock, music)
0.0620	(rock)
0.0542	(home&kitchen)
0.0502	(sports&outdoors)
0.0500	(pop, rock)
0.0500	(rock, music, pop)

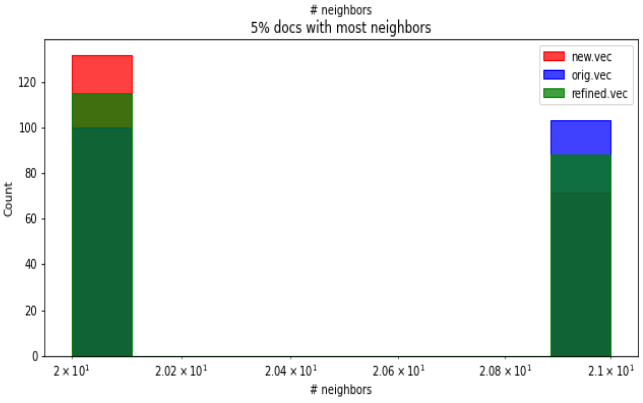
(b) Frequent categories

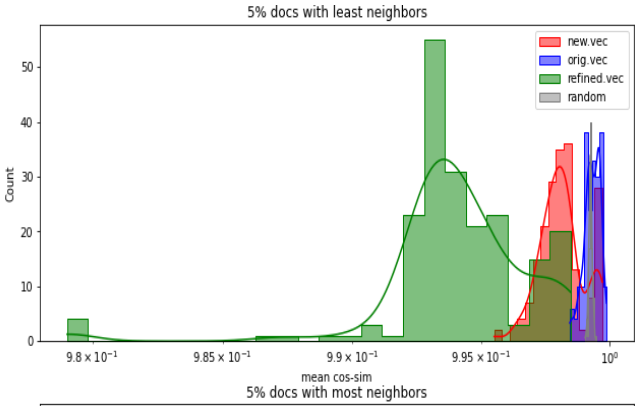
Figure 5: Tables of high TF-IDF value and frequent categories

