university of groningen / faculty of science and engineering

Multi-Document Keyphrase Extraction

Bachelor's Project Thesis

Daniel Skala, s3953602, d.s.skala@student.rug.nl

Supervisors: Daniela Jašš (Slido/Cisco) & Dr. George Azzopardi (RuG) & Dr. Fadi Mohsen (RuG)

July 19, 2022

Abstract: Multi-Document Keyphrase Extraction (MDKE) is one of the fundamental problems withing Natural Language Processing (NLP). It is widely used in practice for tasks such as text summarisation, topic generation and clustering. One of the recent advances in MDKE is the creation of the *MK-DUC-01* dataset. Due to its novelty and lack of research on MDKE, we want to investigate the reproducibility of the performance of various KE algorithms. In addition, we propose two novel methods for keyphrase extraction built on top of TopicRank. The first algorithm 'SlidoRank' is asymptotically faster and more scalable due to the replacement of the slow topic graph generation and the PageRank algorithm used in TopicRank. The algorithm also outperforms TopicRank in terms of $F1@k$ scores tested on the *MK-DUC-01* dataset. The second proposed algorithm 'Embeddings' extends SlidoRank by semantic similarity of keyphrases. For a special configuration of hyperparameters, the Embeddings algorithm yields even better $F1@k$ scores than SlidoRank.

Contents

1 Introduction

Automatic Keyphrase Extraction (KE) is one of the fundamental problems within the field of Natural Language Processing (NLP). With the advancement of document digitalization, creation of large datasets and text processing, keyphrase extraction has gained its popularity for its vast applicability in real-world problems. Such use cases include extracting keywords from scientific papers, text summarisation, document indexing, or topic generation and clustering.

1.1 Problem definition and Motivation

An important factor for KE is the target document from which keyphrases should be extracted. In Single-Document Keyphrase Extraction (SDKE) the text is often centered around one or very few dominant topics while in Multi-Document Keyphrase Extraction (MDKE) the amount of dominant topics varies a lot more. Despite its ability to describe large sets of documents, the research on MDKE has always lagged behind SDKE. For the use cases with large number of long documents or high sub-topic variation, SDKE is no longer sufficient which makes this problem more appealing to research. (Shapira et al., 2021)

1.2 Main scientific challenges

One of the major recent contributions for MDKE evaluation is the creation of the *MK-DUC-01* dataset (Shapira et al., 2021) which opens new possibilities for the improvement/innovation of novel keyphrase extraction algorithms. A number of these will be also presented in this thesis, along with an attempt for reproducing the obtained $F1@k$ scores described in this paper (Shapira et al., 2021).

Furthermore, since this thesis will be done in cooperation with Slido (acquired by Cisco), we will drive our attention to the topic generation sub-task for data from the Slido events.

1.3 Aims & Objectives

First aim of this Thesis is to validate the relevance of the MK-DUC-01 dataset by reproducing the results from the scientific paper the dataset was published in (Shapira et al., 2021).

Second aim of this Thesis is to solve the questions-by-topics feature in Slido with two algorithmic approaches, test various configurations of their parameters and evaluate their performance and the keyphrases they yield in terms of F-scores on the MK-DUC-01 dataset. In the end, I will provide a parameter configuration that gives the best F-scores on the above mentioned dataset.

1.4 Overview of the proposed idea within Slido

Slido is aiming at making events/meetings more engaging by bringing the audience closer to the speaker through increased interaction. (Muthmainnah, 2019) During a Slido event, the clients (audience) can send questions to the host (speaker) and upvote them. There has been an ongoing attempt to analyze incoming questions and create a variety of features to improve the user experience.

One of these features is topic generation through keyphrase extraction which will help participants better navigate through the list of questions. Algorithms that solve this task will be the focus of this paper.

1.5 Report structure

This Thesis is divided into 13 sections. The first section is an introduction to the topic of Keyphrase Extraction and the context of Slido. Second section describes the state of the art literature on MDKE. This section is followed by the proposal section where I describe what is the main objective of the Thesis in greater detail. The Methods section describes how the datasets will be obtained and the fifth section Experiments and Results explains how the results were reproduced. Section seven is a thorough description about the TopicRank algorithm and in section eight I describe the first proposed algorithm. Section nine describes the Embeddings approach and section ten shows the results of the hyperparameter search. Thesis is summarized in the Conclusion section which is followed by the Future work and Acknowledgements section. In the Appendix I include a qualitative comparison between the state of the art and our proposed algorithms.

2 State-of-the-Art

2.1 Search Terms

To gather information about the state of the art research on Multi-Document keyphrase extraction, we have used a number of data repositories and databases such as IEEE-Xplore, Smartcat, and GitHub.

Below is a subset of example search queries used to obtain relevant papers, datasets, and other works:

- Single-Document keyphrase extraction
- Multi-Document keyphrase extraction
- Benchmark dataset for KE evaluation
- Algorithms on keyphrase extraction
- Supervised vs. unsupervised KE algorithms
- Keyphrase extraction using word embeddings
- Keyphrase extraction using term frequencies

2.2 Works on Multi-Document KE

One of the first works on MDKE was done by Khaled Hammouda, Diego N. Matute, and Mohamed S. Kamel (Khaled M Hammouda and Kamel, 2005). Their work is centered around extracting keyphrases based on word-stem overlaps between multiple documents. Their algorithm could accurately identify the most dominant topics using term frequencies of the shared word-sequences. They based their evaluation on comparing word-stem overlaps between the obtained keyphrases and the keyphrases retrieved by SDKE on individual documents. (Shapira et al., 2021)

Another work by Gábor Berend and Richárd Farkas (Berend and Farkas, 2013) was approached by performing SDKE on each document in the document set and then merging these lists of keyphrases together. In the merging process they also incorporated word embeddings and base knowledge from Wikipedia. Evaluation was done by comparing the cosine similarity of the system keyphrases and the ones provided from the scientific papers from ACL workshops (Ulrich Schäfer and Oepen, 2012). Similar approach was also performed by Farnoush Bayatmokou (Farnoush Bayatmakou and Mohebi, 2017; Shapira et al., 2021).

2.3 Evaluation of Multi-Document KE

Most SDKE works perform a comparison of the obtained keyphrases and the gold list of keyphrases. However, that is not the case for MDKE works since (until recently), no extensive benchmark dataset was available. Most of the previous works on MDKE have gone around this by evaluating against present and available SDKE datasets which are not that informative and appropriate for this task.

The most promising evaluation metric for MDKE algorithms is the standard $F1@k$ score which comprises both precision and recall in a harmonic mean. However, other metrics can be used, such as Mean Reciprocal Rank, Mean Average Precision, or the Normalised Discounted Cumulative Gain (Sun and Chi, 2020).

2.4 The state-of-the-art benchmark dataset for MDKE - MK-DUC-01

Recently, a number of researchers from Bar-Ilan University and UNC Chapel Hill have been working on a new dataset (including a literature review) for MDKE called *MK-DUC-01*. This dataset is based on the *DUC-2001* SDKE dataset (Wan and Xiao, 2008) from the domain of news articles. The dataset consists of 30 topics, each consisting of 10.27 related news articles on average. Every article, as well as every document set, was individually summarized by a number

of experts from the field. These summaries of different lengths were further processed and a list of relevant keyphrases was extracted. On average, there are 8.08 keyphrases per document (Wan and Xiao, 2008; Shapira et al., 2021) and 2.205 words per keyphrase.

To construct the benchmark dataset for MDKE, a number of refining procedures were applied. These procedures include automatic merging, reranking, deduplicating of keyphrases, computing a *word_score(w, t)* and using it to compute the document frequency in a specific document set. The resulting dataset was later manually refined and cleansed from low informative or synonymous keyphrases. This yielded a finalized *MK-DUC-01* dataset (Shapira et al., 2021) which will be investigated in this thesis.

2.4.1 Random sentences dataset

In my Thesis, I will also use the random sentences dataset mainly for time complexity evaluation. This dataset consists of 1800 random sentences retrieved from a Random Sentence Generator.(sen,) Since the sentences are random, there is no dominant topic present. The average number of tokens per sentence is 12.1125.

3 Proposal

Due to the novelty of the *MK-DUC-01* and lack of research on MDKE, it is crucial to investigate the recent papers and expose the datasets and algorithms to a stress test mainly in order to see if the results and methods are reproducible. Our work will be centered around the paper (Shapira et al., 2021) and we will specifically focus on reproducing the $F1@k$ score in Tf-Idf, TextRank, and TopicRank algorithms. The attempt to reproduce the existing results might strengthen or attenuate the reliability of the above-mentioned paper which is an important step in the state-of-the-art research on MDKE and its novel evaluation datasets.

3.1 Proposed algorithms

Furthermore, we also propose two new Multi-Document keyphrase extraction algorithms that are built on top of the current TopicRank algorithm.

The first algorithm is a modification of TopicRank which aims at speeding up the topic-creation process asymptotically by replacing the PageRank sub-algorithm. This algorithm is based on word-stem overlaps, jaccard distance, and agglomerative clustering.

The second algorithm is aiming for a better $F1@k$ by enhancing the previous algorithm by word embeddings generated by a language model (eg. sBERT, DistilBERT). With the use of the embeddings, the algorithm will introduce semantic similarity between keyphrases. We will further investigate the influence of various parameters (eg. the language model) on the performance of the algorithm.

These algorithms will be evaluated on the same *MK-DUC-01* dataset and their $F1@k$ scores will be compared to the scores of Tf-Idf, TextRank and the original TopicRank algorithm.

3.2 Research questions

This paper aims at scrutinizing the following research questions:

- Are the algorithms in the MDKE paper in (Shapira et al., 2021) reproducible?
- How do our novel algorithms compare to the current state-of-the-art algorithms with respect to speed performance and the quality of retrieved keyphrases (in terms of $F1@k$)?
- Is it possible to improve $F1@k$ scores by forming clusters of similar keyphrases and picking their representative keyphrase?
- How do various parameters influence the clustering process?

4 Methods

The primary idea of our proposed SlidoRank algorithm (together with Embeddings) is to replace the topic graph used in TopicRank by a faster and simpler frequency-based approach and together with other algorithmic improvements asymptotically increase the speed performance of the algorithm. Figure 4.1 shows a high-level diagram illustrating the structure of the proposed algorithm.

