Publication Records Analysis and Processing

1 Introduction

In this report, we share our findings and discussions regarding the analysis of three diverse datasets, using various word normalization and segmentation techniques. Subsequently, we indexed documents from DBLP and built a search-engine to retrieve them. Then, we studied the research trends within WSDM conference over the past years. Finally, we devised and built a simple user interface to retrieve documents from our search engine based user's input.

2 Domain Specific Dataset Analysis

For this segment, we gathered text data from the following domains, where each document is defined as content from Google News, Reddit posts, and PDF journals:

- 10 x Google News: 2023 Israel – Hamas war
- 10 x PDF: Journals related to the field of Deep Learning
- 10 x Reddit posts: Related to League of Legends

Both the Google News and Reddit posts were manually transferred into an Excel sheet before being imported into Python. This manual transfer helps reduce the amount of dirty data generated during the import into Python.

The journals were downloaded in PDF format and loaded into Python using the PyPDF library. However, there were instances when the PyPDF library couldn't accurately identify the indentation of sentences, leading to a loss of information or incorrect information in the loaded data. This issue was more common in the author's section and areas where pictures were used. Despite this, information for the majority of each journal was generally preserved.

2.1 Tokenization

We utilized the `word_tokenize()` function from the NLTK library for tokenization. This NLTK tokenizer function relies on pre-trained unsupervised machine learning models within the Punkt module. It tokenizes strings using a combination of blank spaces and punctuation.

Domain 1: Google News
In the context of Google News, articles often contain quoted figures, dates, and names of events, people, and countries. The `word_tokenize()` function demonstrated accurate tokenization in the following examples in Tab 1:

- Numbers separated by commas and words separated by hyphens were precisely tokenized and identified as distinct words.
- Common punctuations (e.g., commas, periods, question marks) were correctly separated into individual tokens.

However, the NLTK tokenizer exhibited limitations in the following scenarios 2:

- Dates were inaccurately tokenized.
- Names with title cases (words with the first alphabet capitalized, e.g., "This Is Title Cases") were tokenized incorrectly, despite the initial alphabet of each name being in uppercase.
- Words ending with " 's " were tokenized into 2 or 3 different tokens. This discrepancy may stem from the ambiguity introduced by the use of apostrophes in different languages (such as Chinese vs. English), where visual similarities can lead to misinterpretation.

Domain 2: Reddit posts
The NLTK tokenizer did not perform well in tokenizing names Tab 3, akin to the issue described in 1c.

Domain 3: PDF Journals relating to Deep Learning
We retrieved 10 scientific journals in PDF format and subsequently imported them into Python using the PyPDF2 library. However, the PDF reader encountered challenges during conversion, leading to the following errors Tab 4:

- The PDF library failed to maintain the indentations present in the original PDFs. Some spacing was omitted, while additional spacing was introduced.
- Numberings were misinterpreted and represented as non-ASCII characters (e.g., †Work).

In addition to discrepancies arising from reading errors, the tokenization of the PDF document revealed the following issues:

- Figure labels were treated as individual tokens.
- Domain-specific names, including those with mixed case (similar to 1d), and those without mixed case, were treated as separate tokens.

S/No	Before Tokenization	After Tokenization	Comments
1a	3,700	|3,700|	Numbers with ',' were usually correctly tokenized
1b	co-founder	|co-founder|	Words with hyphens were correctly tokenized
1c	shot, mutilate	|shot| |,| |mutilated|	Words separated by punctuations such as comma were correctly separated

Table 1. Accurate examples of word tokenization using NLTK in Google News dataset.

S/No	Before tokenization	After tokenization	Correct token	Comments
1d	Web Summit	|Web| |Summit|	|Web Summit|	Names of meetings, people and location should be treated as a single token
1e	Paddy Cosgrave	|Paddy| |Cosgrave|	|Paddy Cosgrave|	
1f	Oct 7	|Oct| |7|	|Oct 7|	Dates should be treated as a single token
1g	Israel's	|Israel| |'| |s| OR |Israel| |'s|	|Israel's|	Tokens with " 's " should be tokenized as a single token

Table 2. Inaccurate examples of word tokenization using NLTK in Google News dataset.

Domain	S/No	Correctly tokenized?	Before tokenization	After tokenization	Correct token	Comments
Reddit	2a	N	G2 Esports	|G2| |Esports|	|G2 Esports|	Team names should be treated as a single token (similar to 1d)

Table 3. Inaccurate examples of word tokenization using NLTK in Reddit posts dataset.

- Formulas were also treated as separate tokens.

As such, we propose additional improvements to the tokenizer as shown in Tab 5.

2.2 Stemming

To enhance the quality and consistency of the data, tokens comprised solely of punctuations were removed before applying the NLTK PorterStemmer. This aims to eliminate any potential bias introduced by punctuations, ensuring a more accurate representation of word distribution both before and after stemming.

In addition, few typos/abbreviations/short word forms were present in our data as both Google News and PDFs were written in a formal tone, tailored for public consumption. Most of the tokens came from the PDF files due to the high number of text present in these PDF files. Despite the PyPDF2 library encountering reading errors, a significant proportion of words were accurately translated and ingested into Python, allowing NLTK to effectively stem the majority of tokens.

The application of stemming resulted in a notable reduction of approximately 24% in the number of unique tokens, decreasing from 16.8k to 12.9k Fig 1. This reduction signifies a consolidation of word variations, notably the most substantial increase in the number of unique tokens was observed among tokens with lengths ranging from 6 to 15 characters as shown in Fig 2.

