Course-ID:	MA-INF 4306
Course:	Lab Development and Application of Data Mining and Learning Systems: Data Mining
Term:	Summer 23
Supervisor(s):	Dr. Jörg Kindermann
Students:	Behnam Fanitabasi, Tahmina Akter

Training and Analyzing Transformer Language Models

Behnam Fanitabasi - Tahmina Akter

September 3, 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. Choosing the best model for a given task, however, becomes more difficult as there are more and more models available. As a result, we search for several evaluation techniques in this project that will help us choose a model. It would also assist us in detecting the Out-of-Distribution data to some level, as we compare the performances of several models trained with various datasets on a unique dataset that was different from the trained ones.

Behnam

Contents

List of Figures

1 Introduction

Behnam

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 enormous amounts of text data using 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. Furthermore, when a language model is exposed to inputs that are significantly different from the data it has seen during training, it can struggle to provide accurate or meaningful outputs because it lacks the necessary context and patterns to handle these new inputs effectively. An Out-of-Distribution dataset could contain uncommon words, phrases, topics, or styles of language that the model has not encountered in its training data. For example, if a model has only been trained on Wikipedia documents, it might struggle with advertising documents such as Amazon descriptions. It is crucial to identify Out-of-Distribution inputs when a model encounters them since it makes language models safer to use in practical applications. It can prevent the model from generating overly confident, misleading, or nonsensical responses when it encounters unfamiliar input. Thus, it is important to select the model that performs the best on the present dataset.
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 an evaluation method for neural language models based on the embedding space. We combine different evaluation methods such as Perplexity and KNN-diagnosis-tool with features such as Key-phrase Extraction to compare the different Language Models using 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 unique key-phrases in the nearest neighbor graphs created by the embeddings and structural features such as chosen subgraphs and dense clusters of the KNN graphs.

1.1 Related Work

Behnam and Tahmina

KNN-diagnosis-tool (Fanitabasi and Kindermann, 2023) is a tool developed in the Lab Development and Application of Data Mining and Learning Systems: Data Science and Big on Winter semester 22-23. It is used to distinguish several embeddings of the same corpus that were acquired by various language models. It employs the following features:

- 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

on the (Recursive) nearest neighbor graphs and their communities where each document is a node and the edges are drawn based on how related the nodes are to one another. The findings of the lab demonstrated that the mean of cos-similarity is more effective than features derived from number neighbors or common categories at differentiating between different vector embeddings. In this project, we attempt to combine this tool with

additional evaluation techniques, such as key-phrase analysis and perplexity, to assess various language models.
A recent research paperLi et al., 2023 titled "A Survey on How Well NLP Models Handle Unfamiliar Text," discusses the testing of natural language processing models when they encounter text they aren't trained on. They discuss three important issues such as making models strong against tricky attacks, making sure they can understand different topics, and dealing with unfairness in the data they are taught from. This paper undertakes a pivotal task by unifying these disparate research areas under a comprehensive framework, thereby filling a notable gap in the existing literature. Through this survey, the authors not only provide a comparative analysis of the three research lines but also identify that Out-of-Distribution inputs are vital for ensuring the safety of language models in practical applications and for selecting the most suitable model for the present dataset. The authors' conclusion advocates for a holistic approach to OOD evaluation, urging researchers to enhance the generalization of NLP models across all three dimensions of OOD data, while also highlighting a critical gap in bridging the divide between adversarial robustness and real-world OOD evaluation.

2 Definition

2.1 Neural Natural Language Models and Vector Embedding

Behnam

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.

2.2 AmazonCat-13k

Behnam

The dataset we are using is called Amazon Cat-13k (Bhatia et al., 2016). It has 13,330 product labels and 1,186,239 (English) product descriptions. The AmazonCat-13k dataset is a large and complex dataset where the properties are not evenly distributed, some properties are more common than others, and there are errors and inconsistencies in the data. Nevertheless, it is a valuable resource for machine learning research. For this project, we randomly select a subset of this dataset with 15,000 documents, which we later use 8,500 documents (due to resource limitation) for training and 5,000 documents for evaluation.

2.3 Term Frequency–Inverse Document Frequency

Behnam

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

Behnam

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 KNN-diagnosis-tool (Fanitabasi and Kindermann, 2023)

Behnam

When given different text embeddings of the same text, the KNN-diagnosis-tool creates KNN and Recursive KNN graphs from the embeddings and extracts various features, such as the mean of cos-similarities and the number of neighbors in the nearest neighbor graphs created using the embeddings, as well as structural features, including chosen subgraphs and dense clusters of the KNN graphs, that may be useful to distinguish the embeddings. In this project, number of common key-phrases is the main feature we want to extract from the graphs.

2.5.1 Nearest Neighbor and Recursive Nearest Neighbor Graph

Behnam and Tahmina

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 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 original node and continuing the same process.

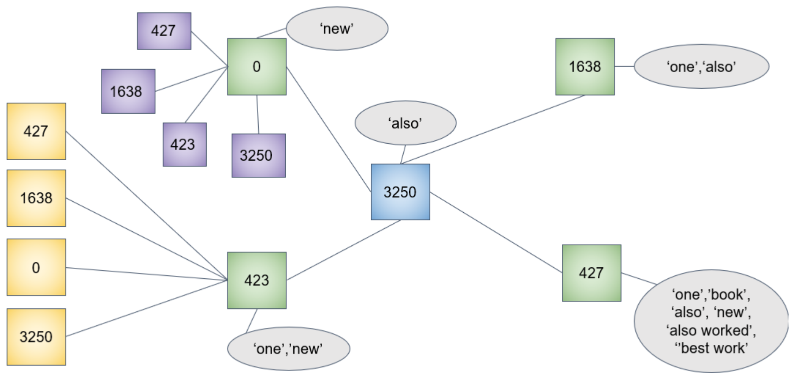


Figure 1: Example of a node in Recursive KNN with K=4
Each node is a document with the number as an identifier. Some Recursive KNN graph nodes are repeated, as indicated by their colors. Each node has a set of key-phrases. We expect a decent model to contain a few unique nodes in the neighborhood of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

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

2.5.2 Community and Community Detection

Behnam

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. The KNN-diagnosis-tool uses the Modularity-based method (*Networkx* 2004) which aims to maximize the modularity of the graph, which measures the strength of the division of a network (graph) into modules.