4.1 Overview

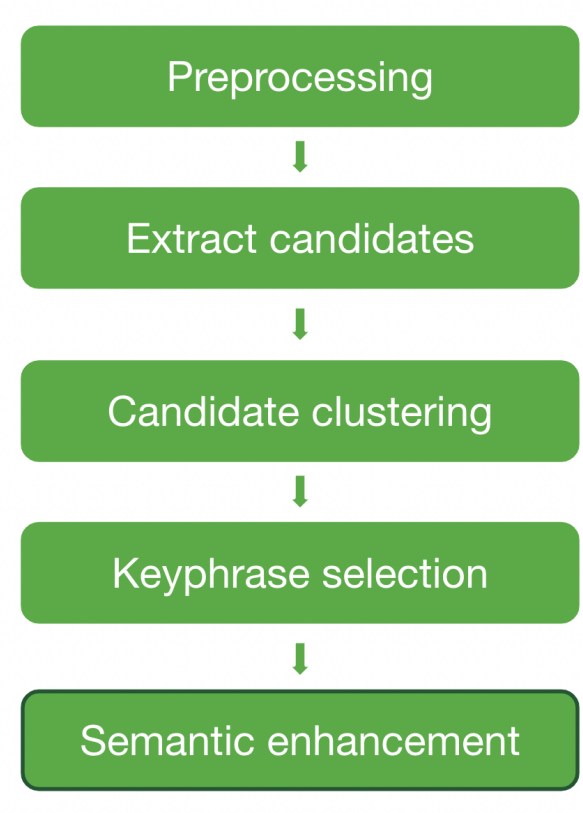

Figure 4.1: Schematic pipeline of the proposed algorithm

4.2 Technology

The project will be developed using the Python programming language with the use of pke which is a Python library for keyphrase extraction. Some of the methods in *pke* will be also used in the proposed algorithms as a basis.

This paper is centered around the following two datasets: The first one is a DUC 2001 dataset, which was released by NIST (National Institute of Standards and Technology) and includes the documents from which the keyphrases will be extracted. Furthermore, we will obtain the *MK-DUC-01* dataset containing the gold list of keyphrases which can be downloaded from GitHub.

This thesis will be made in cooperation with Slido (acquired by Cisco Systems, Inc.) I will be working on a Cisco Laptop (all speed performance graphs were performed on this laptop - MacBook Pro (16-inch, 2021), M1 chip, 16GB RAM) and the main supervision will also be from Cisco (more about supervision in the Planning section). All code-bases will be stored on a private personal GitHub repository. The code might become publicly available after the research is finalized and if there will be no legal issues.

5 Experiments and Results

When it comes to reproducing the results, I will mainly focus on the first column of the 'concat' part of the table 2 (see 5.1) from the MK-DUC-01 paper (Shapira et al., 2021). The Unigram-F1@k is an important metric but due to time constraints I will not include it in my Thesis and instead the focus will be on the standard F1@k.

Although the authors uploaded code for evaluation on their GitHub repository (this was done very recently and after my topic was chosen and approved), I will write my own evaluator from scratch and compare my results with the author's evaluator. To do so, we need to ensure that:

- The input text is the concatenation of all the documents and passed as one string
- We use the F1@k and not the unigram-F1@k scores
- The output set of keyphrases is truncated at 20
- The golden list of keyphrases is truncated at 20
- When encountering a substitute cluster (which is a collection of very similar keyphrases - part of the golden list) we need to select the first keyphrase for evaluation and ignore the rest. (*Note: Although it is explained in the paper, it is not evident which strategy was used in the construction of Table* 2.)
- We are evaluating the stemmed form of keyphrases

We start by loading the appropriate datasets, in this case *MKDUC01_keyphrases.json* for the golden list of keyphrases and the *MKDUC01.json* for the documents. To process the golden set we construct a dictionary of topic IDs as keys and list of keyphrases as values. We also "flatten" the list by selecting the first phrase in case of substitute clusters. Cropping the list at 20 keyphrases yields the final dataset.

Next, we load the documents dataset, iterate over every topic ID and join the corresponding documents into one concatenated string *docs_concat*.

Now we can test any algorithm by passing the concatenated string of documents into the it (either as a string or a list, depends on the algorithm). Algorithm 5.1 shows testing SlidoRank (Mk. I).

	Concat								*Merge*							
	F1@k				**unigram-F1@k**				**F1@k**				**unigram-F1@k**			
Algorithm	**1**	**5**	**10**	**20**	**1**	**5**	**10**	**20**	**1**	**5**	**10**	**20**	**1**	**5**	**10**	**20**
Tf-Idf	0.32	1.87	4.44	5.67	4.56	16.77	26.58	35.11	0.63	1.60	2.44	4.50	4.52	17.04	24.04	31.00
KPMiner (El-Beltagy and Rafea, 2009)	1.27	4.80	7.56	9.68	5.52	21.27	30.55	34.88	1.59	5.34	7.56	10.84	5.39	19.45	28.85	38.93
YAKE (Campos et al., 2020)	2.54	7.20	10.67	13.17	6.24	24.80	33.30	36.04	2.86	5.87	8.23	10.84	7.06	25.75	33.87	37.72
TextRank (Mihalcea and Tarau, 2004)	0.63	4.00	4.67	8.01	10.11	28.25	32.03	30.72	2.54	9.88	13.79	17.17	9.08	29.66	39.90	41.28
SingleRank (Wan and Xiao, 2008)	0.95	5.08	6.90	12.01	12.51	29.52	32.81	31.98	2.86	8.80	14.23	18.51	9.26	28.96	38.56	42.01
TopicRank (Bougouin et al., 2013)	1.59	6.40	9.33	11.33	5.47	18.69	28.75	36.72	4.13	10.94	16.01	18.68	6.75	25.22	39.24	44.88
TopicalPageRank (Sterckx et al., 2015)	1.27	5.61	7.79	13.35	12.16	29.81	33.67	32.44	2.86	9.08	15.56	19.84	9.06	28.82	39.11	42.41
PositionRank (Florescu and Caragea, 2017)	1.90	8.28	11.80	16.35	9.31	27.09	32.20	33.22	3.17	8.53	15.56	19.51	8.44	27.50	39.20	44.11
MultipartiteRank (Boudin, 2018)	1.27	6.13	10.23	12.17	6.04	20.08	30.12	38.70	4.13	11.21	17.34	21.34	7.64	25.63	39.55	45.61
CollabRank (Wan and Xiao, 2008)					-				2.86	9.61	14.9	17.84	9.37	28.03	37.68	41.26
(Bayatmakou et al., 2017) [multi-doc]					-				0.09	0.93	1.46	1.88	1.84	7.74	11.69	15.18

Table 2: Results on various KE algorithms tested with the *Trunc-20* version of our MK-DUC-01 dataset. In *Concat* mode all topic documents are concatenated as a single text input, and in *Merge* mode algorithms are run on individual documents after which keyphrase lists are merged and reranked. The bottom two algorithms are multi-document based KE algorithms, and work in *Merge* mode only.

Figure 5.1: Table from the paper MK-DUC-01 paper (Shapira et al., 2021)

Algorithm 5.1 Testing SlidoRank (Mk. I)

Require: *docs_concat*
  $kps \Leftarrow SlidoRank(docs\_concat)$
  $kps\_phrases \Leftarrow$ *obtain only the phrases*
  store the generated *kps_phrases*

At the end, we store the generated keyphrases in a dictionary with topic IDs as keys and generated list of keyphrases as values. Finally, we pass this dictionary into the evaluator function.

Function $compute\_f1()$ takes as parameters the golden list of keyphrases, the generated dictionary of keyphrases and returns the final $F1@k$ scores for the defined values of k. For every topic, we truncate the predicted keyphrases at 20 (the golden list is already truncated), stem both predicted and golden list and compute the $F1$ score $F1 = \frac{2*p*r}{p+r}$ where p stands for precision and r stands for recall. In the end we take the average of all $F1$ scores for every topic which yields a final value for some k. We repeat for $k = \{1, 5, 10, 20\}$.

5.1 Results

We test *Tf-Idf*, *TextRank*, and *TopicRank* with our evaluator and obtained the results depicted in table 5.1.

All f-scores for all three algorithms from table 5.1 are identical to the scores obtained by the author's evaluator (eg. $F1@5$ for TextRank). However, most of the values from table 5.1 differ from the values in figure 5.1 by $1-2$ percentage points. These differences might be caused by different versions of algorithms used (as the *pke* library is an open-source library).

6 Keyphrase Extraction methods

There are three main approaches of extracting important keyphrases from a body of text:

1. **Supervised** methods approach KE as a classification task (Bulgarov and Caragea, 2015). Such tasks require large labeled datasets with document-keyphrases training samples, a neural network model (eg. MLP(Pal and Mitra, 1992), Bayes(Rish et al., 2001)) and a training and testing process. Although supervised learning has recently become increasingly popular due to availability of large datasets in various fields, large document-keyphrases datasets are still often difficult to find or use.

2. **Unsupervised** approaches generally share the following structure:

   (a) Pre-process document

   (b) Generate candidates from cleansed document

   (c) Score candidates (determine their importance)

   (d) Candidate post-processing (eg. deduplication)

   (e) Ranking and final keyphrase selection

   Popular unsupervised algorithms include *Term Frequency Inverse Document Frequency (TF-IDF)(Christian et al., 2016)*, *Rapid Automatic Keyword Extraction (RAKE)(Rose et al., 2010)* or *Yet Another Keyword Extractor (YAKE)(Campos et al., 2020)*.

   Some unsupervised methods approach the task using an Encoder-Decoder neural architecture although these architectures often find their use in text summarisation or

F1-scores for the original algorithms

Algorithm	$F1@1$	$F1@5$	$F1@10$	$F1@20$
Tf-Idf	0.32	2.67	5.56	6.5
TextRank	0.63	4.0	5.33	9.17
TopicRank	0.95	6.67	9.56	11.67

Table 5.1: Obtained $F1$ scores for Tf-Ids, TextRank and TopicRank using the author's evaluator. The values are in %. Some scores are identical but some differ in $1-2$ percentage points.

compression.(Hinton, 2012)

A subcategory of unsupervised methods are *Graph-Based* approaches that build a fully connected graph from candidates and use a graph algorithm to determine their importance / scores (eg. PageRank (Xing and Ghorbani, 2004)). One of the first graph-based algorithms was introduced in 2004 by Mihalcea and Tarau (Shobha S. Raskar, 2014) under the name TextRank. TextRank builds a graph where two words are connected if they occur within the same window of words in a document. The importance of these words is then computed using a random walk algorithm.

