Embedding-based Scientific Literature Discovery in a Text Editor Application

Onur Gökçe, Jonathan Prada, Nikola I. Nikolov, Nianlong Gu, Richard H.R. Hahnloser
Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland
{onur, johny, niniko, nianlong, rich}@ini.ethz.ch

Abstract

Each claim in a research paper requires all relevant prior knowledge to be discovered, assimilated, and appropriately cited. However, despite the availability of powerful search engines and sophisticated text editing software, discovering relevant papers and integrating the knowledge into a manuscript remain complex tasks associated with high cognitive load. To define comprehensive search queries requires strong motivation from authors, irrespective of their familiarity with the research field. Moreover, switching between independent applications for literature discovery, bibliography management, reading papers, and writing text burdens authors further and interrupts their creative process. Here, we present a web application that combines text editing and literature discovery in an interactive user interface. The application is equipped with a search engine that couples Boolean keyword filtering with nearest neighbor search over text embeddings, providing a discovery experience tuned to an author's manuscript and his interests. Our application aims to take a step towards more enjoyable and effortless academic writing.

The demo of the application[1] and a short video tutorial[2] are available online.

1 Introduction

Writing is a complex problem-solving task that burdens authors with a high cognitive load (Hayes, 2012), which especially applies to inexperienced researchers (Shah et al., 2009). The typical workflow of composing an academic manuscript (be it a proposal, report, or paper) is an iterative process of conceptualizing ideas, formulating search queries, browsing search results, reading papers, eventually followed by assimilating and integrating the discovered knowledge.

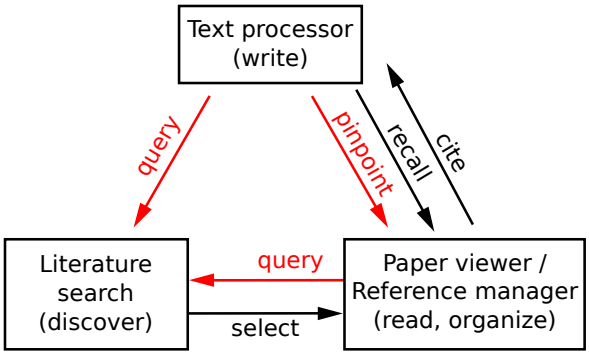

Figure 1: The typical workflow of scientific writing is largely based on independent software tools (text processor, reference manager, literature search engine, and paper viewer) that draw on diverse cognitive processes (recalling and citing articles, as well as searching, retrieving, and reading articles, black). Our web application focuses on assisting authors in literature discovery and in pinpointing relevant text passages in a paper (red).

The current toolbox of scientific writing consists of text editors, search engines, reference managers, and paper viewers. These components are typically independent applications with limited interactivity. Consequently, authors are forced to navigate through diverse user interfaces repeatedly and need to link different parts of their workflow manually. We believe that there is a need for technology that makes literature discovery a seamless extension of the writing experience (Figure 1).

Implicitly, each scientific statement requires an in-depth search for supporting or conflicting findings in the literature. Accordingly, authors must retain a strong motivation to iterate through many combinations of search terms even when the apparent gain from the search becomes sub-optimal (Azzopardi et al., 2018). In addition, the keywords intended for traditional search engines can be in-

[1] https://SciEditorDemo2020.herokuapp.com/
[2] https://youtu.be/pkdVU60IcRc

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 320–326
July 5 - July 10, 2020.

trinsically biased because authors seek confirmation (Nickerson, 1998) or because of gaps in their knowledge (Athukorala et al., 2013). The use of synonymous terminology, such as with the names of species in botany (Rivera et al., 2014) or field-specific nomenclature (Hodges, 2008), further complicates formulating comprehensive search queries. Last but not least, the exponential increase in the number of scientific publications (Larsen and von Ins, 2010) makes it increasingly difficult to keep track of the literature and to incorporate new findings into one's work.

Such challenges call for novel tools to alleviate the obstacles faced by authors. We, therefore, set out to design a workflow that simplifies the exploration of the scientific literature by making use of advances in natural language processing (NLP). We introduce a web application for writing scientific text with integrated literature discovery, paper reading, and bibliography management capabilities.