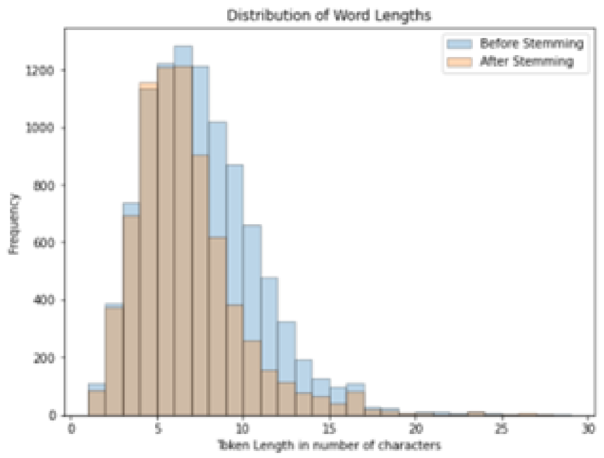


Figure 1. Distribution of word length before and after stemming

S/No	Before tokenization	After tokenization	Correct token	Comments
3a	|Figure 1|	|Figure| |1|	|Figure 1|	Figures X should be treated as a single token
3b	Key and Value states (KV)	|Key| |and| |Value| |states| |(| |KV| |)|	|Key and Value States (KV)|	Domain specific names (similar to 1d) should be treated as a single token
3c	NVIDIA P100	|NVIDIA| |P100|	|NVIDIA P100|	
3d	Convolutional layers	|Convolutional| |layers|	|Convolutional layers|	
3e	z= (z1, ..., z n)	|z=| |(| |z1| |...| |z| |n| |)|	| z= (z1, ..., z n)|	Formulas should be treated as a single token

Table 4. Inaccurate examples of word tokenization using NLTK in PDF Journals dataset.

S/N	Problem	Proposals for improvement	Cases not accounted for
1	Names with title cases are not considered as a single token	Group consecutive title-cased word which do not occur at the start of a sentence as the single token	Names with lower case/punctuation are excluded. E.g. Barney and Friends/Barney & Friends
2	Dates are considered as a single token	Group numbers up to 4 digits occurring before or after a month (in abbreviated or original form) together, separated by a single punctuation as a single token	Numeric dates not accounted for. E.g. 12-31-2020

Table 5. Improvement proposals for occurred problems and some cases are not accounted for.

2.3 Sentence Segmentation

Google News

Sentence segmentation works exceptionally well with Google News, primarily because articles are copied and pasted from online sources, significantly reducing the occurrence of dirty data.

PDF Journals

When applying sentence segmentation to PDFs, although segmentation is generally accurate, challenges arise from the inclusion of names, trademarks, non-ASCII characters, etc., which may contain full stops. While NLTK's `sent_tokenize()` often discern full stops within names, the presence of full stops in various places apart from the end of a sentence can lead to confusion and adversely impact sentence segmentation. Consequently, this confusion results in prematurely segmented sentences, causing some tokenized sentences from the PDF files to be underestimated and appear shorter than their actual length.

Reddit Posts

Reddit post data was transferred to Excel before being imported into Python, which effectively minimized the presence of dirty data. However, given the predominantly informal tone of Reddit posts, writers often omit and replace full stops with new paragraphs. This absence of full stops leads to NLTK overestimating the length of certain sentences and inadvertently combining multiple paragraphs and sentences.

The NLTK sentence tokenizer faces challenges when processing PDF sentences due to the presence of full stops in names, sentence endings, and formulas. This leads to NLTK segmenting sentences in PDFs being significantly shorter than their actual length. Consequently, PDF files exhibit the shortest average sentence length compared to Google News and Reddit posts.

Reddit posts are generally characterized by an informal tone and shorter sentences. NLTK tends to overestimate these sentence lengths, grouping multiple sentences together due to the lack of full stops. This results in the average sentence length for Reddit posts appearing longer than that of PDF files, contrary to the expected shorter length.

In contrast, Google News is designed for a public audience with a formal tone and relatively clean data. NLTK performs relatively well in segmenting Google News sentences, providing a fairly accurate representation of their actual sentence length. This analysis aligns with the fact that English sentences typically contain 20-25 words for optimal readability.

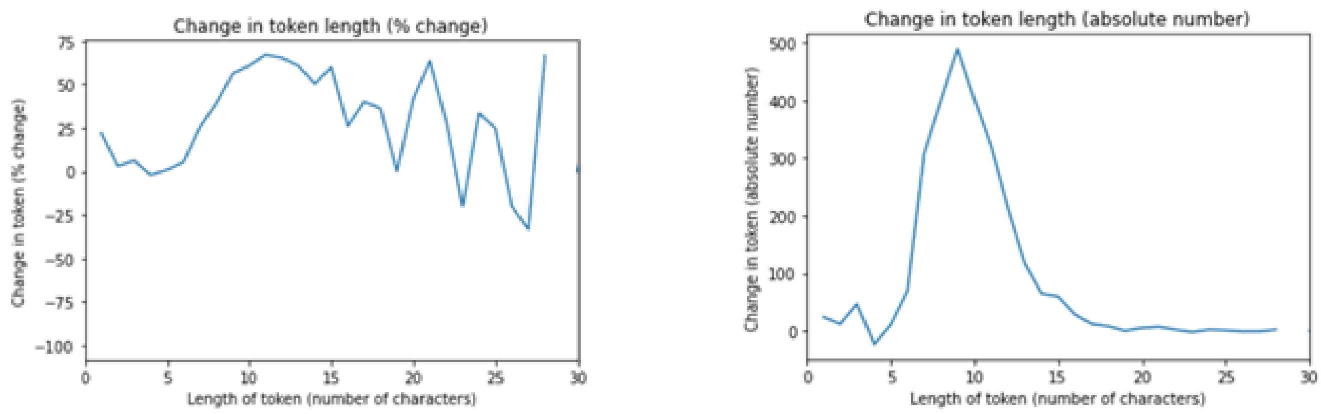


Figure 2. Changes in token length in percentage (%) and absolute number of PDF documents.