2.6 Key-phrase Extraction

Tahmina

Key phrase extraction is a natural language processing technique that involves automatically identifying and extracting the most important and relevant phrases or terms from a given text. The extracted key phrases capture the essential topics, concepts, and ideas discussed in the text, facilitating summarization, information retrieval, categorization, and organization of textual content. This process aids in condensing the main content of a text into a concise representation of its core meaning, without requiring readers to go through the entire text. There are several approaches to keyphrase extraction, ranging from traditional statistical methods to more advanced machine learning techniques. Some common methods include:

- Frequency-based Methods: These methods identify keyphrases by analyzing the frequency of words or phrases within the document. Terms that appear frequently are often assumed to be important. However, these methods might not consider the context or semantic meaning of the terms.
- TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF is a statistical measure that evaluates the importance of a term within a document relative to its importance in a collection of documents. Terms that are frequent in the current document but rare in the corpus receive higher TF-IDF scores.
- Graph-based Methods: These methods build graphs where nodes represent words or phrases, and edges represent relationships between them. Algorithms like Tex-

tRank, inspired by PageRank for web page ranking, use graph analysis to identify keyphrases based on their centrality in the graph.

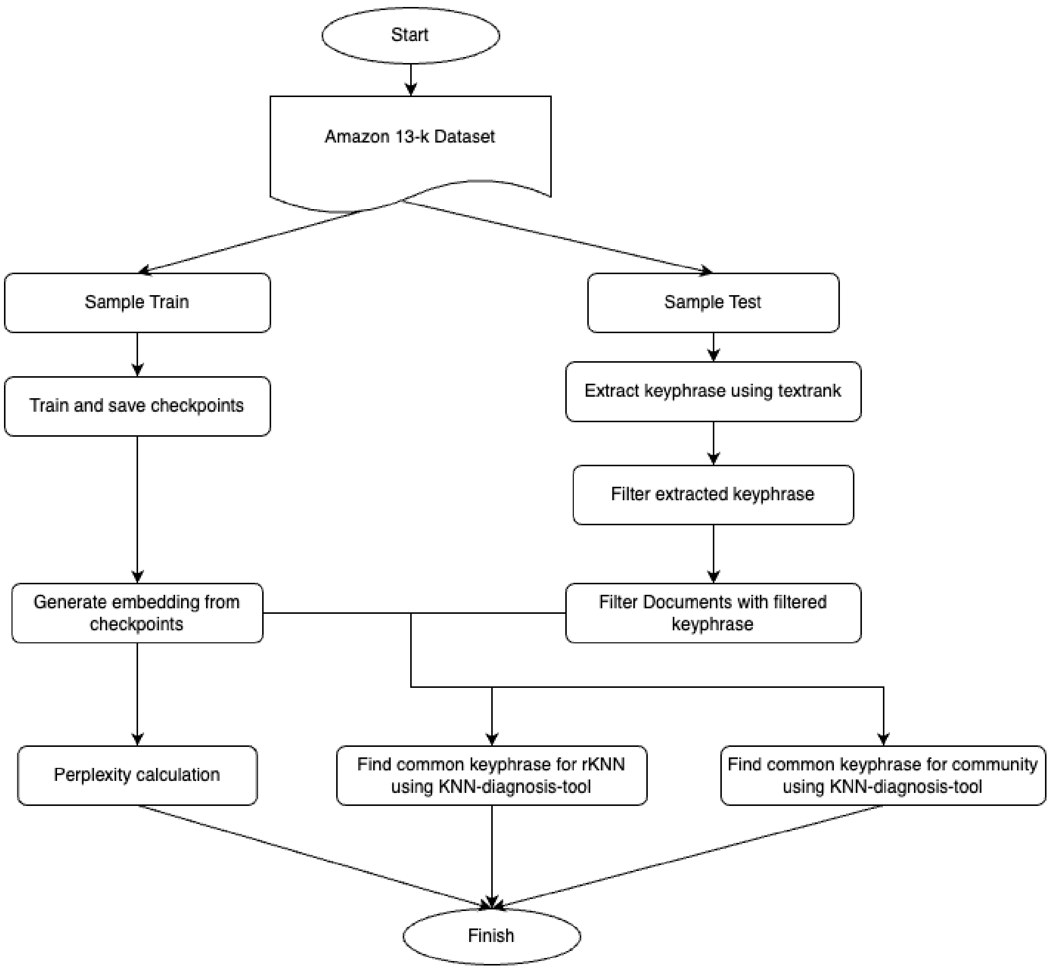
- Topic Modeling: Techniques like Latent Dirichlet Allocation (LDA) can identify latent topics in a document collection and associate keyphrases with these topics. This method is especially useful for capturing the underlying themes within a set of documents.

The choice of method depends on the specific use case, available data, and desired level of accuracy. Carlo Abi Chahine et al., 2009 introduced a technique for a support system in indexing, which involves taking an ontology and an ordinary text document as input and generating contextualized keywords from the document as output. The approach's efficacy was assessed through the utilization of Wikipedia's category links as a resource for terminological information.

3 Flow Diagram

Figure - 2 has the flow diagram that shows the steps we followed to do our research.