There exist a number of TextRank variations that use different graph-building algorithms and different scoring methods claiming to have outperformed TextRank. These variations include SingleRank, PositionRank, MultipartiteRank, or TopicRank. In this Thesis, we will investigate TopicRank since it serves as a backbone for our proposed algorithms.

7 TopicRank

TopicRank is a popular graph-based unsupervised keyphrase extraction algorithm. It improves TextRank by building a graph of topics (instead of graph of all candidates) which are constructed using agglomerative clustering. TopicRank is a recent algorithm proposed by Adrien Bougouin, Florian Boudin, Béatrice Daille in 2013 in this paper (Adrien Bougouin, 2013). (One of the authors (Florian Boudin) is also the main contributor to the *Python Keyphrase Extraction - pke*, a very popular Python library used for NLP tasks.)

In their paper they claim to have significantly outperformed the state-of-the-art TextRank on three out of four datasets. These results persuaded us to use TopicRank as the basis for our proposed algorithms used in the Slido's questions-by-topics feature.

7.1 Implementation of TopicRank

In order to understand our proposed algorithms, it is important to first understand the implementation details of TopicRank. Hence, I will dedicate this section to explaining how the algorithm works in detail, what are its components, structure and its strengths and weakenesses.

As stated earlier, the abstract structure of TopicRank follows the standard graph-based pipeline pattern ($Preprocessing \rightarrow Candidate\ Extraction \rightarrow (Candidate\ Clustering) \rightarrow Graph - Based\ Ranking \rightarrow Keyphrase\ selection$). In the following subsections I will describe the workings and implementation of each of these stages. I will use the TopicRank implementation available in the *pke* library written by Florian Boudin.

7.1.1 Preprocessing

Like in any other NLP task, we first start with cleaning the original document from non-descriptive words (stopwords) such as 'and', 'the' or punctuation. For this the author uses the *load_document*() function which takes the path to a file, a list of stopwords, type of normalization and performs the preprocessing. The implementation of the *load_document*() function is non trivial but I will not go into details here since it is a standard preprocessing step done before any KE algorithm can be applied. The implementation can be found in *pke.base*.

7.1.2 Candidate Extraction

Candidate extraction is a process of extracting potential keyphrases from a body of preprocessed text. To do this, we first determine the Part-Of-Speech tags (POS tags) such as NUM, ADV, $PROPN$ or others for every token in the text. Next, we iterate over these tokens and select the longest sequence of tokens with the predefined POS tags. (The default POS tag set is generally $\{NOUN, PROPN, ADJ\}$ since nouns, proper nouns and adjectives tend to be most descriptive of a text (Hulth, 2003). However, sometimes this set includes also NUM.) These sequences are further processed and discarded if they happen to be too long/short, or include some unwanted tokens. (POS tagging is a non-trivial task and some POS tagging models can tag certain tokens in some contexts inaccurately.) This is done using the $candidate\_filtering()$ method.

Algorithm 7.1 Candidate Extraction

Require: document
  $POS \Leftarrow \{'NOUN', 'PROPN', 'ADJ'\}$
  $seq = []$
  for every tuple (token, POS tag) in document **do**
    if POS tag in POS **then**
      $seq \Leftarrow add\ token$
    end if
  end for
  $candidate\_filtering()$

7.1.3 Candidate Clustering

Candidate Clustering is where TopicRank starts to differ from TextRank. The motivation for clustering is that candidates within the same topic tend to share similar tokens/words/stems. For example an article about tunnels can include candidates such as: "tunnel, tunnels, underground tunnels, tunnelling, etc." In TextRank, these candidates would be treated separately whereas in TopicRank, these candidates are merged into one cluster which is represented by one candidate. This is where the name TopicRank comes from - it is an important step which is performed before the topic graph is created.

Before the candidates can be clustered, they first have to be vectorized. This is a topic in itself and clustering methods will be described more deeply in later sections.

The TopicRank implementation uses one hot encoding of the candidates which are then clustered using agglomerative clustering with average linkage and jaccard distance for the distance matrix. The cluster hierarchy is then cut using $fcluster()$ method from scipy using a threshold of 0.74 which leads to cluster formation. The pseudocode in 7.2 describes this process.

Algorithm 7.2 Topic Clustering

  $candidates, X \Leftarrow vectorize\_candidates()$
  $Y \Leftarrow pdist(X, 'jaccard')$
  $Z \Leftarrow linkage(Y, 'average')$
  $clusters \Leftarrow fcluster(Z, 0.74, 'distance')$
  for id in range(max(clusters)) **do**
    add candidates to corresponding clusters
  end for

7.1.4 Graph-Based Ranking

The next step is the creation of a fully connected graph using the topics obtained from the previous step. Topics are represented as nodes where the weighted edges are the "semantic relations" between the topics. However, this semantic relation is not based on feature vectors but merely on words occurring within the same window of tokens in the document. Formally speaking the weight $w_{i,j}$ of an edge is:

$$w_{i,j} = \sum_{c_i \in t_i} \sum_{c_j \in t_j} dist(c_i, c_j) \tag{7.1}$$

$$dist(c_i, c_j) = \sum_{p_i \in pos(c_i)} \sum_{p_j \in pos(c_j)} \frac{1}{\|p_i - p_j\|} \tag{7.2}$$

Where t_i, t_j are topics, $dist(c_i, c_j)$ is the distance between two candidate keyphrases and $pos(c_i)$ is a set of all the positions of the candidate in a sentence. (Adrien Bougouin, 2013)

After the graph is constructed and the "semantic" weights are set, now we determine the significance of the topics using the PageRank algorithm (although in the original TopicRank paper, authors use TextRank model proposed by Mihalcea and Tarau, 2004, that is based on the concept of "voting"(Adrien Bougouin, 2013; **?**)).

Although PageRank was originally designed for ranking web pages based on the links between them, it is also used in this scenario to rank the topics. The PageRank algorithm first computes the transition matrix from the graph and iteratively applies it on the nodes and edges until the ranks converge. The default number of iterations is set to 100. The output of the algorithm is the set of topics with their scores (higher being more significant).

The graph representing "semantic" connections together with the PageRank algorithm works very well in determining the significance of the topics in the document. However, the construction of the

fully connected graph, determining the edges and ranking of the topics using PageRank turns out to be the slowest, most computationally expensive process in the pipeline making it almost unusable for very long documents.

7.1.5 Keyphrase Selection

Last step in the TopicRank pipeline is final keyphrase selection. At this point we have a set of topics of certain significance score where each topic consists of one or more candidates. To produce the final set of keyphrases, one representative candidate (or token) needs to be picked. Although this seems like a trivial problem, it is one of the most difficult problems in the entire pipeline; which token/candidate represents a cluster the best.

There are a number of strategies that can be applied here and I will dedicate one section to explaining what are the other options as well as some experimental results showing the impact of these strategies on the F1-score. The authors of the TopicRank paper have tackled the task accordingly: "To find the candidate that best represents a topic, we propose three strategies. Assuming that a topic is first introduced by its generic form, the first strategy is to select the keyphrase candidate that appears first in the document. The second strategy assumes that the generic form of a topic is the one that is most frequently used and the third strategy selects the centroid of the cluster. The centroid is the candidate that is the most similar to the other candidates of the cluster." (Adrien Bougouin, 2013)

Clearly, these strategies are context-dependent and for the Multi-Document scenario of the Slido's questions-by-topics context, different strategies have to be applied.

8 TopicRank without PageRank - SlidoRank

The primary motivation for our first proposed algorithm is the questions-by-topics feature in Slido. To reiterate, the specific context Slido operates within is the following: during an event participants send questions to the host of the event as a part of the Q&A session. For large events with hundreds or thousands of participants, the amount of questions sent to the host is very large. To prevent multiple participants asking the same question, one of the strategy is the ability to upvote questions that participants like. This upvote feature functions as a user-ranking system which builds a hierarchy of questions that participants want to ask. (Slido also offers more advanced sentence similarity feature but that is not of primary concern for this Thesis.)

The problem arises when the amount of questions present in the Q&A section is too large. If eg. 500 questions are present in an event, participants will read only a tiny fraction of questions which will lead to less upvotes, and more duplicate questions being asked. To improve user experience and attenuate question redundancy, Slido offers a questions-by-topics feature. This feature extracts keyphrases from all the questions and clusters them into topics which are then displayed to the user. The idea was first introduced in one of the Slido hackathons and incrementally developed by the Data Team.

The idea began with the TopicRank implementation but we quickly realized that for our use case of millions of simultaneous events containing hundreds or thousands of participants all over the world, TopicRank was too computationally intensive and slow. Figure 8.1 shows TopicRank run times for the $MK - DUC - 01$ dataset on up to 1140 sentences. We can already observe that on 342 sentences, TopicRank takes 5 seconds and grows approximately with the square of the number of sentences.

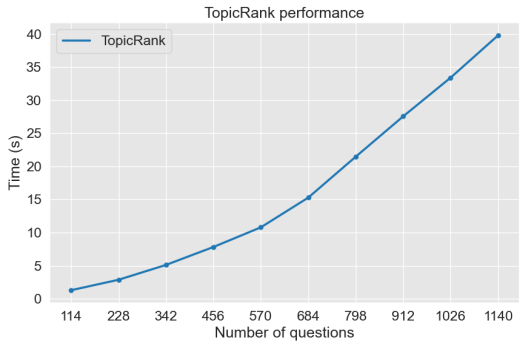

Figure 8.1: Performance of the TopicRank algorithm tested on the $MK - DUC - 01$ dataset

8.1 Alternative to the topic graph and PageRank

Our proposed algorithm is more complex than TopicRank and has been incrementally developed, speeding up various parts of the pipeline. To fully understand the workings of the algorithm, I will explain step-by-step which changes were made and how these changes influence the overall performance.

To further refer to our first proposed algorithm I will simply use a name *SlidoRank*.

8.1.1 Preprocessing and Candidate Extraction

The first two steps, preprocessing and candidate extraction, of the pipeline remain almost identical to TopicRank: candidates are extracted as longest sequences of tokens with predefined POS tags ($\{'NOUN','PROPN','ADJ'\}$).

8.1.2 Candidate Clustering

Just like in TopicRank, candidates are first vectorized using a sparse matrix and clustered using agglomerative clustering based on the jaccard distance of stemmed tokens. The resulting cluster hierarchy is then cut using the default threshold of 0.6 (further experiments with the threshold are explained in section 8.2.2).