Domain	Premature sentence segmentation	Comments
PDF files	Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. – (end of token 8) – 2018. – (end of token 9) –	NLTK can identify a full stop as part of a name on certain occasions, but its accuracy in doing so is inconsistent, leading to premature sentence segmentation.
	– (end of token 18) – Author's addresses: S. Zhang and L. Yao, University of New South Wales; emails: shuai.zhang@unsw.edu.au; lina.yao@unsw.edu.au; A. – (end of token 19) – Sun	

Table 6. Premature sentence segmentation in PDF documents

3 Search Engine for DBLP

3.1 Indexing

In this section, we attempt to create a search engine for documents within the DBLP corpus. DBLP is an online database for bibliographic information on major computer science publications. These publications include journal articles, conference papers, and books. As of the writing of this report, there are roughly 6.9 million bibliographic information available on the website. This information includes the title of the publication, authors' names, and the year of publication.

The search engine which is built allows users to search for publications (i.e., the documents) using the fields "title", "author", "publication venue", and "year". In order to do so, we first indexed some of the bibliographic information that was provided in a .xml file. These bibliographic information are represented as tags of the elements in the .xml file, and the ones we indexed are shown in Tab 8. These tags are chosen because they contain relevant information for the corresponding search fields. For example, the <book> element has both <booktitle> and <title> tags. However, after some data exploration, we realized that the <title> tag under these elements is often not sufficient, as some of the relevant information for the "title" search field falls under the <booktitle> tag. Similarly, the information required

Domain	Premature sentence segmentation	Comments
Reddit	In the name of the Flowers, the Kobe, and the Holy Phreak Our Father, who art in heaven this day. – (end of token 0) – We pray a Coup d'état of our sweet NA.	NLTK identifies the end of a sentence solely when a full stop is present. Reddit writers commonly replace full stops with escape sequences. This is usually not picked up by NLTK
	This year will be different for our boys of NA Reddit, please say it with me. – (end of token 4) – We pray.	

Table 7. Premature sentence segmentation in Reddit documents

for the "publication venue" search field is within various tags, depending on the element. If the element is a <book>, then the required information falls under the <publisher> tag, whereas if the element is a <article >, then the information falls under the <journal> tag.

Indexing is the method by which documents are being processed and organized, such that subsequent search and retrieval of these documents would be efficient. Since we are indexing different fields, each of the fields is considered a corpus. For example, the "title" field is a corpus, where the documents are the <title> and <booktitle> tags from all the publications. By indexing this field, we collate all the unique words in the <title> and <booktitle> tags and have them point to the documents that contain these words.

In this project, the indexing is done with the use of a third-party library, *Whoosh*. Apart from indexing, *Whoosh* provides other functionalities such as parsing user queries, searching through an index, and basic word normalization. As there are roughly 6.9 million documents, each with differing tags, indexing without any processing would likely create an index that takes up lots of storage. Therefore, to reduce the computational cost, we performed some simple word normalization techniques on the tags. Table 2 below shows the normalization done to the terms in the corresponding tags.

The `SimpleAnalyzer()` and `StandardAnalyzer()` found in Tab 9 are functions available in *Whoosh* library. These analyzer functions take in a Unicode string as input and return a generator of tokens, with various normalizations performed to the tokens. We used two such analyzers, where the `SimpleAnalyzer()` and `StandardAnalyzer()` both perform tokenization and lowercase folding, while the latter has an added optionality of performing stop word removal. Here, we performed lowercase folding for the <author> and <editor> fields, which will change all the alphabets to lowercase ones. Since the names of authors are generally named entities, with or without case folding, we do not expect any major hiccups from this operation. This operation was also applied to the other tags, apart from <year> since we do not expect much information to be lost in general.

For the remaining tags, apart from <year>, we applied stop word removal on top of case folding. This operation removes all the commonly used words such as "the" and "a", which generally are very frequently used. In other words, the inverse document frequencies (idf) of these stop words are usually very high, indicating that their usefulness for document retrieval is low. For example, the word "the" would likely appear very frequently in many articles, so queries with "the" will not narrow down the search space effectively. As a result, it may be disregarded entirely, from the indexing and query parsing. Along the same vein, these stop words would typically have high collection frequency (but not always), resulting in higher memory costs if we included them in the index. So, by removing them, we effectively reduce the computational cost with minimal performance trade-offs. In our project, we used the built-in list of stop words in Whoosh. Finally, we did not apply stop word removal to the <author> and <editor> tags as any part of the names are informative and useful in helping us search for the authors.

Tags	Corresponding search field
<title>, <booktitle>	Title
<author>, <editor>	Author
<publisher>, <seriesyear>, <school>, <journal>	Publication venue
<year>	Year

Table 8. Tags that are indexed for the corresponding search field.

Tags	Analyzer used (Whoosh functions)	Normalization performed
<author>, <editor>	SimpleAnalyzer()	Lowercase folding
<title>, <booktitle> <publisher>, <series year>, <school>, <journal>	StandardAnalyzer()	Lowercase folding Stop Word Removal
<year>	None	None

Table 9. Normalization performed on the corresponding tags, using the corresponding analyzer.

We did not perform any normalization to the terms in the <year> tag as they are generally just years (e.g., 2020). We do not expect anything to be normalized from here.

To prepare for the indexing, we also defined a schema, using the in-built `Schema()` function. These are essentially the search fields we are indexing, and we will add the corresponding tags into the fields. We ran the indexing for the entire .xml file, which contains 6,786,619 elements (i.e., documents). The time taken to run every 10% of the documents is shown in Tab 10.

It is clear from Tab 10 that the indexing time does not linearly increase along with the percentage of indexed documents, and this is due to the index segment merging mechanism. When a new index is passed into the directory, the writer `AsyncWriter()` will merge the previous segment with the new segment to optimize the memory but this results in the time to search for the position of the terms before merging and the time performing the merge. When two indexes are large enough to be merged together, the writer will merge them, and this results in some abnormally large amount of running time. Moreover, part of this is also due to the dataset because we only read some specific tags so when reading the data file, some part of the file does not contain the tags we need and the writer takes time to skip those parts.