Tahmina



Start
Amazon 13-k Dataset
Sample Train
Sample Test
Train and save checkpoints
Extract keyphrase using textrank
Filter extracted keyphrase
Generate embedding from checkpoints
Filter Documents with filtered keyphrase
Perplexity calculation
Find common keyphrase for rKNN using KNN-diagnosis-tool
Find common keyphrase for community using KNN-diagnosis-tool
Finish

Figure 2: Flow Diagram

4 Implementation steps

4.1 Model Training and Embeddings Calculation

Behnam

In order to compare various models, we first pick two models and fine-tune them with a subset of the AmazonCats-13k dataset having 8,500 product descriptions. For this purpose, we choose bert-base-uncased (Devlin et al., 2018) which was pretrained on BookCorpus, a dataset consisting of 11,038 unpublished books (Zhu et al., 2015) and English Wikipedia (excluding lists, tables, and headers) (Foundation, n.d.) in a self-supervised fashion with two objectives: Masked language modeling (MLM) and Next sentence prediction (NSP). In the uncased version, all the texts have been lowercased. We also choose longformer-base-4096. longformer-base-4096 (Beltagy, Peters, and Cohan, 2020) is a BERT-like model started from the RoBERTa checkpoint with the Masked language modeling objective which supports sequences of length up to 4096 tokens. It processes long sequences of text efficiently by introducing local and global attention patterns. Instead of a fully connected attention matrix (quadratic computational complexity w.r.t sequence length), it has a sparse attention matrix that makes the computational complexity linear. In addition to BooksCorpus and English Wikipedia, it was also pretrained on CodeSearchNet, a dataset of 2 million (comment, code) pairs from open-source libraries hosted on GitHub (Husain et al., 2019). The TSDAE (Wang, Reimers, and Gurevych, 2021) approach is used to fine-tune the BERT model. In this method, an encoder maps a noisy input to a fixed-sized embedding, and a decoder then tries to recreate the original text without the noise. The Longformer model is fine-tuned using masked language modeling where the task is to predict masked words or tokens in a given text. The *Experiments and Results* section contains the setup that is used to train the models. The final models and the checkpoints at different steps are saved, and the evaluation set is given to the models' encoders in order to calculate the embeddings. The embeddings are the hidden states and each resulting embedding has a dimension of 768.

4.2 Key-phrase Extraction

Tahmina

After the process of generating embeddings, our focus shifts towards the examination of our test datasets. In order to ensure data quality, we first conduct the removal of null values from the test dataset, consolidating the dataset into a coherent unit. The subsequent objective is the extraction of key-phrases from this composite text.
For this purpose, we use the TextRank algorithm, a noteworthy graph-based ranking model in the domain of phrase extraction in NLP, as introduced by the authors in the paper cited as Mihalcea and Tarau, 2004. The underlying principle of TextRank revolves around the construction of a word network. This network is built on the basis of the sequential appearance of words within the text. Notably, connections are established between words that appear in immediate succession. The strength of these connections is determined by the frequency with which the two words co-occur within close proximity in the text.
Subsequently, our focus narrows to the identification of "Noun" phrases as the target of the analysis. Through the utilization of the TextRank tools, our exploration yields a substantial total of **50,015** distinct phrases extracted from our sample test dataset.

4.3 Key-phrase Filtering

After the initial extraction of keywords, a series of sequential refinement steps has been undertaken: *Tahmina*

1. **Phrase Length Restriction:** Phrases are constrained to a maximum length of five words. This results in a discernible reduction from an initial pool of 50,015 phrases to 47,492 phrases.
2. **TF-IDF Thresholding:** Utilizing Term Frequency-Inverse Document Frequency (TF-IDF) analysis, words within the identified phrases are subjected to a threshold criterion of 0.004. Only words exceeding this threshold are retained, causing a reduction from the aforementioned 47,492 phrases to a refined set of 17,156 phrases.
3. **Phrase Frequency-based Pruning:** Employing a criterion of retaining the top decile (i.e., the highest 10%) of phrases based on their frequency, the phrase count is further reduced from 17,156 to 1,715 phrases.
4. **Word Frequency-based Pruning:** A subsequent refinement stage involved preserving only the top percentile (i.e., the uppermost 1%) of phrases, based on word frequency. This final pruning leads to a culled set of 1,380 phrases.

In this manner, a systematic and iterative approach is employed to progressively distill the initially extracted keywords into a more focused and reduced collection, resulting in the final set of 1,380 scientifically curated phrases.

4.4 Document Filtering

Tahmina

After performing the initial step of filtering out key-phrases, we proceed to review and organize our evaluation dataset. This step is taken to enhance the efficiency of our subsequent processes. We choose to retain only those documents that include at least one key-phrase from the set of filtered key-phrases.
We observe that out of the total of 5000 documents we had initially, 2755 documents met the criteria of containing at least one of the filtered key-phrases. These 2755 documents are then chosen to be the final set of documents on which we conduct our comprehensive evaluation.

4.5 Evaluation

Tahmina

The evaluation framework adopted in this study constitutes three fundamental stages: Perplexity Computation, Recursive K-Nearest Neighbors (KNN) Analysis With Key-phrase Evaluation, and Community-Detection-Based Analysis. Each of these phases is meticulously designed to dissect distinct aspects of the model's performance and efficacy. The delineation of these stages is provided below:

4.5.1 Perplexity Computation

Behnam

Perplexity is one of the common metrics for evaluating language models which computes how well a probability model predicts a sample and captures the degree of 'uncertainty' or 'randomness' of that model. A well-written document has a low perplexity, whereas a random-written document has a high perplexity. We compute the embeddings of the evaluation set (5,000 product descriptions) using the trained models and saved checkpoints. Each description is passed to the appropriate tokenizers (the *bert-base-uncased* tokenizer for the Bert model and *Allenai/longformer-base-4096* for the Longformer model) to compute the token IDs of the description. Then the token IDs are fed to each model where the labels are the same as the token IDs. Each model's loss is calculated based on how well it can predict the masked tokens. Perplexity is calculated using the formula $perplexity = e^{(sum(Loss)/n_{token})}$, where n_{token} is the total number of tokens that the models have processed. The embeddings are also used for the next steps to create KNN and RKNN.