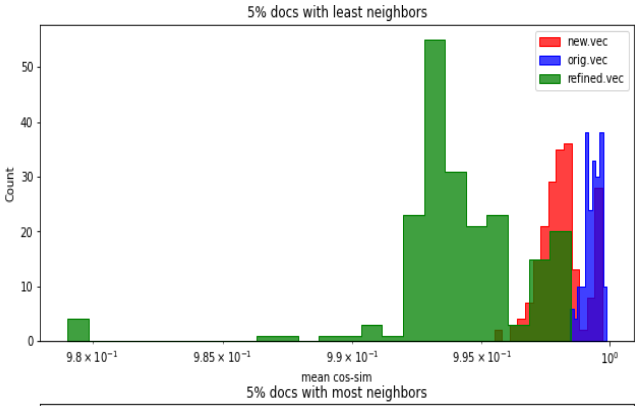

5% docs with least neighbors
new.vec
orig.vec
refined.vec
random
Count
neighbors

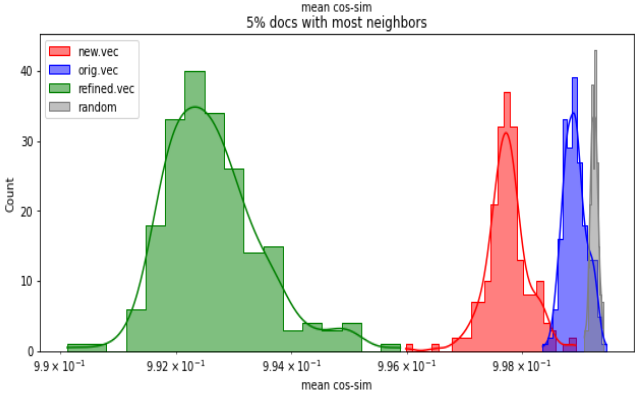

5% docs with least neighbors
new.vec
orig.vec
refined.vec
Count
neighbors

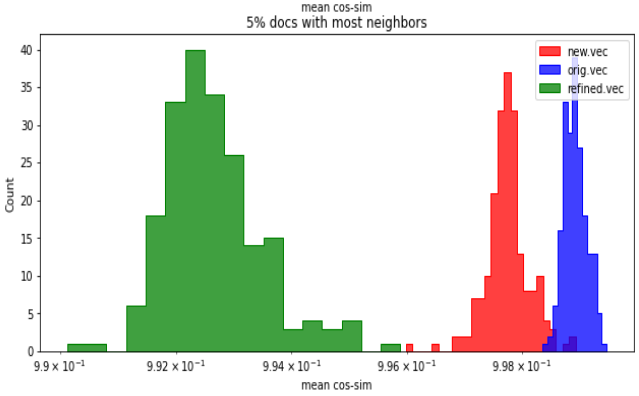

5% docs with most neighbors
new.vec
orig.vec
refined.vec
random
Count
neighbors


5% docs with most neighbors
new.vec
orig.vec
refined.vec
Count
neighbors


5% docs with least neighbors
new.vec
orig.vec
refined.vec
random
Count
mean cos-sim


5% docs with least neighbors
new.vec
orig.vec
refined.vec
Count
mean cos-sim


5% docs with most neighbors
new.vec
orig.vec
refined.vec
random
Count
mean cos-sim


5% docs with most neighbors
new.vec
orig.vec
refined.vec
Count
mean cos-sim

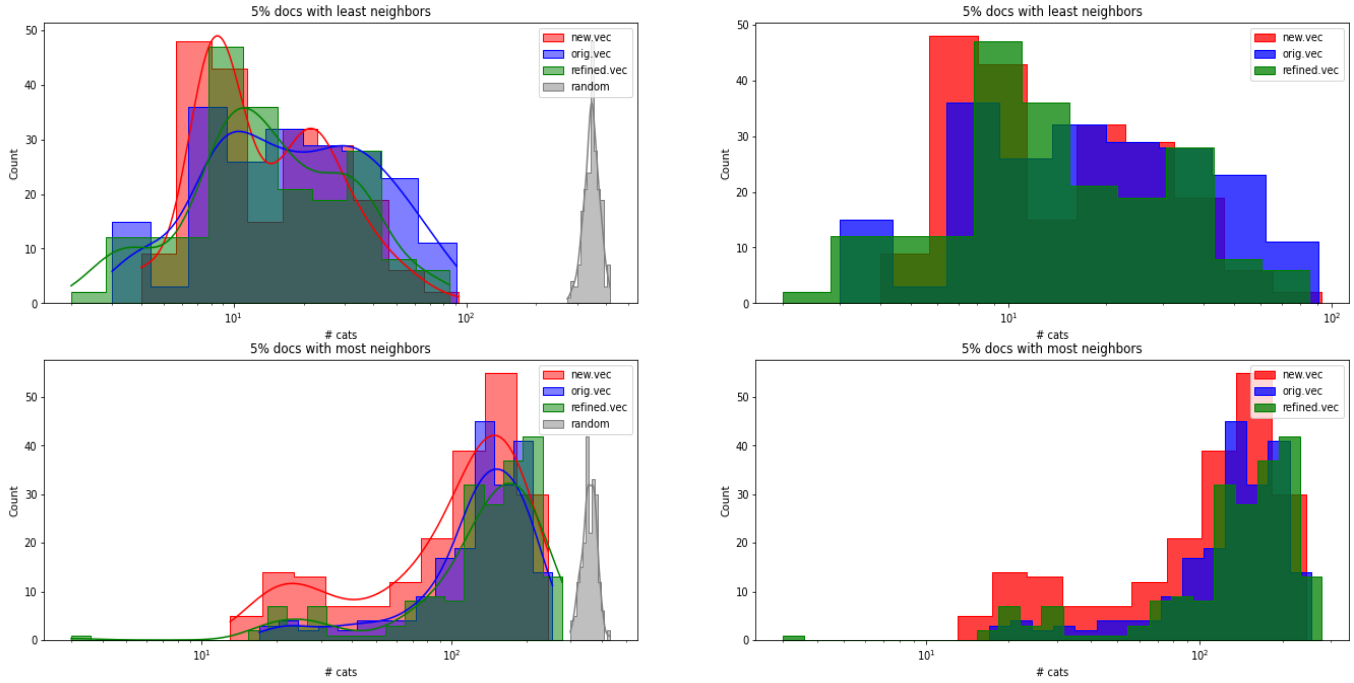


Figure 6: Feature distributions the top 5% of the nodes with the most and least number of neighbors