3.2 Searching

When using a search engine, users will typically convey their information needs using a text query. With this text query, the search engine will search through the index (or indexes), merge, and rank the results, before showing these results to the users. Before the query could be used to search the index, the same normalization steps as the ones applied to the index have to be performed on the query. This will change the search query to the same "form" as the index before we compare them. The basis of comparison, hence the score, can be different, involving a vector space model or BM-25.

In our project, the search is handled by the built-in `Searcher()` function in the *Whoosh* library. The user's query is processed by the `MultifieldParser()` function, which parses the query into terms, following the same schema used during the indexing process. This ensures that the parsed terms in the query match the terms in the index. The `MultifieldParser()` function checks the terms in the parsed query against multiple fields provided by the user. The function also automatically ranks the matches, using a default BM-25 score.

To analyze our search engine, we randomly chose a few queries to analyze the search engine: '2023', 'Robert Computer Vision'. We return the top three results, showing their scores, docID, and an excerpt from the document where the match is highlighted. The results are shown in Fig 3. The results returned may have multiple matches within the same field (see Fig 3a, third result) or from different fields (see Fig 3b, all results). This is expected, since `MultifieldParser()` searches for each parsed query token in all the fields, and return documents with at least one field containing each of the parsed query token. We can also see that documents (of the same length) with the same number of query terms have the same score (second and third documents

n	1	2	3	4	5	6	7	8	9	10
Time taken (s)	114.06	159.83	110.41	229.11	101.07	563.46	128.18	876.17	97.87	294.85

Table 10. The amount of time taken to index the n^{th} batch of the 10% of documents.

in Fig 3b). This is because the BM-25 score is a function of term frequency within a document, and the two documents here have the same term frequency. Interestingly, the documents with more matches may not have a higher score, as shown in Fig 3a, where the two top results have slightly higher scores than the third result, despite the latter having more matches within the `<title>` field. This is because BM-25 is also a function of document length – it penalizes documents with longer lengths, ceteris paribus. Something unexpected, however, was that the top results for Fig 3a are from the field `<title>` when we expect it to be from `<year>`. In fact, the field of `<year>` was entirely unmatched by `MultifieldParser()`.

4 Research Trend Explorer

In this section, we examined topics presented at the Web Search and Data Mining (WSDM) conference and analyzed how trends have evolved over the years. To do this, we analyzed tokens at the n-gram level and used Document Frequency (DF) scores for our analysis.

4.1 Data structure

We retrieved documents from WSDM using the "publication venue" search field in the DBLP database, capturing publication titles and their respective publication years. The data was structured as a dictionary, with key-value pairs representing the year of publication and the corresponding publication titles. Each title was treated as an individual document, while documents from the same year were grouped into a corpus. This approach allowed us to associate each key as a separate corpus, making the analysis of yearly research trends equivalent to examining trends within each corpus. Our initial exploration revealed 1000 publications between 2017 and 2023. The specific publication counts are as follows: From 2017 to 2023, there were 19, 116, 128, 151, 181, 211, and 194 documents respectively. Due to the limited number of documents in 2017, we excluded this year from our research.

4.2 Data preprocessing

Before lemmatization, we identified terms such as 'graph neural networks' and 'graph neural network' within the titles. These phrases would be treated as distinct bigrams, resulting in lower DF scores for both, despite them representing the same concept. With lemmatization, both terms collapse to their base form – 'graph neural networks' – resulting in higher weightage and a more accurate representation of the distinct concept. We performed other preprocessing steps before lemmatization as well:

- Word tokenization using NLTK's word_tokenizer
- Case folding, converting tokens to lowercase
- Removal of punctuation, extra spaces, blank tokens, and numeric tokens

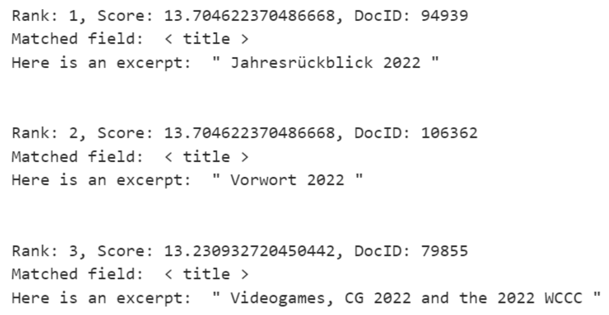
We also implemented stop word removal after the abovementioned steps. The original stop words list included *NLTK* and *Gensim* stop words. Attempts to include more stop words involved identifying tokens with low Inverse Document Frequency (IDF) and moderately low Term Frequency (TF). However, due to the limited number of publications, additional stop words could not be included. The list of stop words was then passed into `CountVectorizer()` and `TFIDFVectorizer()`, rather than directly removing them from the corpus, as titles are densely packed with keywords.

4.3 Metric selection

To ascertain the research trends within each corpus, we explored the analysis of individual words or phrases. Due to the limited document count in each corpus, we opted to experiment first with unigrams, then bigrams, and trigrams.

Term frequency (TF) represents the frequency of a particular term within the corpus. One limitation of using TF is the presence of duplicated terms in the same title. For example, in titles such as "Self-supervised Graph Structure Refinement for Graph Neural Networks." (found in the corpus of 2023), the term frequency for "graph" is 2, despite being a single publication. Consequently, a higher term frequency does not necessarily indicate greater prominence of the token "graph," and the same applies to other n-grams. Therefore, term frequency alone may not be sufficient and is not considered.