In TopicRank, the resulting clusters are then converted into a fully connected graph and scored. However, we have decided to skip this stage and simply score the clusters according to the number of candidates they include.

Recall that the context of Slido is an event with a set of questions for the host. Take for example a company meeting about inflation and a potential pay raise. Let us also assume that the clusters in the event are as follows: [salary, salary increase, higher salary, current low salary], [crysis, economic crysis], and [high inflation]. Lastly, to make matters simpler, let us assume that one candidate corresponds to exactly one unique question (this is rarely the case). From this setting it is clear that the cluster about salary is the most dominant one since most of the questions in the event are about salaries. Hence this cluster will obtain a higher score than the remaining clusters about crysis and inflation.

8.1.3 Keyphrase Selection

The keyphrase selection approach used in TopicRank takes into consideration the order of candidates - candidates appearing earlier in text will be prioritized. That is not the case in our Multi-Document setting. We look at the data as a set of questions present at an event at a point in time. The order of asked questions is not relevant at all for which topics are most dominant in an event.

For the first iteration of SlidoRank we will be using the TopicRank's approach of selecting final keyphrases (I will further refer to this as "representative selection") but further experiments with selecting the representative of a cluster will be described in section 8.2.6.

8.1.4 Evaluation Benchmarks - generation 1

According to our hypothesis, skipping the PageRank algorithm will have a positive influence on the overall time complexity. Since the scoring of clusters also differs, we expect a difference in the $F1@k$ scores. Figures 8.2 and 8.3 show the speed performance of the entire pipeline (from preprocessing to final keyphrase selection).

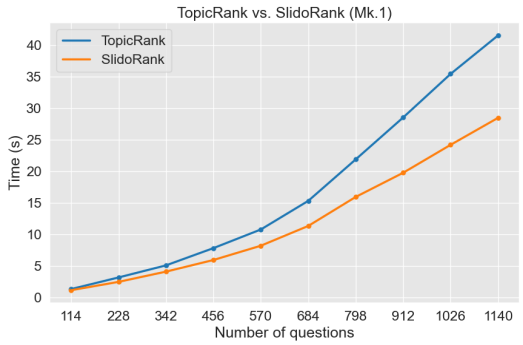

Figure 8.2: TopicRank vs. SlidoRank speed comparison on the $MK-DUC-01$ dataset.

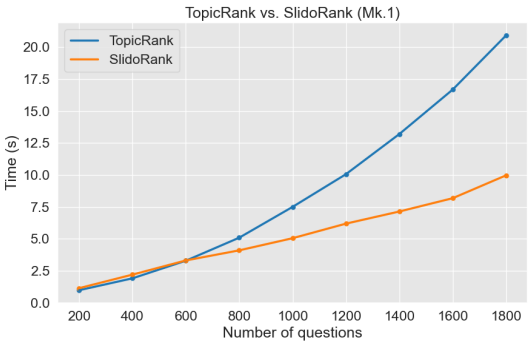

Figure 8.3: TopicRank vs. SlidoRank speed comparison on the dataset of 1800 randomly generated sentences.

Figures 8.2 and 8.3 show that skipping the topic graph and PageRank algorithm indeed significantly improves the computational complexity and hence speeds up the pipeline.

8.2 SlidoRank, Mk. II

To further improve the speed performance and the set of resulting keyphrases (as well as code quality and scalability) we introduce a number of improvements at various stages of the pipeline which will be explained in the following sections.

8.2.1 Reject one-token candidates

We introduce a small change in the *get_candidates()* function and allow only candidates whose lexical form is greater than 1. The reason for this is to keep more contextual information which candidates with a very short lexical form (only one word) do not offer.

8.2.2 Optimizing the clustering threshold using Silhouette scores

Clustering is an ill-defined problem; most real-world tasks include noise or ambiguity which make problems like clustering very difficult. There is a number of clustering algorithms, starting with simple k-means, through more advanced hierarchical clustering to very advanced density-based approaches like HDBSCAN.

TopicRank as well as SlidoRank use agglomerative clustering which iteratively merges two nearest data points or clusters and returns a hierarchy of clusters (where at the bottom of the hierarchy are n clusters for n data points and on top is 1 cluster for n data points). In order to obtain the desired clusters we have to specify a level of this hierarchy, often called a threshold. This threshold was set to a default value of 0.6.

The problem is that a threshold of 0.6 is not the best possible threshold for every event; sometimes a threshold of 0.7 or 0.5 would yield better results. In order to investigate the "goodness" of clustering we can use a metric such as Silhouette score. Silhouette score measures the inter and intra-cluster distances and returns a score between -1 and 1. The score is 1 when the distances between clusters are large and distances between data points within their cluster is small.

To search for an optimal threshold we cut the hierarchy at different levels and compute the silhouette score. Then, from all the scores we pick a threshold that leads to the highest silhouette score. The algorithm 8.1 outlines this process.

Algorithm 8.1 Get optimal threshold

Require: distance matrix X and the cluster hierarchy Z
  $silhouettes \Leftarrow []$
  $thresholds \Leftarrow [0.0, 0.01, 0.02, \ldots, 0.99, 1.0]$
  for every threshold t **do**
    $clusters \Leftarrow fcluster(Z, t)$
    $silh \Leftarrow silhouette\_score(X, clusters)$
    add silh to silhouettes
  end for
  return threshold with maximum silh. score

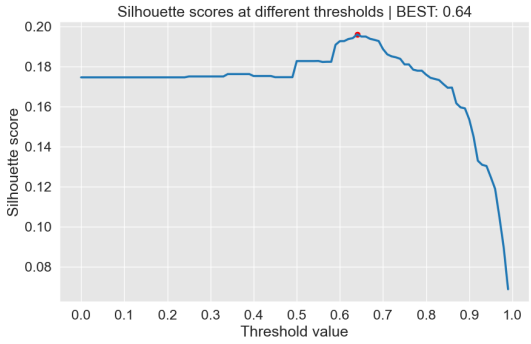

Figure 8.4: Silhouette scores at different threshold values for questions from Elon Musks Slido event.

Table 8.1 presents the impact of the threshold value on the overall f-score tested on the MK-DUC-01 dataset.

As we see from table 8.1, the clustering threshold has a significant influence on the f-scores. The threshold that gives the best overall results is 0.5. However, what comes as a surprise is that the worst performing threshold (among the ones I tested) is the "optimal" one based on the silhouette score. We can safely conclude that optimizing the inter-intra cluster distances via silhouette scores is a bad strategy in obtaining the highest f-scores on the MK-DUC-01 dataset. The reason for this is not self-evident and investigating this behaviour is a good candidate for future work 12. For the remaining benchmarks I will be working with $t = 0.6$.

8.2.3 Redefinition of candidates through "Occurrences" class

The first SlidoRank implementation had three important classes:

- **Question**: A class holding a question entity including *question_id*, list of candidates, text, length, POS tags and other attributes.
- **Candidate**: A class for a candidate entity holding surface form, lexical form, the position in a question and other attributes.
- **Cluster**: A class having an ID, candidates, all associated questions and the representative.

In SlidoRank II the Question and Candidate classes were transformed into @*dataclass* objects with have defined fields which are filled in later in the process after instantiation. Moreover, the Candidate class now serves as a wrapper class for *OccurrenceInQuestion*. Since every candidate is a subsequence of a specific question(s). It has an *occurrences* field which holds question IDs of questions in which the candidate occurs (including

Thresholds for SlidoRank (Mk. II)

Thresholds	$f1@1$	$f1@5$	$f1@10$	$f1@20$
$t = 0.5$	**2.86**	**8.81**	13.8	14.85
$t = 0.54$	2.54	8.54	**14.02**	**15.01**
$t = 0.55$	2.54	8.54	13.81	15.01
$t = 0.6$	2.54	8.54	13.13	15.01
$t = 0.7$	0.95	4.53	6.0	6.67
$t = 0.74$	0.32	3.2	4.44	4.83
$t = t_{opt}$	0.32	4.0	6.22	6.0

Table 8.1: F-score comparison of SlidoRank (Mk. II) with different thresholds tested on the MK-DUC-01 dataset. t_{opt} is the optimal threshold found by the silhouette score.

its offset).

The Cluster class now only includes a list of candidates, all questions associated with the candidates, frequency (length of the all questions list), and a representative.

8.2.4 Improved jaccard distance computation

Another significant improvement is the replacement of the standardized function *pdist*() from the *scipy* library (*scipy.spatial.distance*). Jaccard distance computes the similarity of two groups (sets) by the ratio of their intersection over their union:

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

In Python code, this can be implemented as follows:

$$\frac{len(set(A).intersection(B))}{len(set(A).union(set(B)))}$$

. However, for large datasets, this computation is too slow.

An article by Roy Hung (hun,) about *Jaccard's Index in Practice* provides a guide in significantly speeding up this computation using linear algebra. In short, the core idea is to reformat the dataset and represent it in a sparse matrix form X, compute XX^T as the numerator and use the identity $|A \cup B| = |A| + |B||A \cap B|$ for the denominator. Figure 8.5 depicts a difference of the first iteration of SlidoRank with the standard *pdist*() function and the improved one.

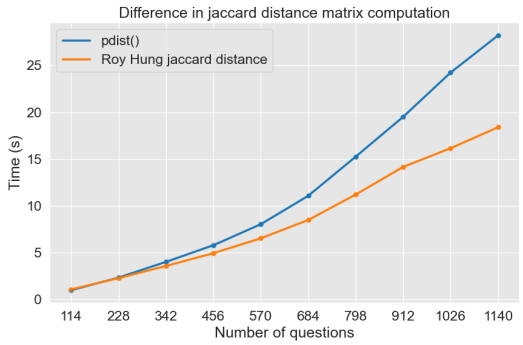

Figure 8.5: Speed performance comparison of two jaccard distance functions tested on the MK-DUC-01 dataset.

8.2.5 Faster cluster formation

The $fcluster()$ method returns a list of cluster IDs, the actual cluster objects still have to be constructed. The TopicRank implementation forms the clusters in quadratic time complexity in the number of clusters. The pseudocode is shown in 8.2.