4.5.2 Recursive KNN Analysis With Key-phrase Evaluation

Tahmina

A recursive KNN approach is employed to establish semantic relationships among key phrases. The KNN technique facilitates the identification of proximate neighbors in a multidimensional space, thus affording a measure of similarity. Moreover, an additional layer of evaluation is imposed by considering the key-phrases. This ensures that the retrieved neighbors are not only geometrically proximate but also substantively aligned based on semantically significant key-phrases.
We follow a Python class named 'KNN' from the previous research of Fanitabasi and Kindermann, 2023 that provides functionality for creating and analyzing Recursive K-Nearest Neighbor (RKNN) networks, deriving features from them, and plotting various distributions based on the extracted features.
The main functionalities are as follows:

- K-Nearest Neighbor (KNN) Creation: The 'knn' method calculates the KNN and Recursive KNN networks based on a similarity matrix. It takes inputs such as the similarity matrix, document-keyphrase relationships, a document-category matrix, K value for neighbors, and nHops for the number of layers for RKNN.

- RKNN Creation and Analysis: The 'Create_RKNN' function orchestrates the creation of RKNN networks for different vector embeddings, using cosine similarity. It calculates the similarity matrix, constructs the RKNNs, and extracts features from them. The extracted features and scores are saved in dictionaries for further analysis.

Overall, this implementation provides a structured framework for creating RKNNs, extracting features, and visualizing the distributions and scores associated with the nodes in these networks. With this implementation, we identify document relationships, clustering, and understanding the significance of keyphrases in the context of the documents.

4.5.3 Community-Detection-Based Analysis

Tahmina

The final phase emulates the concept of community detection. Community detection algorithms are applied to identify coherent clusters of interconnected nodes. We explored different embeddings and K values to create and evaluate graphs. The analysis includes examining common key phrases associated with the max and min communities. We calculate edge weights based on similarity, detect communities using greedy modularity, and visualize the results.

- Parameters and Embeddings:

  - Graph (From previous KNN based on Keyphrase and number of neighbors K), distance metric ('cos'), and Resolution were considered as parameters.

  - Embeddings from each checkpoint were considered to iterate, each represented by a name and an associated file containing embedding data.

- Community Detection:

  - Community detection is performed on the graph following the same algorithm by the previous research of Fanitabasi and Kindermann, 2023

  - The max and min significant communities are extracted, and common key phrases associated with these communities are calculated.

- Filtering:

  - After Community detection, the common key phrases in max(30%) communities and min(30%) communities are considered.

All the values from community detection are stored in a nested dictionary structure, grouping them by K value and embedding type for our analysis.

5 Experiments and Results

Behnam and Tahmina

With an unmodified configuration, the bert-base-uncased model is trained for 10 epochs. The longformer-base-4096 model undergoes 80 epochs of training as no pre-trained model is used because of the configuration change. To make the model work with the tokenizer *Allenai/longformer-base-4096*, we increased the vocabulary size to 50265. Due to resource constraints, the number of hidden layers is also reduced from 12 to 5. For both models, the learning rate is set to $1e{-}4$ and the batch size is equal to 5.

Table 1: Perplexity of the models' checkpoints in different steps on the evaluation set

Longformer checkpoints	Perplexity	**Bert** checkpoints	Perplexity
4000	316.46	1000	735.09
8000	301.74	3000	164.02
12000	305.57	5000	270.42
16000	308.97	7000	223.63
20000	291.35	9000	96.67
24000	301.61	11000	252.14
28000	303.98	13000	619.23
32000	298.83	15000	105.01
34000	301.56	16000	220.84

We are unable to train the models effectively with enough data because of resource limitations. As a result, fine-tuning the models on our training set does not converge. However, we take the perplexity as a control parameter and compare its changes to the results obtained from RKNN- and Community-based analysis.
We conduct scientific experiments in various ways to explore different scenarios. To do this, we adjust two key parameters: "k" and "nhops." Specifically, we test these parameters

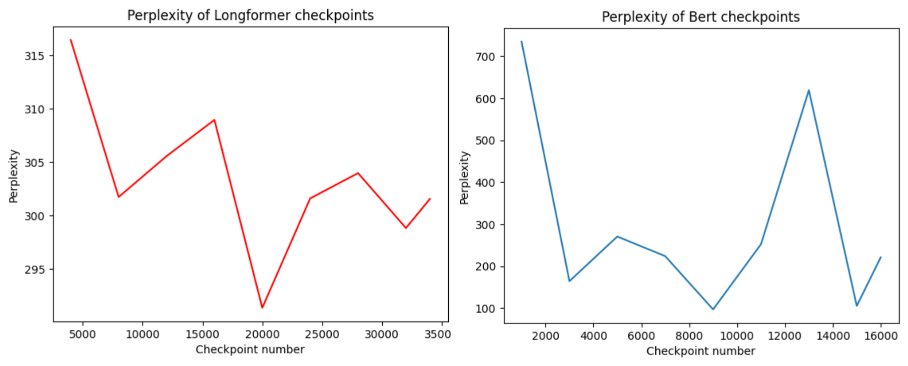


Figure 3: Perplexities of the models on different checkpoints

with values of 3, 4, and 5 for "k," and 3 and 4 for "nhops." Additionally, we investigate how changing the maximum number of key phrases per document affects our experiment. We test two threshold values: 5 and 10. Our experiment involves two language models, BERT and Longformer. In our specialized K-nearest neighbors (RKNN) analysis, we focus on 2% of the nodes with the most neighbors and 2% of those with the least neighbors. For community analysis, we look at the biggest (30%) communities and smallest (30%) communities to gain insights into their characteristics and behaviors.
All the experiments we perform using the BERT model are shown in Figure 6 and Figure 7. We expect an increase in the number of common key phrases for the latest checkpoint. However, we observe as the checkpoint increases, the number of key phrases does not consistently rise smoothly.
Another observation we made is that, in most scenarios(rKnn), the most recent checkpoint has the highest number of common key phrases. However, there are two exceptions in the scenario, specifically when k=4 and nHop=4, and when k=5 and nHop=4.