Document frequency (DF), which represents the number of documents the term appears in, addresses the

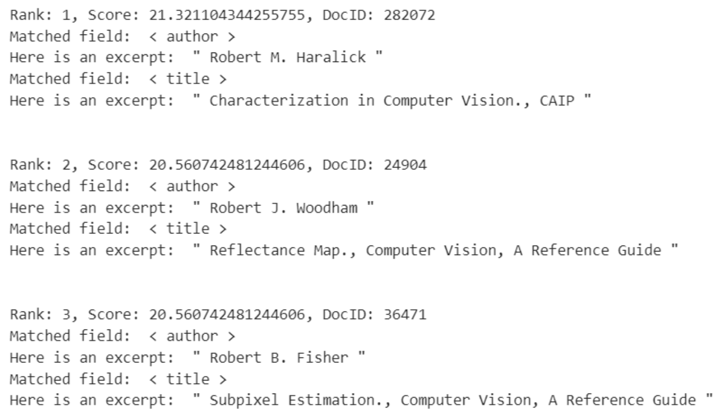


```
Rank: 1, Score: 13.704622370486668, DocID: 94939
Matched field:  < title >
Here is an excerpt:  " Jahresrückblick 2022 "


Rank: 2, Score: 13.704622370486668, DocID: 106362
Matched field:  < title >
Here is an excerpt:  " Vorwort 2022 "


Rank: 3, Score: 13.230932720450442, DocID: 79855
Matched field:  < title >
Here is an excerpt:  " Videogames, CG 2022 and the 2022 WCCC "
```

(a) Query: "2023"



```
Rank: 1, Score: 21.321104344255755, DocID: 282072
Matched field:  < author >
Here is an excerpt:  " Robert M. Haralick "
Matched field:  < title >
Here is an excerpt:  " Characterization in Computer Vision., CAIP "


Rank: 2, Score: 20.560742481244606, DocID: 24904
Matched field:  < author >
Here is an excerpt:  " Robert J. Woodham "
Matched field:  < title >
Here is an excerpt:  " Reflectance Map., Computer Vision, A Reference Guide "


Rank: 3, Score: 20.560742481244606, DocID: 36471
Matched field:  < author >
Here is an excerpt:  " Robert B. Fisher "
Matched field:  < title >
Here is an excerpt:  " Subpixel Estimation., Computer Vision, A Reference Guide "
```

(b) Query: "Robert Computer Vision"

Figure 3. Documents retrieved from our search engine, with different queries

issue above. Using the example above, the DF of the token "graph" increases by one. This adjustment provides a more accurate measure of token prominence, as the increase in document frequency reflects the number of documents featuring the corresponding token. Combining TF and the inverse of DF, we obtain another commonly used metric called TFIDF.

In our project, we considered two metrics as potential proxies for research trends:

- TFIDF scores: Tokens with high TFIDF values from the same corpus tend to correspond to keywords in general
- Document Frequency: In general, tokens with high document frequency tend to be stopwords, but our documents are titles, which are usually succinct and consist mainly of keywords. Therefore, tokens with high document frequency might correspond to popular research topics.

To identify the research trend in each corpus, we calculated the TFIDF and DF scores of all the terms in a given corpus. Terms here may refer to individual tokens or phrases. We explored the metrics using the 2023 corpus, as it contains one of the highest numbers of publications.

Unigrams only
The results from the experiments are shown in Fig 6 (Appendix). We visualized the results using both bar graphs and word cloud. While the word clouds could visualize more terms, the bar graphs provide the precise scores of the corresponding terms. Consequently, we deemed it advantageous to utilize the bar chart for identifying general trends in subsequent iterations.

The sequence of top tokens, evaluated by both DF and TFIDF score, yields consistent results, evident in both the bar chart and word cloud. However, by only using unigrams (i.e., individual token), we lack contextual nuance. For instance, the token ‘learn’ can refer to phrases such as ’self-directed learning’ or ’machine learning’. To circumvent this, we explore the use of bigrams and trigrams instead.

Bigrams and trigrams
The results from the experiments are shown in Fig 7 (Appendix).By having only bigrams and trigrams, the phrases seem to contain more contextual information. While the top 3 tokens exhibit slight variations in rankings, there are also subtle differences among the remaining 7 tokens.

In an effort to explore further, we experimented with substituting bigrams with 4-grams. However, the empirical results were not ideal due to the high number of consecutive tokens involved.The abundance of tokens at a higher n-gram order posed a challenge in extracting meaningful insights.

4.4 Results and discussion

From our initial exploration, we decided to use DF score with bigrams and trigrams. While dealing with a corpus abundant in non-keywords, TFIDF generally proves to be a superior choice for identifying trending topics, but in the case of analyzing title data, which is saturated by keywords, we found DF to be more useful. Moreover, the interpretability of DF adds to its appeal.

In Fig 4 below, each bar graph shows the top 10 terms with highest document frequency, where the terms are either bigrams or trigrams, in a given year. Notice that the term with the highest frequency in each year is a bigram. This is true regardless of the corpus since any document that contains the trigram $w_1w_2w_3$ must contain the bigram w_1w_2, whereas the converse will not be true. As such, choosing n to be $a \leq n \leq b$ biases the representative key phrase to be an a-gram. From here, it might seem that fixing n=a will suffice, but this does not guarantee a proper phrase. For instance, the term with the highest DF score in 2022 is "graph neural", which is not a valid phrase. Without the inclusion of trigrams, we may not have enough context to determine what "graph neural" actually represents. However, notice that "graph neural network" also appears as one of the terms with the highest document frequency. Hence, we can deduce that the representative key phrase for 2022 will most likely be "graph neural network".

Relying solely on n-grams may not be sufficient, and the selection of representative key phrases requires human judgment. Heuristically, we look at the top k (say $k = 5$) terms with the highest DF score or until we see a trigram, whichever is earlier. For example, in 2023, we will look at the top three terms, since the third term is the trigram "graph neural network". The remaining bigrams (pass the trigram) appear fewer times than the trigram, indicating that they are rather rare, and could be disregarded. We can aggregate these terms with the highest DF score and decide whether the different bigrams and trigrams could be semantically related. In the example of the 2023 corpus, the top three terms seem to be related to the concept of "graph neural network". Hence, we could choose "graph neural network" as our representative key phrase. If, however, the terms are not semantically related (an example would be the bar graphs in 2020), we can simply pick the first bigram, assuming the bigram is coherent. Although this heuristic do not always produce the best phrase, it narrows down the search to at most k.