Algorithm 8.2 Make clusters in $O(n^2)$

Require: list of clusters from $fcluster()$
  for every cluster ID **do**
    $cand\_dict \Leftarrow dict()$
    for j in range(len(clusters)) **do**
      if clusters[j] == cluster_id **then**
        add candidates to cand_dict
      end if
    end for
    add to cluster_objects
  end for

The improved version uses only one for loop thanks to the use of Python's dictionary objects as shown in 8.3. This asymptotically speeds up the cluster formation process.

Algorithm 8.3 Make clusters in $O(n)$

Require: list of clusters from $fcluster()$ and $candidates$
  $cands \Leftarrow candidate\ keys$
  $cand\_objects \Leftarrow candidate\ objects$
  for $cand\_ID, cluster\_ID$ in enum(clusters) **do**
    $cand \Leftarrow cands[cand\_id]$
    $obj \Leftarrow cand\_objects[cand\_id]$
    $clusters\_dict[cluster\_id][cand] \Leftarrow obj$
  end for

Figure 8.6 depicts the dramatic asymptotic speed up from quadratic to linear time complexity.

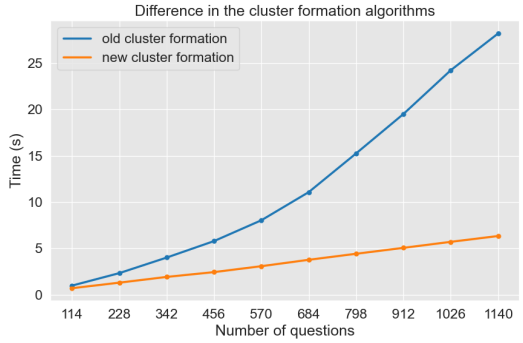

Figure 8.6: Speed performance comparison of two cluster formation algorithms tested on the MK-DUC-01 dataset (with the faster jaccard computation).

8.2.6 Representative Selection

Representative selection is s non-trivial task which goes as follows: given a set of candidate keyphrases, which token or set of tokens best describes the entire group of keyphrases? As an example, take the following set {methane, aerosol, gas, hydrogen, electricity}. One might say that this cluster can be best described as *power sources* since electric cars are becoming very prevalent, hydrogen engines are being researched as well and the remaining chemicals can be used in some combustion engine. However, we can also describe the cluster as *chemicals* or *fuel*. The selection of the representative is dependent on the context of the entire event. What complicates this task even more is the idea of selecting a representative that is not present in the set itself.

Luckily, since our clusters are based on word-stem overlap using the jaccard distance, the stems of the candidates (at least some of their tokens) will mostly remain the same. Hence our clusters might look as follows: {tunnel, tunnelling, underground tunnels, ...}. This nice property of "shared stem" allows us to pick a representative keyphrase from the set itself - which is also done in both TopicRank and SlidoRank algorithms.

The original TopicRank solution uses two heuristics for representative selection: for each cluster either pick the first occurring candidate or apply a more advanced strategy of choosing the most frequent candidate in the set and selecting the first occurring one (if there are more candidates with the same frequency).

Both TopicRank approaches take into consideration the order in which candidates are occurring (their offsets) in text. As previously stated, this heuristic is not important for our use case; the order of occurrence does not matter for a set of asked questions. The descriptive ability of a cluster is the determining factor since the representatives are the first thing that the user sees.

Our representative selection algorithm first obtains subsequences sorted by the number of tokens and then checks which subsequence is in the majority of other candidates. That candidate is selected as a representative of a cluster 8.4. In contrast to other representative selection algorithms (like the one in TopicRank), this approach tends to prefer single tokens over multi-word phrases.

Algorithm 8.4 Get Representative

```
rep ⇐ None
if there is only 1 candidate then
    select that candidate
else
    subseq ⇐ get_candidate_subsequences()
    for sub_stem, sub_lex_forms in subseq do
        if sub_stem in majority then
            rep ⇐ max(sub_lex_forms)
        end if
    end for
end if
if rep is None then
    rep ⇐ most_frequent_candidate()
end if
```

8.2.7 Comparison of different representative selection algorithms

Since the final keyphrase selection has a strong influence on the f-scores I provide a comparison of different - basic to complex - representative selection algorithms in table 8.2.

The results in table 8.2 indicate that representative selection algorithm does have an influence on the f-scores. Despite the differences being small, our representative selection algorithm outperforms the remaining three (naive) approaches at F1@10 and F1@20.

8.2.8 Final results

To conclude, implementing "TopicRank without PageRank" and optimizing the time complexity of various stages of the entire pipeline led to an algorithm that not only outperforms the current state-of-the-art TopicRank on every F1@k metric but is also asymptotically faster; making it practical for large-scale purposes. The above mentioned improvements allowed us to scale up the questions-by-topics feature in Slido and offer the functionality for events of "arbitrary" sizes (see figures 8.7 and 8.8).

However, I would like to emphasize that the MK-DUC-01 dataset does not match the Slido use case entirely. Therefore, building the entire pipeline **only** on the basis of the f-scores from this dataset is very risky since it might not "blend in" with the Slido use case well (users might simply dislike it). The MK-DUC-01 dataset might give us an indication of which parameters are worth scrutinizing and optimising but we have to be aware of the danger of over-fitting to this dataset.

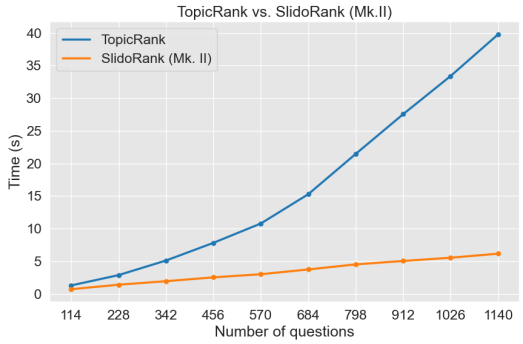

Figure 8.7: Speed performance TopicRank vs. SlidoRank (Mk. II) tested on the MK-DUC-01 dataset

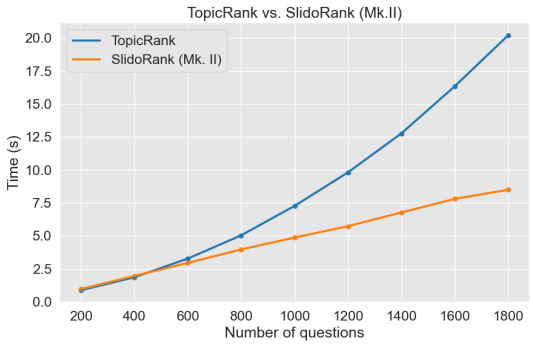

Figure 8.8: Speed performance TopicRank vs. SlidoRank (Mk. II) tested on the dataset of 1800 random sentences

9 The Embeddings algorithm

Although being fast and scalable, SlidoRank has a fundamental limitation - it does not group together semantically similar candidates. TopicRank is trying to go around this by applying a heuristic that "words occurring close to each other often have similar semantic meaning" in their topic graph. Although this heuristic might hold in some scenarios, it is obvious that some words can have very similar meaning while occurring in completely different contexts. To solve this issue, we have to introduce the concept of word embeddings.

9.1 Word Embedding

Word embedding is an abstract representation of a word as a vector. The simplest way to embed a word is an approach called "one hot encoding"(Rodríguez et al., 2018). Given a vocabulary (list of words) or length n, every word in the vocabulary will be attributed with a vector of zeroes with a single 1 at the index at which the word occurs withing the vocabulary. While this

F-score comparison of different representative selection algorithms

Representative	$f1@1$	$f1@5$	$f1@10$	$f1@20$
Our	2.54	8.54	**13.13**	**15.01**
First Candidate	**2.86**	**9.34**	12.67	14.5
Random	1.9	7.48	11.56	12.84
Most frequent	2.54	7.744	12.68	13.675

Table 8.2: Our algorithm is described in section 8.2.6. 'First candidate' algorithm selects the as representative the first candidate in the candidate list (candidates are appended in order in which they appeared in the document). 'Random' algorithm selects a random candidate as representative. 'Most frequent' algorithm selects the most frequent candidate from the candidate list.

F-scores for TopicRank vs. SlidoRank (MK. II)

Algorithm	$f1@1$	$f1@5$	$f1@10$	$f1@20$
TopicRank	0.95	6.67	9.56	11.67
SlidoRank (Mk. II)	**2.54**	**8.54**	**14.02**	**15.01**

Table 8.3: F-score comparison of SlidoRank (Mk. II) with TopicRank tested on the MK-DUC-01 dataset. Clustering threshold for SlidoRank is 0.6 **and the representative selection algorithm is "our".**

approach might work for some simple applications, it is possible to construct the embedding in much more efficient and compact way - feature vectors.

Feature vector (in the context of NLP and word embeddings) is, just like in one hot encoding, a vector representation of a specific word. While in one hot encoding the size of the vector equals the size of the vocabulary, feature vectors can be arbitrarily long (short). The core concept of feature vectors is to extract or "learn" the features of a word, its semantic meaning and its concept through training a neural network architecture which is exposed to large quanta of text. While word embeddings and neural architectures that build them are a very important topic withing the field of Artificial Intelligence and Machine Learning, I will not go into much detail about the workings of such architectures.

In this section I present *The Embeddings algorithm*, an extension of SlidoRank which "fine-tunes" the output of SlidoRank (MK. II) to group together semantically similar candidates. We do so by using a pre-trained sentence transformer (sBERT) with the aim of increasing the f-scores on the MK-DUC-01 dataset. I will provide the outline for this algorithm, implementation details and most importantly the evaluation results.

9.2 Outline of The Embeddings

The Embeddings is not a novel algorithm aiming at solving the KE task in a novel way. It has to be understood as an extension or a "build-up" on top of the SlidoRank algorithm. We can extend the original pipeline as follows:

1. Pre-process document
2. Generate candidates from cleansed document
3. Score candidates (determine their importance)
4. Candidate post-processing (eg. deduplication)

5. Ranking and final keyphrase selection
6. **Generate candidate embeddings**
7. **Compute centroids of clusters**
8. **Add semantically similar candidates to clusters**

9.3 Generating candidate embeddings

Since we used a pre-trained model, generating the embeddings was very simple. We overrode the *Candidate* dataclass with a new field *embedding* as folows: $self.embedding = MODEL.encode(' '.join(self.surface\_form))$. We are embedding joined surface forms (surface forms is an array of tokens) because this includes the most natural form of candidate with most of the semantic information. Similarly, we overrode the *Question* dataclass as: $self.embedding = MODEL.encode(self.text)$. The reason for embedding entire questions will be explained later in this section.