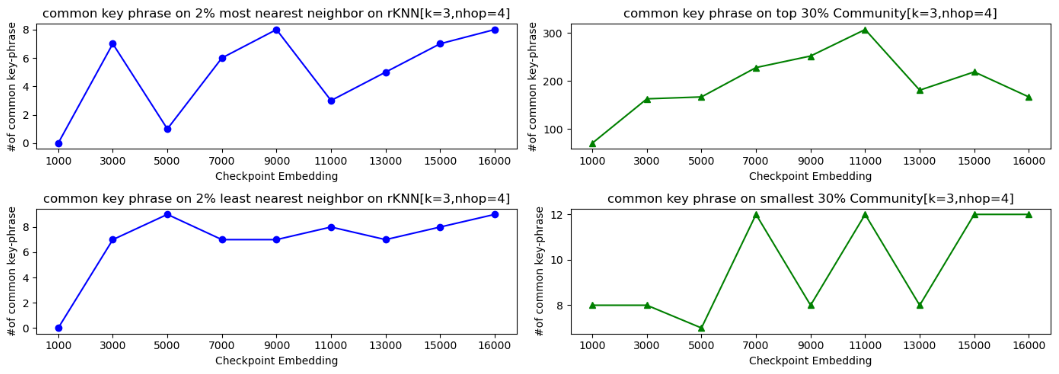


Figure 4: Common Key-phrase on RKNN and Community(k=3, nHops=4, model=Bert, maxKeyPhrase=10)

In Figure 4, where we set "k" to 3 and "nHops" to 4, the analysis works well for both types of analysis (rknn). For max(30%) community, it performs really well until we reach 11,000 checkpoints. After that, there is a drop in number of keywords. For the min(30%) community, there are some ups and downs in the middle, but it becomes stable towards the end. Unlike BERT, Longformer shows a noticeable distinction between neighbors with the most connections and those with the fewest shown in Figure 8 and Figure 9. In the case of Longformer, the graphs do not look very similar. In Figure 5, where K was 3 and nHops was 4, RKNN performed well but the Community graph has a fluctuation.

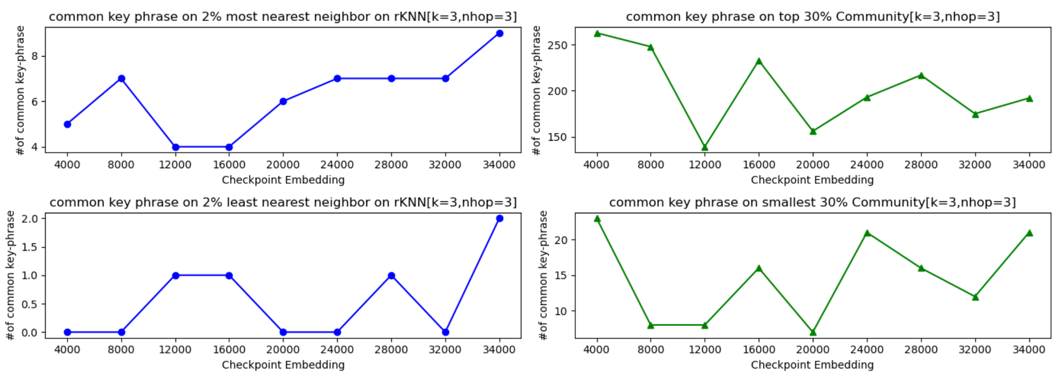


Figure 5: Common Key-phrase on RKNN and Community(k=3, nHops=4, model=Longformer, maxKeyPhrase=10)

We observe the relation between RKNN, Community, and Perplexity for both models(Table 2, Table 3, Table 4, Table 5). Here, '+' represents value increases, '-' represents value decrease, ' ' represents value does not change.

Our hypothesis is when the perplexity value increases, the number of common key phrases will decrease, and conversely, when the perplexity value decreases, the number of common key phrases will increase.

Analyzing Table 2, we can observe that our assumptions align with checkpoints 1000 to 3000 (completely), checkpoints 3000 to 5000 (with one contradiction), and checkpoints 3000 to 5000 (with one contradiction). Also, scenario K=3 with nHops=4 and scenario K=4 with nHops=4 tend to support our hypothesis more, despite two contradictions compared to other scenarios. From Table 3, we can see that for k=3 scenario gives a less conflicted result. From Table 4, we can see that checkpoint 12000 to 16000 almost completely contradicts our assumptions. And for scenario K=3, nHops=4 and scenario k=5, nHops=3 and scenario k=5, nHops=3 match(with four contradictions) with the hypothesis more compared to other scenarios.
We found that no scenario 100% matches our hypothesis. However, for both models, we observe that the relation between RKNN and perplexity matches with the hypothesis more than the relation between Community and perplexity. We also observed that between the two models, Bert model matches the hypothesis more than Longformer model.

Table 2: Observation of RKNN and Perplexity for Bert Model (max key phrase =10)

Checkpoint	K=3 nH=3	K=3 nH=4	K=4 nH=3	K=4 nH=4	K=5 nH=3	K=5 nH=4	Perplexity
emb 1000 to 3000	+	+	+	+	+	+	-
emb 3000 to 5000	-	-	+	-	-	-	+
emb 5000 to 7000	+	+	-	+	+	+	-
emb 7000 to 9000	-	+	+	-	+	-	-
emb 9000 to 11000	-	-		+	+	+	+
emb 11000 to 13000	+	+	-	-	-	-	+
emb 13000 to 15000	-	+	+	+	-		-
emb 15000 to 16000	+	+	-	-	+	-	+

Table 3: Observation of Community and Perplexity for Bert Model (max key phrase =10)