Our application allows authors to retrieve papers that are similar to their manuscript (or to some of its parts) by utilizing text embeddings (Section 3.2). In addition, the authors can confine the scope of retrieved papers to specific interests by applying keyword-based Boolean filters (Section 3.1). Finally, to guide the authors in skim reading, similar sentences can be automatically highlighted in the retrieved papers. With these features, we aim to make the processes of literature discovery and scientific writing more efficient and enjoyable.

2 Related Work

2.1 Platforms for Literature Search, Discovery, and Reference Management

Currently, there are many independent applications for searching for and sharing of publications (e.g., Google Scholar, Pubmed, Web of Science, Meta, ResearchGate, and Iris.AI), for managing bibliography (e.g., Mendeley, Readcube, Paperpile, EndNote, and F1000), and for processing text (e.g., Microsoft Word, Google Docs, Overleaf, Dropbox Paper, and Sciflow). However, end-to-end applications that combine text editing with NLP-powered interactive literature discovery are scarce. Traditionally, text processors can interact with external software to search for content, to manage references, or to improve writing style via plug-ins, but such interactions are typically limited.

A recent application, Raxter.io, provides a single interface for document writing and literature searching. Although Raxter.io allows fine-tuning of document-based search queries, its methods are not fully disclosed, and it neither supports flexible keyword definitions nor the automatic highlighting of relevant passages. Raxter.io also does not display the full body of papers unless the users manually import them.

2.2 Methods for Literature Discovery

Traditional search engines use a bag-of-words model with a frequency-based ranking function such as BM25 (Robertson, 2009) to retrieve documents that match a query of one or more search terms. Obtaining useful search results requires well-formulated search queries (Aula, 2003), which can be a challenging task during exploratory search (Belkin, 2000) and constitutes a cognitive load (Gwizdka, 2010) that our application aims to ease.

Document similarity search methods (Wan et al., 2008), by contrast, use entire documents as the search queries, circumventing the need to define keywords for the search. State-of-the-art methods for retrieving similar documents rely on text embeddings (Conneau et al., 2018; Adi et al., 2016; Le and Mikolov, 2014) and on efficient approximate nearest neighbor search algorithms (Johnson et al., 2017). However, embedding-based search methods seem rather inflexible in refining searches, because it is unclear how to steer search results in a particular direction without painstakingly having to modify the query document.

Both keyword- and embedding-based search methods provide unique advantages, but there have not been many attempts at combining these methods to overcome their respective limitations.

3 Literature Discovery

The pipeline for literature discovery in our application consists of two steps (Figure 2). First, the search engine retrieves a subset of the papers from our database that match a user-defined keyword-based filter. Second, the search engine ranks the filtered papers according to their similarity to the manuscript using document embeddings. We describe each of the two steps in detail below. Our database contains 2.7M papers from the Pubmed Central Open-Access subset (PMC-OA)[3].