With our result, the representative key phrases, from 2017 to 2023, are 'online social network', 'web search', 'knowledge graph', 'neural network', 'graph neural network', 'graph neural network', 'graph neural network' . In particular, we observe that the initial trending research areas tend to be related to the web, such as 'online social networks'. This trend was taken over by the increased interest in 'neural networks', and finally being more specific to 'graph neural networks'.

4.5 Limitations

The method outlined above faces challenges in modeling semantic connections between terms, treating similar concepts as distinct. While manual tuning by domain experts could address this, it may be impractical for a more extensive range of years. Conversely, our method may not always produce fine-grained results. For example, in one of the years, we identified 'neural network' as the main research topic, without more context about the type of neural network. If another conference such as the Special Interest Group on Information Retrieval (SIGIR) is chosen, the above method may yield the phrase "information retrieval" as the main research trend over the years. This phrase, however, is too generic as one would expect this conference to have a heavy focus on this field. As such, finer details would provide more useful context in such cases.

Some other limitations also come with the use of an n-gram model. Firstly, the model only looks at neighboring tokens in each document. By doing this, we implicitly assumed that the key phrases must come directly from the title. For instance, in the 2023 corpus, we have "Hate Speech: Detection, Mitigation, and Beyond." as a title. Here, the research area is 'social media', but this term is not found within the title itself. Also, the terms must have a strict sequential order. So, terms like "retrieving information" and "information retrieval" will be treated as distinct concepts. Finally, we note that n is a parameter, and its choice will affect the type of phrase we obtain. Here, $n = 2$ or 3, implies that phrases like "reinforcement learning with human feedback" will not be chosen.

In an attempt to resolve some of the issues, our team attempted to build a bigram language model, which generates a phrase (of chosen length, k) using an approximation of probability distributions of the tokens in the corpus. The distinction is that the generated phrase need not be a pre-existing phrase within the corpus, and tokens that appear more often are more likely to be generated, while taking into account their structural relationship (i.e., which word precedes another). However, the empirical result was not very promising – the model tends to produce phrases that are not coherent. For example, when generating a phrase of length 3 from the 2023 corpus, we obtained phrases like "the summarisation from". Also, this model does not resolve the lack of semantic relation modeling between terms.

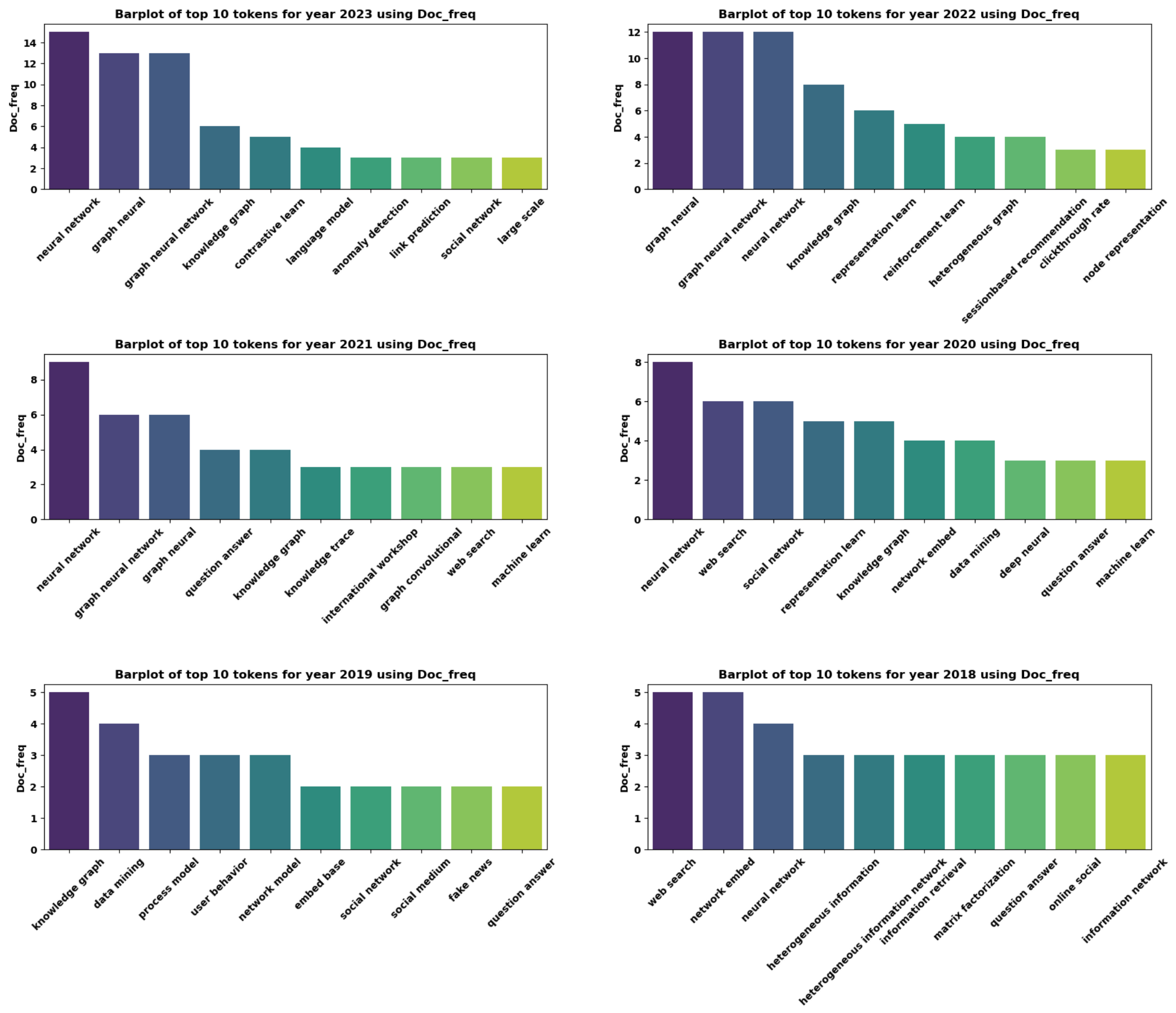


Figure 4. Top 10 terms with the highest DF scores across the years.