Checkpoint	K=3	K=4	K=5	Perplexity
emb 1000 to 3000	+	+	+	-
emb 3000 to 5000	+	+	+	+
emb 5000 to 7000	+	-	+	-
emb 7000 to 9000	+	+	-	-
emb 9000 to 11000	+	-	-	+
emb 11000 to 13000	-	+	+	+
emb 13000 to 15000	+	-	-	-
emb 15000 to 16000	-	+	+	+

Table 4: Observation of RKNN and Perplexity for Longformer Model (max key phrase =10)

Checkpoint	K=3 nH=3	K=3 nH=4	K=4 nH=3	K=4 nH=4	K=5 nH=3	K=5 nH=4	Perplexity
emb 4000 to 8000	+	+	-	-	+	+	-
emb 8000 to 12000	-	-	+	+	-	-	+
emb 12000 to 16000		+		-	+	+	+
emb 16000 to 20000	+	-		-	+	-	-
emb 20000 to 24000	+	+		+	-	+	+
emb 24000 to 28000		-		+		-	+
emb 28000 to 32000		+		-		-	-
emb 32000 to 34000	+		-			-	+

Table 5: Observation of Community and Perplexity for Longformer Model (max key phrase =10)

Checkpoint	K=3	K=4	K=5	Perplexity
emb 4000 to 8000	-	-	+	-
emb 8000 to 12000	-	+	+	+
emb 12000 to 16000	+	-	-	+
emb 16000 to 20000	-	+	+	-
emb 20000 to 24000	+	+	+	+
emb 24000 to 28000	+	-	-	+
emb 28000 to 32000	-	+	-	-
emb 32000 to 34000	+	-	+	+

6 Conclusion

This project presents useful statistical and structural features that could aid us in the evaluation of different language models prior to using them on downstream tasks. It also shows promising results in detecting the out-of-distribution datasets for different models and finding the better-suited models for these datasets. A subset of the Amazon Cat-13k dataset (training set) was used to fine-tune a BERT and a Longformer model. The projects' outcomes therefore demonstrated to us that the RKNN analysis with key-phrase evaluation has good potential for the desired tasks and can be further explored in future endeavours.

Behnam

The KNN-diagnosis-tool with the Key-Phrases extraction and the evaluation is available via *GitHub* 2023.

7 Future Work

Behnam

The project's next step will involve the usage of various datasets and language models. For different language models, we will evaluate them with KNN-diagnosis-tool using both in-distribution and out-of-distribution datasets in order to investigate the differences and the connection between our evaluation approach and the perplexity values. The evaluations are done using different values for the KNN-diagnosis-tool parameters, such as nHops and K, to see the effects and, ideally, find the optimal ones.
Furthermore, due to resource limitations, we were not able to train the models properly. Thus, we will use larger subsets of the datasets and alternative training configurations to increase the likelihood of the convergence of the models.

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`.

Chahine, Carlo Abi et al. (2009). "Context and Keyword Extraction in Plain Text Using a Graph Representation". In: *CoRR* abs/0912.1421. arXiv: `0912.1421`. URL: `http://arxiv.org/abs/0912.1421`.

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

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: `1810.04805`. URL: `http://arxiv.org/abs/1810.04805`.

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

Fanitabasi, Behnam and Jörg Kindermann (2023). *GitHub*. `https://github.com/behnamfani/KNN-diagnosis-tool`.

Foundation, Wikimedia (n.d.). *Wikimedia Downloads*. URL: `https://dumps.wikimedia.org`.

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.

Husain, Hamel et al. (2019). "CodeSearchNet challenge: Evaluating the state of semantic code search". In: *arXiv preprint arXiv:1909.09436*.

Li, Xinzhe et al. (2023). *A Survey on Out-of-Distribution Evaluation of Neural NLP Models*. arXiv: 2306.15261 [cs.CL].

Mihalcea, Rada and Paul Tarau (July 2004). "TextRank: Bringing Order into Text". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, pp. 404–411. URL: https://aclanthology.org/W04-3252.

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

Networkx (2004). https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.modularity_max.greedy_modularity_communities.html. 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.

Wang, Kexin, Nils Reimers, and Iryna Gurevych (Apr. 2021). "TSDAE: Using Transformer-based Sequential Denoising Auto-Encoderfor Unsupervised Sentence Embedding Learning". In: *arXiv preprint arXiv:2104.06979*. URL: https://arxiv.org/abs/2104.06979.

Zhu, Yukun et al. (Dec. 2015). "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". In: *The IEEE International Conference on Computer Vision (ICCV)*.