9.3.1 Language model selection

One of the most fundamental questions or points of debate is language model selection. From all the pre-trained models available, which one should we use? Models differ in the training datasets, number of parameters or architecture and leading to the difference in speed performance and their "depth of understanding natural language" (or their performance on various NLP tasks). Since it is still unknown to us which language model would be best for our use-case, I will also include a comparison of various models and their influence on the resulting f-score. For now, we have decided to use *paraphrase-MiniLM-L12-v2* which appears to be a good combination of the size of the model (hence its speed performance) and its quality (how well it performs on various NLP tasks).

The *encode*() function is computationally expensive since it has to feed the word through the (generally large) neural network architecture and obtain a feature vector. This means that we are expecting a significant increase in time complexity.

9.4 Cluster centroids

After redefining the *Question* and *Candidate* classes we override the *Cluster* class. Cluster holds all the candidate objects and now it will also hold their embeddings. This means that from the perpective of semantic similarity and feature vectors, we can perceive a cluster as a distribution of n-dimensional feature vectors in a vector space. This is, by itself, very useful since it allows us to study the mathematical attributes of these distributions.

9.4.1 Centroid

One important property of a cluster is its "centroid" vector. We define a centroid vector as a vector $\mathbf{v} \in R^n$ whose sum of cosine distances to all other vectors within the cluster is minimal. As is standard practice, we use cosine distances to measure the similarity of two high-dimensional vectors (for low dimensions other metrics can be used as well). Nice property of this vector is that it describes a set of features that are best descriptive of all the candidates. The centroid is computed by defining an array of embeddings from all the candidates, summing all the vectors component-wise and dividing by their amount. We took use of the numpy library and the pseudocode is shown in 9.1.

Algorithm 9.1 Compute cluster centroid

Require: Cluster object
  $sum\_vec \Leftarrow \mathbf{0}$
  for every cluster candidate embedding *emb* **do**
    $sum\_vec \Leftarrow sum\_vec + emb$
  end for
  $centroid \Leftarrow \frac{sum\_vec}{\#embeddings}$
  return *centroid*

9.4.2 Cohesion

Another important property of a cluster is the shape of the distribution itself, its cohesion. Cohesion is a floating point number given by the mean of the cosine distances of all candidates from a cluster centroid. If this cohesion value is large, this means that the average distance from the centroid vector is large. On the other hand, low cohesion coefficient means that the average distance of a candidate to centroid is small, hence the cluster distribution is more "packed together". The reason for computing the centroid and cohesion will be described in section 9.5

The computation of cohesion is also straightforward: we first compute the centroid and take the mean distance of all the candidates to the centroid 9.2. Note that the word "cohesion" here has a counter-intuitive meaning since high cohesion generally means that the cluster is strongly "tied together" - the word *density* would be more appropriate.

Algorithm 9.2 Compute cluster cohesion

Require: Cluster object
  centroid ⇐ *compute_centroid*()
  dist_arr ⇐ []
  for every cluster candidate embedding *emb* **do**
    dist_arr.append(*cosine*(*emb*, *centroid*))
  end for
  return the mean of *dist_arr*

9.5 Adding semantically similar candidates to clusters

The core idea of enhancing the clusters is to loop over all the candidates from all the clusters and add a candidate to a cluster if the cosine distance between the candidate embedding and the cluster centroid is "sufficiently small". This procedure should ensure that even candidates that do not share a common stem and hence are in a different cluster, but are semantically similar to candidates in other cluster, will be added. As an example, a candidate *laptop* will be added to a cluster *[computer, cisco computers, computing devices]* even though it does not share a common stem with any of the candidates.

Before we start comparing the cosine distances and adding the candidates, we first perform a similar operation of merging semantically similar clusters. During testing we noticed that there are some semantically similar clusters that should be merged into one larger cluster (eg. *[price, high price, pricey]* and *[cost, big cost, costly]* - these two clusters should be merged into one since they are both semantically very similar despite not sharing a common stem).

This led to the introduction of *cluster_clusters*() (by my supervisor Daniela) functions which, despite its unfortunate naming, does exactly what it says - it clusters or merges clusters together. We first create a matrix X containing all the centroids of all clusters, a result is an array of vectors. These vectors are clustered using agglomerative clustering with average linkage and a threshold of 0.4. Lastly, candidates are connected to these centroids and new cluster objects are formed. This algorithm allows us to fully merge those clusters whose cosine distance of the average of their feature vectors is small - resulting in *cost* and *price* clusters to become one cluster (or topic).

(Cluster clusters was merely an experiment and after some investigation we realized that no matter the clustering threshold (inside the function), the F-scores were significantly lower (by a half) then originally. The reason for this is still to be determined and further investigation is needed. For now, we have decided to leave this function out.)

Final stage of this process is to add specific candidates to other clusters. We iterate over all the candidates present in an event and compare their embedding with every cluster centroid. If the cosine distance is "small enough", we add the candidate to the cluster. We also increase the $frequency$ of the cluster and for analytic purposes we store the representative of a cluster the candidate is originally from. Let us now dive deeper into what does "small enough" distance mean.

Cosine distance is a number from the interval $[-1, 1]$ so a reasonable first attempt is to use a constant threshold E of, say, $E = 0.5$. However, not all clusters have the same shape and density. It should be more difficult for a candidate to be added to a very dense cluster then to a "loose" cluster, see figure 9.1.

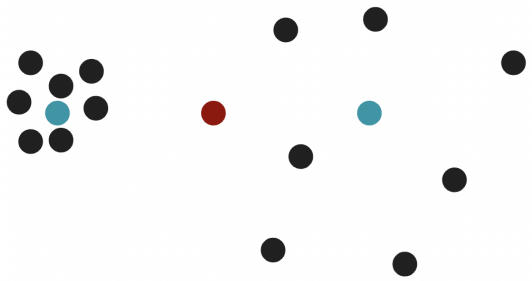

Figure 9.1: Two clusters with different density with candidate (red) having the same distance from both centroids but should be added to the left cluster.

Hence, we can refine the threshold E to take into consideration both the constant value and the cluster density ρ, yielding:

$$E(c) = \frac{C + \rho(c)}{2}$$

where C is the constant value and $\rho(c)$ is the density of a cluster c. It is difficult to say whether these two components should have the same weight. In some scenarios the cluster density can contribute only little while the constant might be more important. We introduce weight w which allows us to vary the importance of the two components yielding:

$$E(c) = (w * C) + (1 - w) * \rho(c)$$

where $w \in [0, 1]$. Now the equation is almost complete but there is another problem - for some set of weights, say, $w = 0.5$, if the cluster is "too loose and spread out" it will likely get all the candidates. Hence we want to penalize clusters that are "too

loose". We do so by taking the logarithm of the density, yielding the final equation:

$$E(c) = (w * C) + \big((1 - w) * log_2(1 + \rho(c))\big) \quad (9.1)$$

The behavior of this equation will be further studied and analysed in section 10.0.1

10 Results

This section presents evaluation benchmarks for the Embeddings extension presenting the $F1@k$ scores with different parameters of the algorithm. We use the MK-DUC-01 dataset and f-score with k values of $\{1, 5, 10, 20\}$. Parameters we will be comparing are the language models and the weight and constant in equation 9.1.

10.0.1 Hyperparameter search

The parameters *weight* and *constant* in the equation 9.1 determine which candidates and how many of them will be added to which cluster. To further investigate the influence of these parameters on the f-scores, I performed a large-scale hyperparameter search using a fast language model *paraphrase-MiniLM-L3-v2* with the focus on F1@20.

For each value of $C \in [0.0, 0.1, \ldots, 0.5]$ we measure the F1@20 for every value of $w \in [0.0, 0.1, \ldots, 1.0]$. There are two reasons for C being capped at 0.5.

1. Large values of C lead to almost "uncontrolled" adding of candidates to clusters which will likely negatively modify the original structure of the clusters generated by SlidoRank

2. Running the tests for large values of C simply takes too much time. It took a couple of hours to generate a single row of values for $C = 0.5$. Running the tests for the entire range of C would likely take in the order of a few days. (This will likely remain as future work as well as testing different values for k in the F1@k metric)

Results of this hyperparameter search are graphically expressed in figures 10.1 and 10.2.

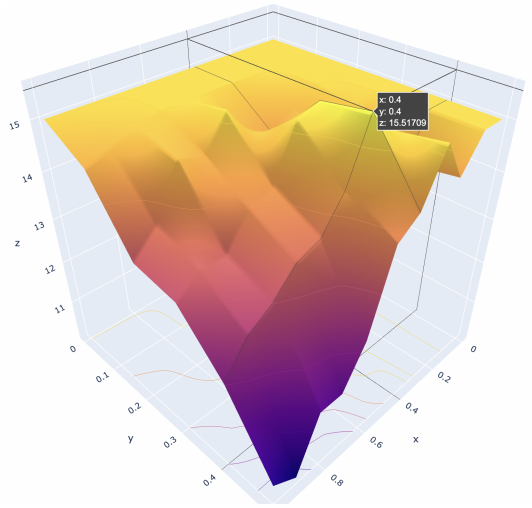

Figure 10.1: Cost landscape for combinations of w and C for all topics from the MK-DUC-01 dataset. The z-values are the F1@20 scores.

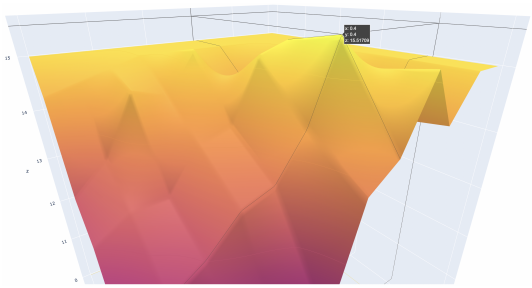

Figure 10.2: The same cost landscape zoomed onto the peak value of $w = 0.4$ and $C = 0.4$ with an $F1@20 = 15.517$.

We can clearly observe from figure 10.1 that larger values of C (say, 0.5) lead to a worse f-score **on average** than smaller values of C. However, the landscape peaks at values $w = 0.4, C = 0.4$ with $F1@20 = 15.517$. Although there is a fundamental limitation in C being capped at 0.5 and the "definition" of the landscape (total number of points), the Embeddings algorithm with values $w = 0.4, C = 0.4$ slightly outperforms SlidoRank (which is essentially the Embeddings algorithm with values $w = 1, C = 0$).