The overall n-gram model, relying on DF scores, presents limitations in semantic understanding and structural constraints on key phrases. To overcome these, we could incorporate semantic understanding through a training lexicon and leverage deep learning models like LSTM or Language Transformers to model word dependencies within entire documents.

5 Application

A simple search application was devised to allow the user to search for publications based on the same 4 query objects defined in 3.2, namely Title, Author, Publication Venue, and Year. Using the PySimpleGUI package to input the search parameters, if a query field is left empty, it is not included in the boolean search. Once all the queries are collected, the index is searched and using the BooleanQuery object from the whoosh package. The top results are then displayed on a pop-up window (see Fig 5).

6 Conclusion

In this project, we worked with various libraries in Python in order to perform various analysis on textual data. From this, we gained a better understanding on how textual data could be pre-processed and used for various applications such as creating a search engine.

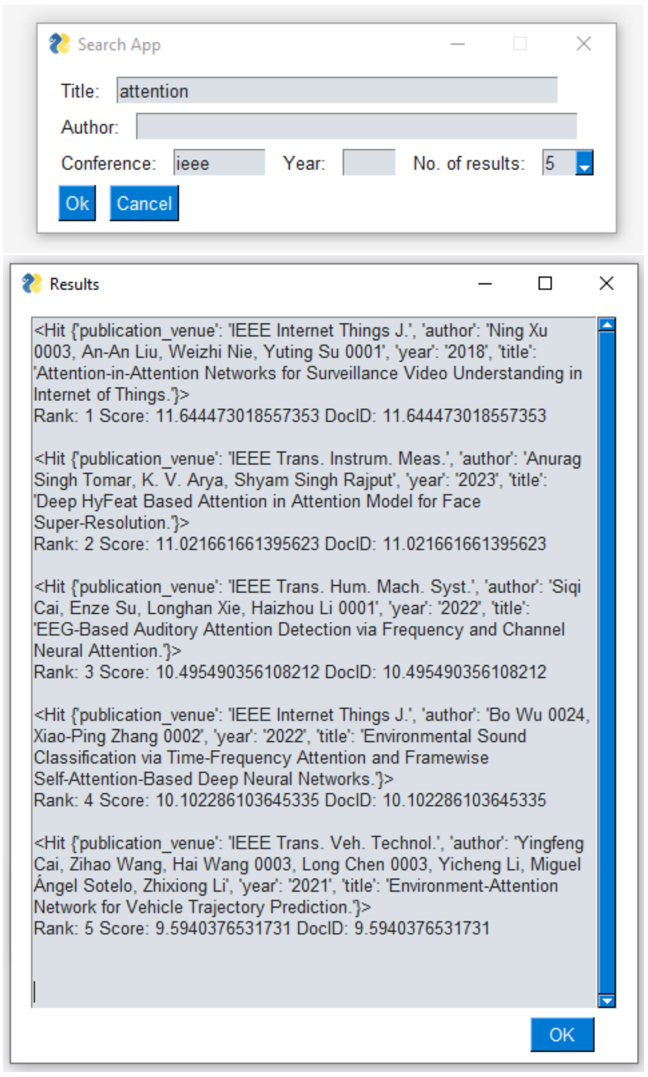
References

[1] Matt Chaput. [n. d.]. *Whoosh.* https://pypi.org/project/Whoosh/
[2] Mathieu Fenniak. [n. d.]. *PyPDF2.* https://pypi.org/project/PyPDF2/
[3] PySimpleGUI Tech LLC. [n. d.]. *PySimpleGUI.* https://pypi.org/project/Whoosh/

[4] Ewan Klein Steven Bird, Edward Loper. [n. d.]. *NLTK*. https://www.nltk.org/contribute.html

[5] Universitat Trier and Schloss Dagstuhl. [n. d.]. *dblp dataset*. https://dblp.org/

A Appendix



```
<Hit {'publication_venue': 'IEEE Internet Things J.', 'author': 'Ning Xu
0003, An-An Liu, Weizhi Nie, Yuting Su 0001', 'year': '2018', 'title':
'Attention-in-Attention Networks for Surveillance Video Understanding in
Internet of Things.'}>
Rank: 1 Score: 11.644473018557353 DocID: 11.644473018557353

<Hit {'publication_venue': 'IEEE Trans. Instrum. Meas.', 'author': 'Anurag
Singh Tomar, K. V. Arya, Shyam Singh Rajput', 'year': '2023', 'title':
'Deep HyFeat Based Attention in Attention Model for Face
Super-Resolution.'}>
Rank: 2 Score: 11.021661661395623 DocID: 11.021661661395623

<Hit {'publication_venue': 'IEEE Trans. Hum. Mach. Syst.', 'author': 'Siqi
Cai, Enze Su, Longhan Xie, Haizhou Li 0001', 'year': '2022', 'title':
'EEG-Based Auditory Attention Detection via Frequency and Channel
Neural Attention.'}>
Rank: 3 Score: 10.495490356108212 DocID: 10.495490356108212

<Hit {'publication_venue': 'IEEE Internet Things J.', 'author': 'Bo Wu 0024,
Xiao-Ping Zhang 0002', 'year': '2022', 'title': 'Environmental Sound
Classification via Time-Frequency Attention and Framewise
Self-Attention-Based Deep Neural Networks.'}>
Rank: 4 Score: 10.102286103645335 DocID: 10.102286103645335

<Hit {'publication_venue': 'IEEE Trans. Veh. Technol.', 'author': 'Yingfeng
Cai, Zihao Wang, Hai Wang 0003, Long Chen 0003, Yicheng Li, Miguel
Ángel Sotelo, Zhixiong Li', 'year': '2021', 'title': 'Environment-Attention
Network for Vehicle Trajectory Prediction.'}>
Rank: 5 Score: 9.5940376531731 DocID: 9.5940376531731
```