8 Appendix

Tahmina

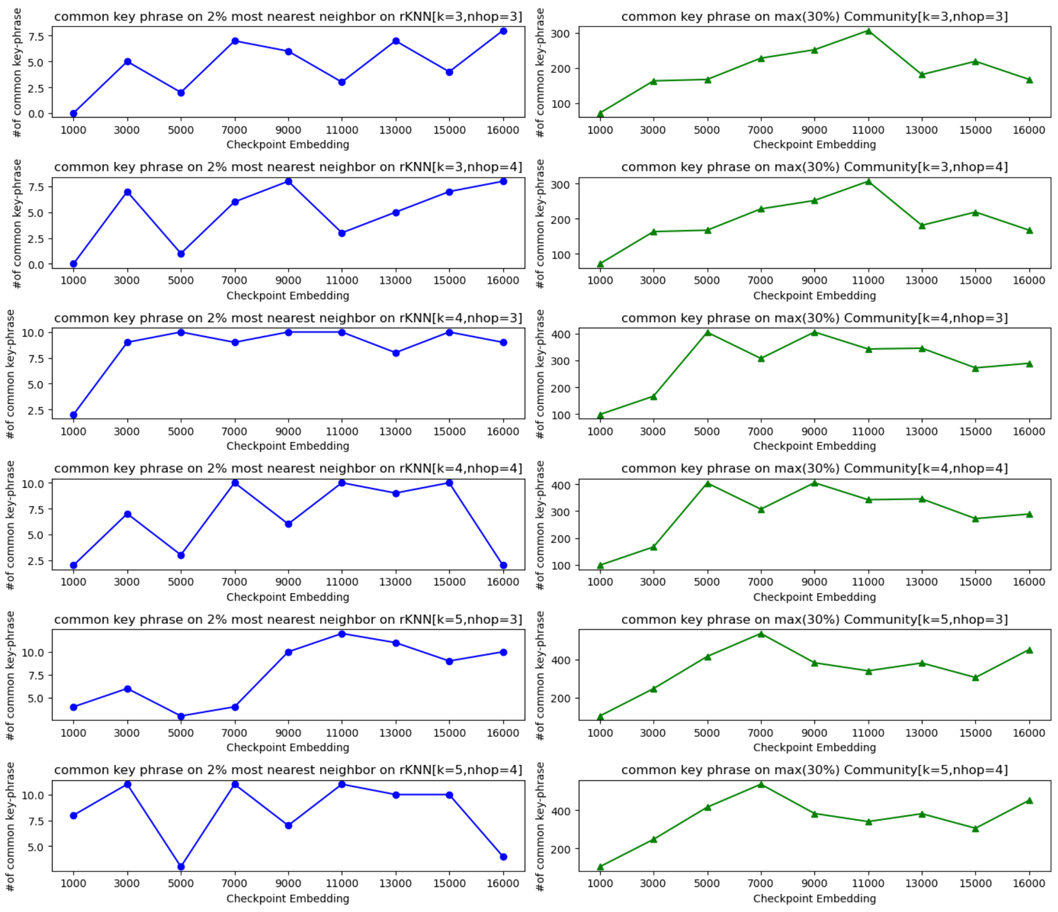


Figure 6: Experiment result for Bert Model(2% most nearest neighbour and max(30%) community)

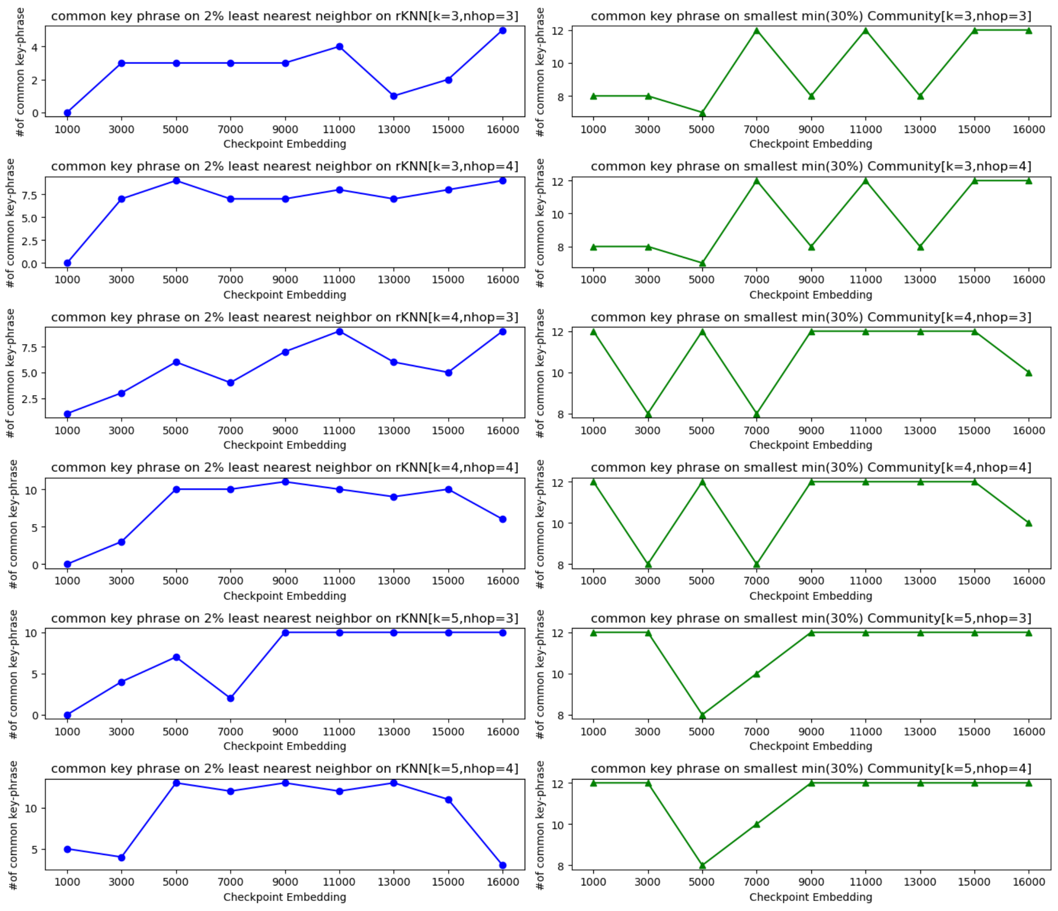


Figure 7: Experiment result for Bert Model(2% least nearest neighbour and min(30%) community)