[3] https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/

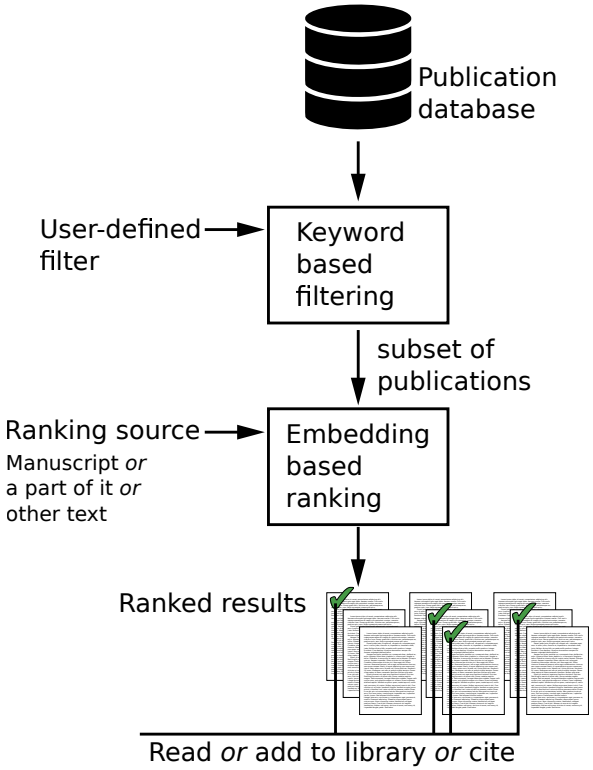

Figure 2: Overview of the literature discovery pipeline in our application. The search engine first filters our database for papers that match a set of user-defined keywords, and then ranks the filtered results according to their embedding-based proximity to a ranking source, such as an entire user manuscript. The top-ranked papers are presented to the user who can then save, cite, or read them, with the possibility of highlighting the most relevant sentences.

3.1 Keyword-based Filtering

An embedding-based search might return many papers that are similar to the manuscript but are of limited interest to the author. For example, authors of a medical manuscript on *lung cancer* may seek similar treatments in the literature for another organ, but embedding-based ranking might retrieve papers only on lung cancer. The keyword-based filter can, in such cases, be used to restrict the ranking operation either to the papers mentioning that *other organ* or to papers that do not mention *lung*. Thus, filtering allows an author to focus the nearest neighbor search on the target keywords or on their absence.

The filtering operation uses an inverted index of all unigrams in the database after the removal of stop words and word stemming (snowball) using the NLTK library[4]. The resulting index has a dictionary size of 9.61M unigrams and requires ∼4 GB of memory.

[4] https://www.nltk.org/

3.2 Embedding-based Ranking

The ranking operation uses the document embeddings of the papers in our database. Given a *ranking source* such as a paragraph or the entire manuscript, "embedding-based ranking" sorts the papers returned by the keyword-based filter according to their cosine distance to the embedding of the ranking source. In other words, embedding-based ranking performs a brute-force nearest neighbor search on a subset of papers. The embedding of the ranking source is computed on demand whenever a search is performed.

As the document embedding model, we use Sent2Vec (Pagliardini et al., 2018) because of its simplicity, speed, and good performance on various benchmark datasets (Pagliardini et al., 2018; Nikolov and Hahnloser, 2019). The model has 400 dimensions and is trained on the PMC-OA corpus using a unigram vocabulary of ∼0.75M terms. After the training, we pre-compute the embeddings of all papers in our database and keep them in memory, which requires ∼4 GB.

To test the performance of our model, we performed experiments on a simple text retrieval task. The goal of this task was to retrieve the full body of a parent paper given its abstract as the search query. We randomly sampled 10000 abstracts from the database and retrieved the 20 most similar papers for each abstract. As an evaluation metric, we counted the fraction of retrievals in which the parent paper appeared on top or among the top 20 results. Our model retrieved the correct parent paper as the top search result in 83.1% of the trials, compared to 71.0% when using a Sent2Vec model trained on Wikipedia (Pagliardini et al., 2018). Furthermore, the parent paper was among the top 20 retrievals in 95.1% of cases when using our model, compared to 87.0% for the Wikipedia Sent2Vec model. The higher retrieval performance of our model in this task likely arises from its training on a domain-specific corpus that contains rare words and terminologies (Roy et al., 2017; Blagec et al., 2019). This suggests that the model would need to be retrained at regular intervals, particularly when papers from other domains are added to the database.

We have not systematically analyzed the retrieval performance when the query is formed by merely a part of the manuscript such as a block of a few

sentences (Gong et al., 2018; De Boom et al., 2015). We leave a detailed exploration of the effects of the query length on performance to future work.

3.3 Scalability of Literature Discovery

Although fast and efficient approximate nearest neighbor methods exist for retrieving the K nearest neighbors of a query vector, such schemes apply to ranking only, but not to the joint filtering and ranking steps (when nearest neighbors are sought among a subset of embeddings from the database). For this reason, in our search engine, there is no simple alternative to brute force search. Nevertheless, we find that retrieval is sufficiently fast, largely because the filtering step reduces the number of neighbors that need to be ranked. In future work, we will explore optimizations of the search engine, such as using approximate hashing techniques (Datar et al., 2004; Norouzi et al., 2012).