Figure 5. Pop-up window and result UI

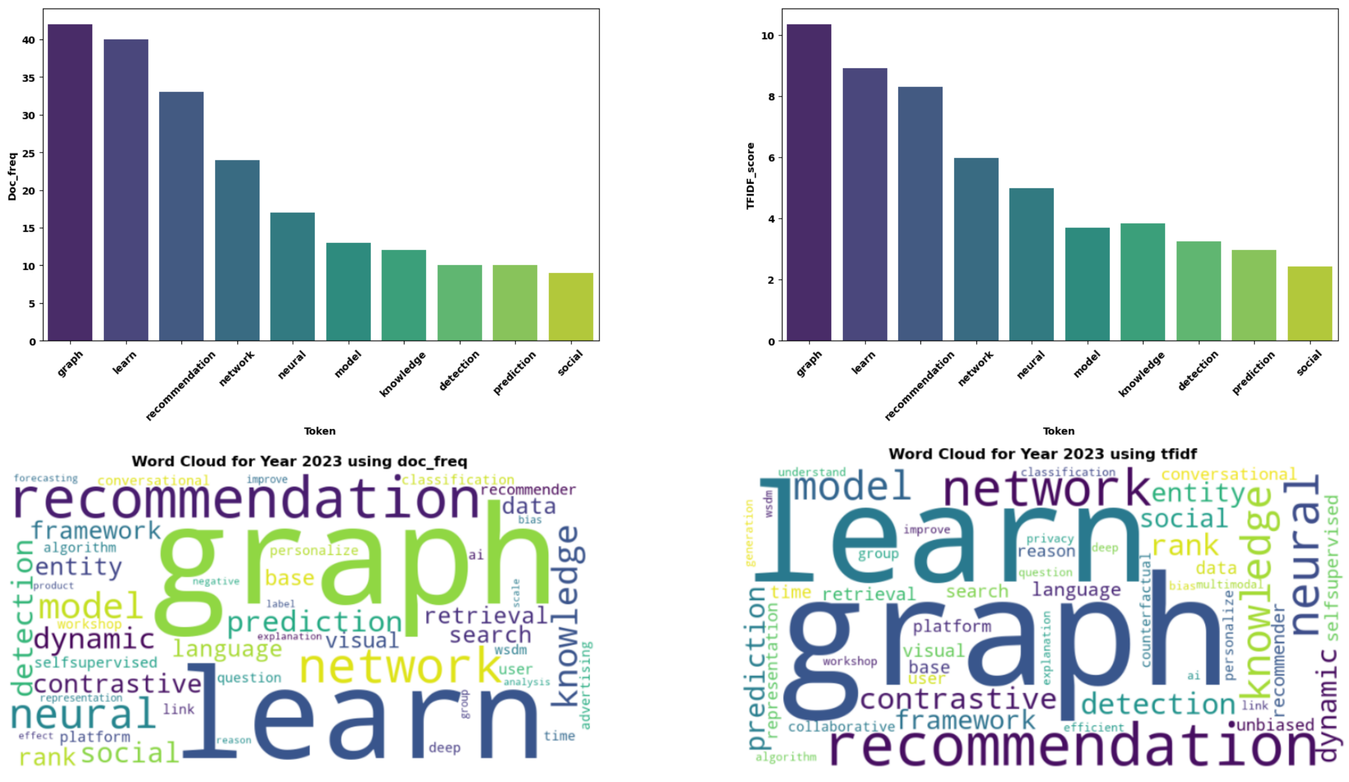


Figure 6. Exploration with unigrams using TFIDF and DF Scores

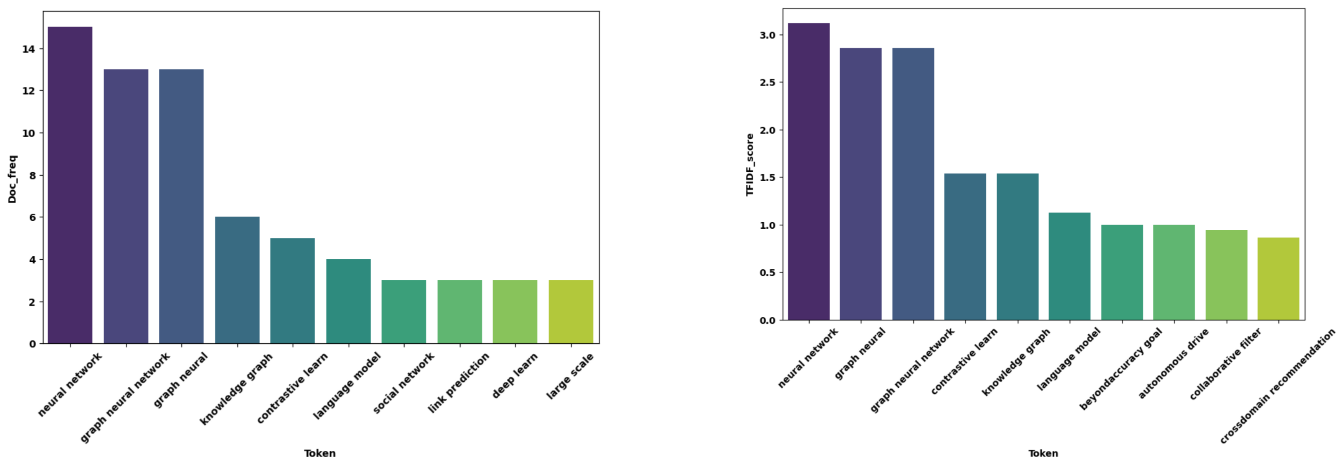


Figure 7. Exploration with bigrams and trigrams using TFIDF and DF Scores