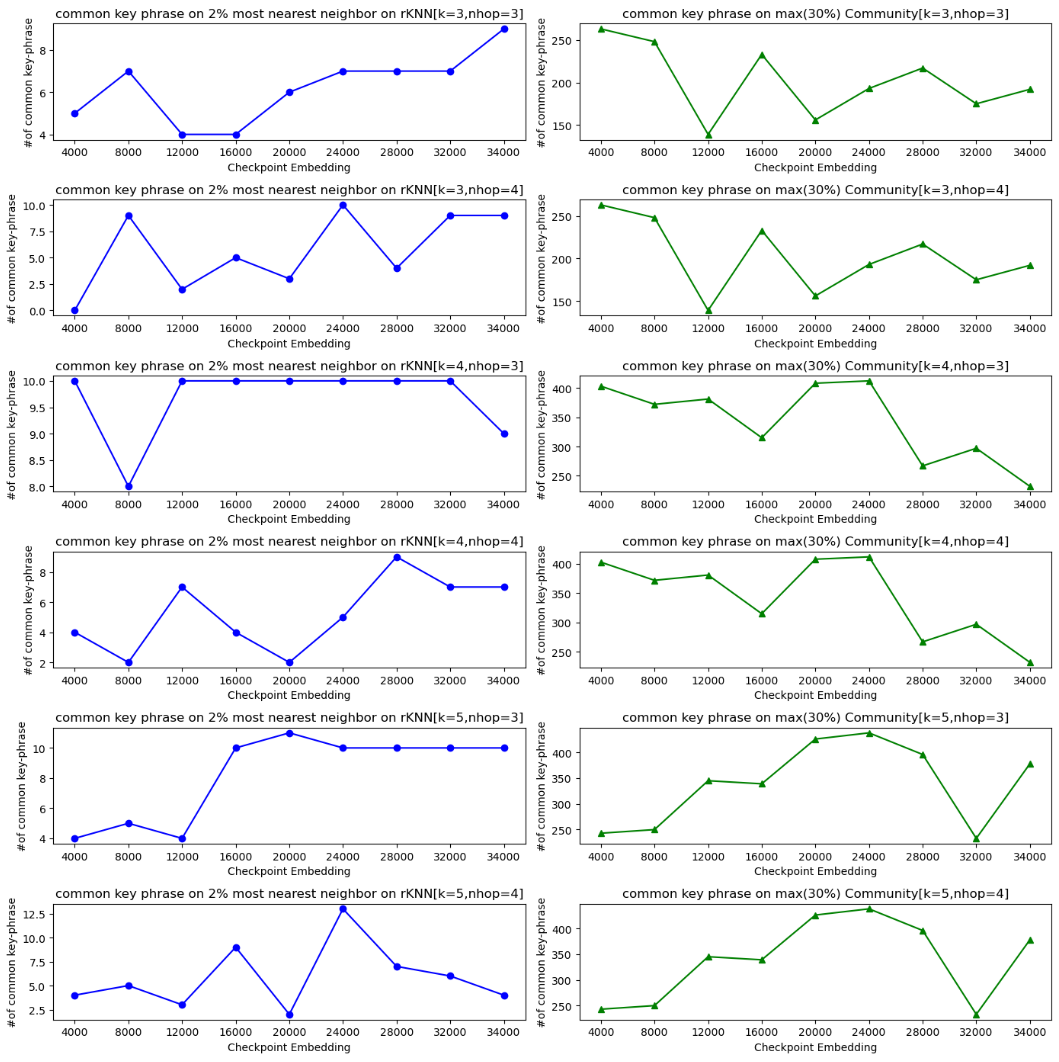


Figure 8: Experiment result for Longformer Model(2% most nearest neighbour and max(30%) community)

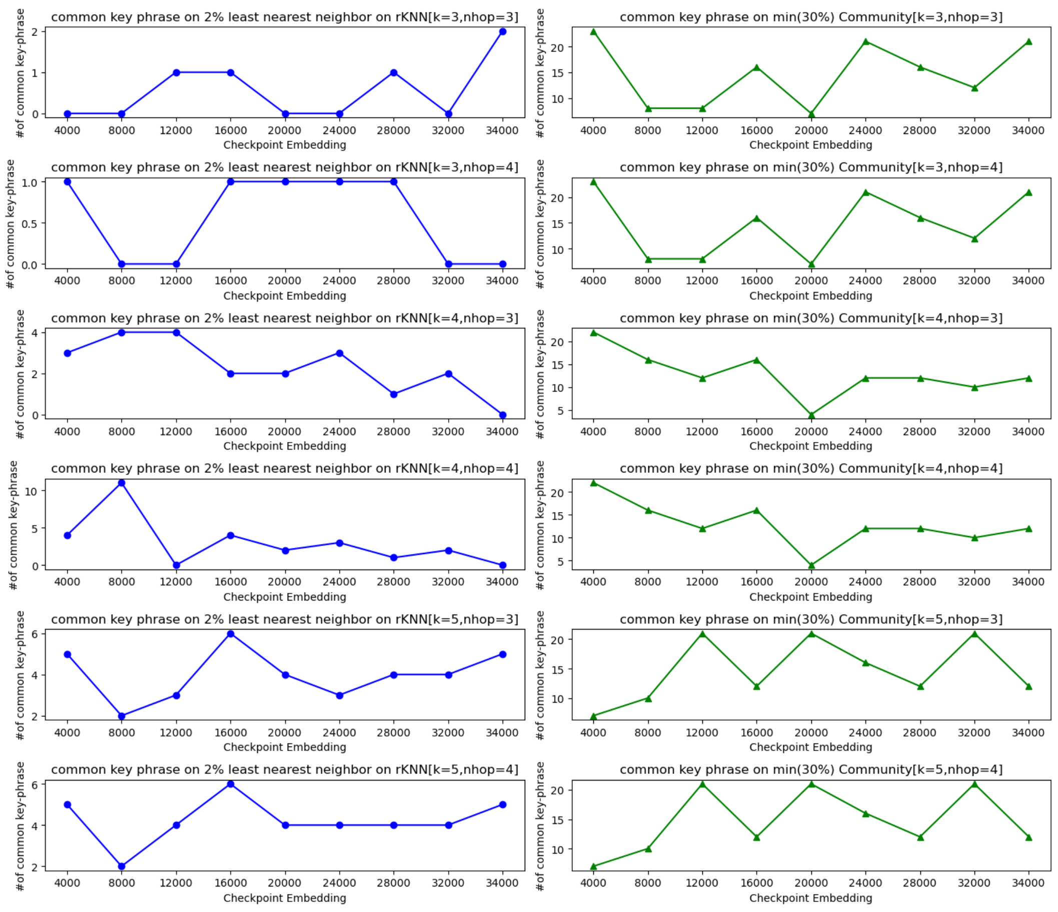


Figure 9: Experiment result for Longformer Model(2% least nearest neighbour and maxmin community)