The cost landscape indicates that the idea of adding semantically similar candidates to clusters works if one uses the correct values in the equation $E(c) = (w * C) + (1 - w) * \rho(c)$. However, one must take into consideration the language model for which the cost landscape will likely differ in shape and the remaining metrics F1@1, F1@5, and F1@10 (these remain as future work).

10.0.2 Language model influence

There is a variety of language models that can be used for generating the embeddings. For testing, we have decided to use three models from the sBERT website which includes a table comparing the performance of various sBERT models. The difference between the *all-* models and the *paraphrase-* models is that *all-* models were trained on all available training datasets while *paraphrase-* were trained only on a subset of the available datasets. To investigate whether the size of the model has an influence on the f-scores we will use three variations of the *all-* models including the small *paraphrase-* which was used at the hyperparameter search:

1. **paraphrase-MiniLM-L3-v2** as the fastest and smallest model used for the cost landscape computation (61MB)

2. **all-MiniLM-L6-v2** as the fastest and smallest model among the *all-* models (80MB)

3. **all-distilroberta-v1** as a mid-sized model (290MB)

4. **all-mpnet-base-v2** as the best performing and largest model (420MB)

Table 10.1 shows the f-scores of the Embeddings algorithm using different language models with parameters of the equation 9.1 clamped at $w = 0.4$ and $C = 0.4$.

According to table 10.1 a clear pattern of the influence of the size of the model to the f-scores is not evident. The differences between model sizes is large but the difference between the f-scores is rather small. This rejects our hypothesis that larger models with "deeper understanding" of language will generate "more accurate" feature vectors which lead to more accurate process of cluster-candidate regrouping. What comes as a surprise is the fact that the smallest model *paraphrase-MiniLM-L3-v2* has better F1@10 and F1@20 scores then any other model tested. It appears that rather than the size of the model, it might be the model's architecture or the training set that might have a more significant influence.

The good performance of the *paraphrase-MiniLM-L3-v2* made me create another benchmark table 10.2 testing three variations of this model *L3*, *L6*, *L12*.

However, the dependence of the F-scores on the language model size does is not evident at all. The largest model has the best F1@1 and F1@20 but the smallest model leads in F1@5 and F1@10.

10.0.3 SlidoRank (Mk. II) vs. Embeddings vs. TopicRank

To conclude this section we present a comparison of the three algorithms, original TopicRank, the significantly faster SlidoRank (Mk. II) and the Embeddings algorithm. Our hypothesis was that by introducing language models and feature vectors we group semantically similar keyphrases resulting in an increase of the f-scores. As shown in figure 10.1, it is possible to obtain better F1@20 score with the appropriate language model and parameters of the equation $E(c) = (w * C) + (1 - w) * \rho(c)$. A constant value of $C = 0.4$ and weight $w = 0.4$ it is possible to increase F1@20, however, many other combinations of w and C lead to a decrease of F1@20. We can also conclude that the distribution of candidate embeddings in feature space (cluster cohesions) does matter for the final keyphrase extraction.

Table 10.3 shows a comparison of the three algorithms with SlidoRank (Mk. II) and Embeddings with the best configuration I found:

- **Tf-Idf**: original configuration as it is implemented in the *pke* library
- **TextRank**: original configuration as it is implemented in the *pke* library
- **TopicRank**: original configuration as it is implemented in the *pke* library
- **SlidoRank**: threshold=0.54, linkage='average', len(lexical_form) > 1, representative='our'
- **Embeddings**: weight=0.4, constant=0.4, model='paraphrase-MiniLM-L12-v2'

Note: TopicRank, SlidoRank and Embeddings use the same set of POS tags: $\{'NOUN', 'PROPN', 'ADJ'\}$

11 Conclusion

11.1 Discussion

It is important to note that for the Slido use-case, the candidates in a topic and their associated questions are equally important, if not more, than representative selection. When the user clicks on a topic, he/she wants to see all associated questions with that topic. As an example, whether a user sees *Tunnels* or *Underground tunnels* as a topic, it is not as relevant as whether all the associated candidates and questions are present in it. However, this is simply too difficult to test and to the best of our knowledge, there exists no large evaluation dataset for this specific case. All

F-scores (Language Models)

Model	$f1@1$	$f1@5$	$f1@10$	$f1@20$
paraphrase-MiniLM-L3-v2 (61MB)	1.9	8.27	**14.02**	**15.52**
all-MiniLM-L6-v2 (80MB)	1.9	8.27	12.68	15.02
all-distilroberta-v1 (290MB)	**2.22**	**8.54**	12.68	14.68
all-mpnet-base-v2 (420MB)	1.59	7.47	10.9	15.01

Table 10.1: F-score comparison of the Embeddings algorithm with three different language models.

F-scores for paraphrase-MiniLM models

Model	$f1@1$	$f1@5$	$f1@10$	$f1@20$
paraphrase-MiniLM-L3-v2	1.9	**8.27**	14.02	15.52
paraphrase-MiniLM-L6-v2	2.22	7.74	**12.9**	15.35
paraphrase-MiniLM-L12-v2	**2.54**	8.0	12.45	**16.35**

Table 10.2: F-score comparison of the Embeddings algorithm using three variations of the paraphrase-MiniLM

F-scores comparison of the default and proposed algorithms

Algorithm	$f1@1$	$f1@5$	$f1@10$	$f1@20$
Tf-Idf	0.32	2.67	5.56	6.5
TextRank	0.63	4.0	5.33	9.17
TopicRank	0.95	6.67	9.56	11.67
SlidoRank (Mk. II)	2.54	**8.54**	**14.02**	15.01
Embeddings	**2.54**	8.0	12.45	**16.35**

Table 10.3: F-score comparison of all algorithms tested on the MK-DUC-01 dataset (truncated). Values are in % and "our" representative selection algorithm was used in SlidoRank (Mk II) and Embeddings.

things considered, the F-score on the MK-DUC-01 dataset is the best evaluation we currently have and we will continue using it in the future.

SlidoRank (Mk. II) turned out to be a great success in outperforming TopicRank in both speed and F-scores based on the MK-DUC-01 dataset. Furthermore, when run with specific parameter configuration, SlidoRank (Mk. II) outperforms almost **all** of its competitors (PositionRank being equally as good). The speed improvements done for the first generation of SlidoRank allowed us to scale the questions-by-topics feature to very big events (almost arbitrarily big) with sustained quality of extracted keyphrases. (It is also important to note that the algorithm itself is just one component of a larger pipeline; SlidoRank is zipped and sent to AWS as a lambda function and the internal processes are stored and further sped up using Elastic Search.)

The primary hypothesis of increased F-scores by introducing semantic similarity turned out to be plausible only in a specific configuration of the equation 9.1 and a good language model. However, the Embeddings extension has a greater influence on which candidates "fall under" which topic rather than the topics itself. This can be very beneficial to Slido users (since the questions under each topic seem more related) despite the increase in F-scores on the MK-DUC-01 dataset being minimal. The Embeddings approach might soon find its way into production of Slido Q&A section or to Open Text Polls.

11.2 Dataset bias

It is important to note that all the parameters have been found and tested only on the MK-DUC-01 dataset. The average number of keyphrases per topic as well as average keyphrase size was known beforehand and the fine-tuning process was done with these statistics in mind. This is a clear case of overfitting and there is absolutely no guarantee that the high F-scores of SlidoRank will remain high across multiple different datasets. However, the MK-DUC-01 is the first high-quality benchmark dataset in a true Multi-Document setting published in October, 2021 and to the best of my knowledge still remains the only one.

11.3 Streamlit demo

As a neat addition to the Thesis, I made a simple web application demo using Streamlit visualizing three different algorithms (side-by-side) and the topics generated by them (see figure 11.1).

To see the results, the user must first either input text into the text-box or upload a *.csv* file.

The demo currently includes SlidoRank + Embeddings with two different language models and an EmbedRank algorithm. Although EmbedRank with Maximal Marginal Relevance (MMR) is also a promising algorithm worth writing about, due to the scope of this Thesis I have decided to skip its description and simply provide an implementation in the Streamlit app.

11.4 Summary

Despite hard efforts, we did not manage to obtain the exact same results for Tf-Idf, TextRank and TopicRank as in table 5.1. However, we managed to obtain scores as close as $1 - 2$ percentage points (or $0.01 - 0.02$ in f-score) difference from the ones in table 5.1. However, for some F-scores we managed to obtain the exact values so we conclude that the results in table 5.1 are partially (or almost) reproducible.

Our first proposed algorithm SlidoRank outperforms the state-of-the-art algorithm TopicRank in both speed performance (on the MK-DUC-01 dataset with 1140 sentences from 30sec to only 5) and the quality of extracted keyphrases (higher F1@k scores at $1, 5, 10, 20$). Moreover, SlidoRank also outperforms other widely used algorithms such as YAKE, TextRank, and others (see table 11.2).

The second proposed algorithm, the Embedding approach, has the potential to increase the F-scores even further for a specific parameter configuration described in section 9 but the use of a language model significantly slows down the process.

The speed and quality improvements that SlidoRank offers allows Multi-Document Keyphrase Extraction to be performed even on very large datasets with high topic variance. This opens new doors for commercial or other use cases (questions-by-topics in Slido).

12 Future work

Slido is aiming at becoming the default go-to application for public events and private meetings with millions of monthly active users all over the world. There is always room for improvement for these algorithms and in the future we are planning to collect candidates that are in the wrong cluster and use these data to change the parameters of the equation 9.1.

There is a long-term vision to replace the 'all-MiniLM-L6-v2' language model by our own in-house language model trained on Slido data.