4 User Interface and Workflow

The user interface (UI) consists of **(1)** a *text editor* that provides basic functionality for drafting a manuscript, such as loading saving documents, formatting text, and inserting LaTeX equations, code snippets, or bullet points (Figure 3a, left), and **(2)** a *literature explorer* encompassing multiple components, which can be accessed on their respective tabs (Figure 3a, right):

- ***Discover*** for performing searches and browsing the search results to discover relevant literature
- ***My Library*** for managing the user bibliography and for citing papers in the manuscript
- ***Read*** for paper viewing and for actions that facilitate literature exploration, such as discovering similar papers to the one being viewed and highlighting the sentences in the paper that are similar to the selected text in the manuscript (Figure 3b, right)

A search can be initiated without keyword filters by clicking the "Similar papers to the manuscript" button located above the text editor. As a result, the 1000 most similar papers are listed in the *Discover* tab with their metadata (title, authors, journal, publication year, and abstract).

A more granular search can be performed by selecting a section (e.g., sentences, paragraphs) from the manuscript, which reveals a hovering menu over the selected text (visible in Figure 3b). Clicking on the magnifying glass icon on this menu performs a search using the selected text as the ranking source and consequently returns the papers similar to the selected text.

To steer discovery towards a particular set of terms, the user can define a keyword-based Boolean filter using the format `term1 term2|term3 !term4` to confine the results to those papers that contain *term1* and (*term2* or *term3*), but not *term4*.

Clicking on a search result displays the content of the paper in the *Read* tab. In this tab, the user finds additional actions above the viewed paper to interact with it.

If, after viewing the paper, the user finds it interesting, then pressing the "Add to Library" button saves the paper in the user bibliography, which can be viewed under the *My Library* tab. Alternatively, the "Cite" button places a reference to the paper at the current cursor position in the text editor and adds the paper to the user bibliography. Inserted references in the manuscript are links, and clicking on them conveniently opens the respective paper in the *Read* tab. Deleting the link removes the reference from the manuscript.

To facilitate the exploration of the literature further, the *Read* tab contains additional functions: "Discover similar papers" performs a search using the viewed paper as the ranking source. If a filter is already present in the *Discover* tab, then the search results are filtered accordingly. The "Highlight" button highlights the 20 sentences in the viewed paper that are most similar to the ranking source, i.e., similar to the query of the last search performed on the application. Alternatively, the user can select a part of the manuscript and press the marker icon on the revealed hovering menu (Figure 3b) to highlight the sentences that are most similar to the selection. The highlighting feature computes the embedding of each sentence in the viewed paper to assess similarity. The "Find Text" field uses the web-browser's built-in *find* functionality to match the value of the field with the viewed paper.

The *My Library* tab lists all the papers in the user bibliography. Ticking the "Cited content only" box filters this list to show only the papers cited in the manuscript. The user can press the "Cite" button next to a paper to insert a reference to the paper at the cursor position in the text editor. The user can also add papers to the library manually by entering

Figure 3: The user interface of the application. a) the *Discover* tab lists the retrieved papers that are similar to the manuscript. b) the *Read* tab allows users to view papers and to highlight the sentences that are similar to the selected text in the manuscript.

the digital object identifier of the paper in the form that appears upon pressing the "Manual entry" button. Items under *My Library* can be removed by clicking on the "Remove" button next to the item.

5 Conclusion

We have described an application that aims to reduce the manual workload involved in exploring the scientific literature. Our application combines the processes of reading papers and of writing scientific manuscripts into a single user interface and links them using NLP algorithms.

In future work, we will focus on expanding the database to include additional domains and article sources. We will work on augmenting the workflow with automated tasks, such as suggesting references as the author writes a manuscript, or notifying users about the latest publications relevant to their work. We will also seek to improve discovery performance by testing more recent text embedding methods (e.g., BERT (Devlin et al., 2018)) and by optimizing the search for different input text lengths, such as a whole document, a paragraph, or even a single sentence.