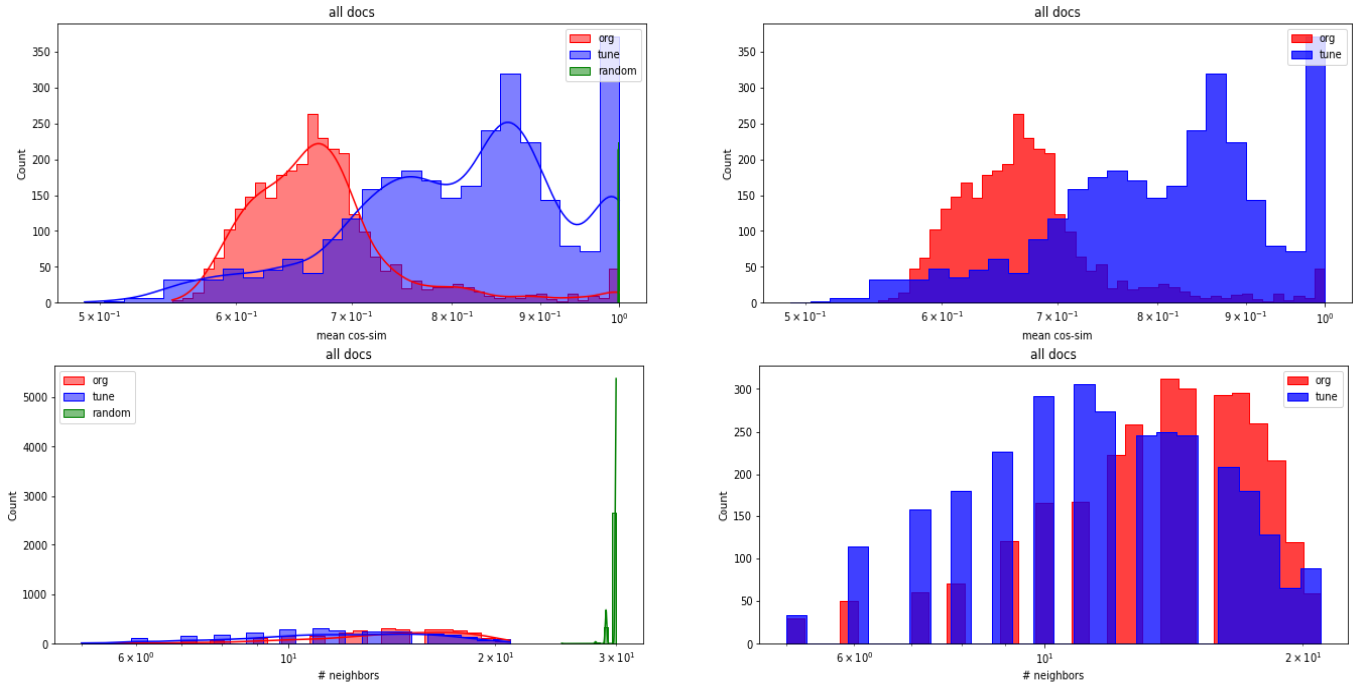


Figure 7: Embeddings obtained by normal and fine-tuned SBERT presented by Dr. Brito and Ms. Mirhaj

The histogram of embeddings with fine-tuning is displayed as blue, whereas the other is displayed as red. We observe an improvement in the embeddings after fine-tuning when we take into account the two observations, namely the increase of mean cos-similarity of fine-tuning embeddings as well as the decrease in the number of neighbors in the neighborhoods of Recursive KNN graphs.