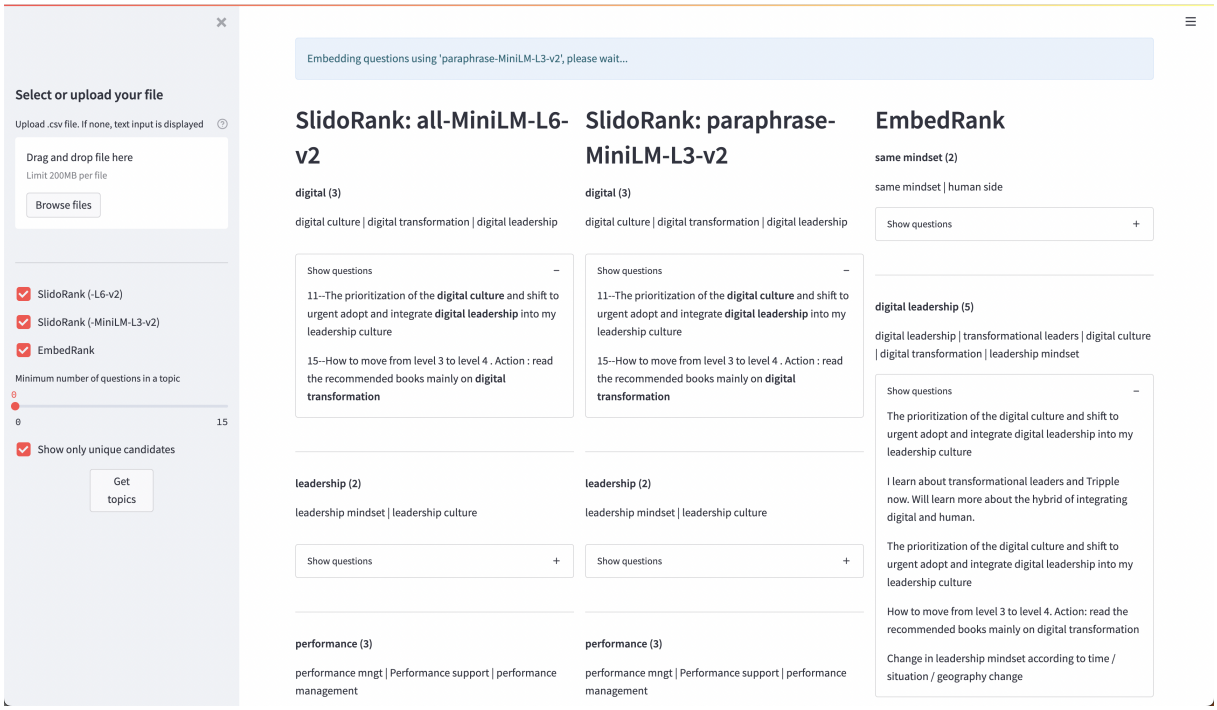

Figure 11.1: Streamlit demo with SlidoRank and EmbedRank. Not deployed due to legal issues but available at request.

	Concat							
	F1@k				**unigram-F1@k**			
Algorithm	**1**	**5**	**10**	**20**	**1**	**5**	**10**	**20**
Tf-Idf	0.32	1.87	4.44	5.67	4.56	16.77	26.58	35.11
KPMiner (El-Beltagy and Rafea, 2009)	1.27	4.80	7.56	9.68	5.52	21.27	30.55	34.88
YAKE (Campos et al., 2020)	2.54	7.20	10.67	13.17	6.24	24.80	33.30	36.04
TextRank (Mihalcea and Tarau, 2004)	0.63	4.00	4.67	8.01	10.11	28.25	32.03	30.72
SingleRank (Wan and Xiao, 2008)	0.95	5.08	6.90	12.01	12.51	29.52	32.81	31.98
TopicRank (Bougouin et al., 2013)	1.59	6.40	9.33	11.33	5.47	18.69	28.75	36.72
TopicalPageRank (Sterckx et al., 2015)	1.27	5.61	7.79	13.35	12.16	29.81	33.67	32.44
PositionRank (Florescu and Caragea, 2017)	1.90	8.28	11.80	16.35	9.31	27.09	32.20	33.22
MultipartiteRank (Boudin, 2018)	1.27	6.13	10.23	12.17	6.04	20.08	30.12	38.70
SlidoRank	2.54	8.54	14.02	15.01				
Embeddings	2.54	8.0	12.45	16.35				

Figure 11.2: SlidoRank and Embeddings in comparison with other widely used algorithms

Although we do have a fine-tuned Slido model, it does not outperform the standard models yet.

Even more long-term vision is to have a large benchmark dataset with Slido events and a list of golden topics together with their representatives, candidates, and associated questions. Such dataset would be a significant help in developing and evaluating our algorithms. Moreover, if the dataset becomes large enough, it might open doors to new approaches like supervised learning.

Lastly, I would like to further analyse the behavior of SlidoRank by testing the algorithm in 'merge mode' instead of only 'concat mode', evaluate precision & recall values, parallelize the evaluator and test all combinations of all parameters.

13 Acknowledgements

Finally, I would like to thank Daniela Jašš for all her support and help during this Thesis and my time at Slido. I also want to thank my colleagues baran, kitkat, sweco, virpo and the rest of the Data Team for their guidance, contributions and code reviews and mrshu for being the best team leader. Lastly I want to thank Dr. George Azzopardi and Dr. Fadi Mohsen for academical supervision and their feedback.

References

Random sentence generator. https://randomwordgenerator.com/sentence.php. Accessed: 2022-05-01.

Roy hung, jaccard's index in practice. https://royhung.com/jaccard-index. Accessed: 2022-05-01.

Adrien Bougouin, Florian Boudin, B. D. (2013). Topicrank: Graph-based topic ranking for keyphrase extraction. *International Joint Conference on Natural Language Processing (IJCNLP)*, pages 543–551.

Berend, G. and Farkas, R. (2013). Singledocument keyphrase extraction for multidocument keyphrase extraction. *Computación y Sistemas*, page 17(2):179–186.

Bulgarov, F. and Caragea, C. (2015). A comparison of supervised keyphrase extraction models. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 13–14, New York, NY, USA. Association for Computing Machinery.

Campos, R., Mangaravite, V., Pasquali, A., Jorge, A., Nunes, C., and Jatowt, A. (2020). Yake! keyword extraction from single documents using multiple local features. *Information Sciences*, 509:257–289.

Christian, H., Agus, M. P., and Suhartono, D. (2016). Single document automatic text summarization using term frequency-inverse document frequency (tf-idf). *ComTech: Computer, Mathematics and Engineering Applications*, 7(4):285–294.

Farnoush Bayatmakou, A. A. and Mohebi, A. (2017). Automatic query-based keyword and keyphrase extraction. *In 2017 Artificial Intelligence and Signal Processing Conference (AISP), IEEE*, page 325–330.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer.

Hulth, A. (2003). Improved automatic keyword extraction given more linguistic knowledge. *In Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, page 216–223.

Khaled M Hammouda, D. N. M. and Kamel, M. S. (2005). Corephrase: Keyphrase extraction for document clustering. *International workshop on machine learning and data mining in pattern recognition, Springer*, page 265–274.

Muthmainnah, N. (2019). An effort to improve students' activeness at structure class using slido app. *JEES (Journal of English Educators Society)*, 4(1):1–7.

Pal, S. K. and Mitra, S. (1992). Multilayer perceptron, fuzzy sets, classifiaction.

Rish, I. et al. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46.

Rodríguez, P., Bautista, M. A., Gonzalez, J., and Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31.

Rose, S., Engel, D., Cramer, N., and Cowley, W. (2010). Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1(1-20):10–1002.

Shapira, O., Pasunuru, R., Dagan, I., and Amsterdamer, Y. (2021). Multi-Document Keyphrase Extraction: A Literature Review and the First Dataset. *arXiv preprint arXiv:2110.01073*.

Shobha S. Raskar, S. H. P. (2014). Keyphrase extraction using supervise learning. *International Journal of Advanced Research in Computer and Communication Engineering*.

Sun, Chengyu, L. H. S. L. T. L. H. L. and Chi, L. (2020). A review of unsupervised keyphrase extraction methods using within-collection resources. *Symmetry 12(11)*.

Ulrich Schäfer, J. R. and Oepen, S. (2012). Towards an acl anthology corpus with logical document structure. an overview of the acl 2012 contributed task. in proceedings of the acl2012 special workshop on rediscovering 50 years of discoveries. *Association for Computational Linguistics*, page 88–97.

Wan, X. and Xiao, J. (2008). Collabrank: Towards a collaborative approach to single-document keyphrase extraction. *In Proceedings of the 22nd International Conference on Computational Linguistics*, page 969–976.

Xing, W. and Ghorbani, A. (2004). Weighted pagerank algorithm. In *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*, pages 305–314. IEEE.

A Appendix

Figure A.1 shows quantitative results - keyphrases extracted from topic d31 from the MK-DUC-01 dataset (truncated at 20) from all five tested algorithms including the golden set.

Extracted Keyphrases from topic d31

	Golden set	Tf-Idf	TextRank	TopicRank	SlidoRank II	Embeddings
1	drug testing	canadian olympic track physician dr. robert luba	johnson	johnson	ben johnson	ben johnson
2	illegal steroid use	fellow canadian olympic sprinter angella issajenko	steroids	steroids	gold medal	gold medal
3	Olympics gold medal	olympic gold medals johnson	ben	american carl lewis	carl lewis	carl lewis
4	Seoul Olympics	canadian olympic assn .	drug	ben johnson	world record	world record
5	banned steroid	canadian ben johnson	olympic	illegal drugs	anabolic steroid	anabolic steroid
6	Ben Johnson	canadian olympic officials	lewis	francis	drug use	drug use
7	world record	international olympic committee president juan antonio samaranch	francis	olympic gold medal	drug test	steroid use
8	anabolic steroid stanozolol	world sprint record last summer	athletes	second world record	charlie francis	drug test
9	world championships	seoul olympics last year	canadian	canada	world championships	gold medalist
10	Charlie Francis	canadian national sprint coach	said	sports columnist	New York	charlie francis
11	100-meter dash	canadian athletes	medal	positive	speed trap	steroid distribution
12	stanozolol use	national hero ben johnson	ben johnson	athlete	toronto star	world championships
13	Carl Lewis	sprinter ben johnson	sprinter	olympics	meter dash	New York
14	urine sample	ben johnson sr .	canada	canadian people	Olympic Games	urine sample
15	steroid furazabol	meters olympics gold medal	gold medal	canadian	seoul olympics	speed trap
16	Jamie Astaphan	international amateur athletics federation	toronto	world	olympic officials	toronto star
17	steroid combination	canadian coach charlie francis	gold	track	urine sample	meter dash
18	Toronto	world class athletes	sports	sprinter horace dove-edwin	los angeles	record time
19	personal physician	olympic gold medal	seoul	suburban toronto	South Korea	other athletes
20	disgraced Olympic sprinter	olympic sprinter	astaphan	field	positive test	Olympic Games

Figure A.1: Set of extracted keyphrases (truncated version on 20) for all algorithms compared to the golden set. Tested on topic $d31$ from the MK-DUC-01 dataset.