Finally, we are aware that keyword-based Boolean filtering might be prone to the same biases and challenges inherent in the traditional search queries, as discussed above. We will investigate whether query expansion techniques (Azad and Deepak, 2019) could mitigate this issue by suggesting or automatically appending semantically related keywords to the Boolean filters.

Acknowledgements

We acknowledge support from the Swiss National Science Foundation (grant 31003A_156976). We also thank the anonymous reviewers for their useful comments.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2016. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. arXiv:1608.04207. Version 3.

Kumaripaba Athukorala, Eve Hoggan, Anu Lehtiö, Tuukka Ruotsalo, and Giulio Jacucci. 2013. Information-seeking behaviors of computer scientists: Challenges for electronic literature search tools. In *Proceedings of the American Society for Information Science and Technology*, pages 1–11.

Anne Aula. 2003. Query formulation in web information search. In *Proceedings of the IADIS International Conference on WWW/Internet (ICWI 2003)*, pages 403–410.

Hiteshwar Kumar Azad and Akshay Deepak. 2019. Query expansion techniques for information retrieval: A survey. *Information Processing & Management*, 56(5):1698–1735.

Leif Azzopardi, Paul Thomas, and Nick Craswell. 2018. Measuring the utility of search engine result pages. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '18*, pages 605–614.

Nicolas J. Belkin. 2000. Helping people find what they don't know. *Communications of the ACM*, 43(8):58–61.

Kathrin Blagec, Hong Xu, Asan Agibetov, and Matthias Samwald. 2019. Neural sentence embedding models for semantic similarity estimation in the biomedical domain. *BMC bioinformatics*, 20(1):178.

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. arXiv:1805.01070. Version 2.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. page 253.

Cedric De Boom, Steven Van Canneyt, Steven Bohez, Thomas Demeester, and Bart Dhoedt. 2015. Learning semantic similarity for very short texts. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1229–1234.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805. Version 2.

Hongyu Gong, Tarek Sakakini, Suma Bhat, and JinJun Xiong. 2018. Document similarity for texts of varying lengths via hidden topics. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2341–2351.

Jacek Gwizdka. 2010. Distribution of cognitive load in web search. *Journal of the American Society for Information Science and Technology*, 61(11):2167–2187.

John R. Hayes. 2012. Modeling and remodeling writing. *Written Communication*, 29(3):369–388.

Karen E Hodges. 2008. Defining the problem: terminology and progress in ecology. *Frontiers in Ecology and the Environment*, 6(1):35–42.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. arXiv:1702.08734. Version 1.

Peder Olesen Larsen and Markus von Ins. 2010. The rate of growth in scientific publication and the decline in coverage provided by science citation index. *Scientometrics*, 84(3):575–603.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196.

Raymond S. Nickerson. 1998. Confirmation bias: A ubiquitous phenomenon in many guises. *Review of General Psychology*, 2(2):175–220.

Nikola I Nikolov and Richard H R Hahnloser. 2019. Large-scale hierarchical alignment for data-driven text rewriting. In *Proceedings of Recent Advances in Natural Language Processing*, pages 844–853.

M. Norouzi, A. Punjani, and D. J. Fleet. 2012. Fast search in hamming space with multi-index hashing. pages 3108–3115.

Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. 2018. Unsupervised learning of sentence embeddings using compositional n-gram features. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers*, pages 528–540.

Diego Rivera, Robert Allkin, Concepción Obón, Francisco Alcaraz, Rob Verpoorte, and Michael Heinrich. 2014. What is in a name? the need for accurate scientific nomenclature for plants. *Journal of Ethnopharmacology*, 152(3):393–402.

Stephen Robertson. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Arpita Roy, Youngja Park, and Shimei Pan. 2017. Learning domain-specific word embeddings from sparse cybersecurity texts. arXiv:1709.07470. Version 1.

Jatin Shah, Anand Shah, and Ricardo Pietrobon. 2009. Scientific writing of novice researchers: what difficulties and encouragements do they encounter? *Academic Medicine*, 84(4):511–6.

Xiaojun Wan, Jianwu Yang, and Jianguo Xiao. 2008. Towards a unified approach to document similarity search using manifold-ranking of blocks. *Information Processing & Management*, 44(3):1032–1048.