PaperWhiz: An NLP-based Recommender System for Machine Learning Papers

Yassine Rodani
yassine.rodani@gmail.com

Abstract

The exponential growth of scientific research in recent years has made it increasingly challenging for researchers, students, and science enthusiasts to stay informed about the latest discoveries and developments in their respective fields. Traditional methods of tracking research papers, such as browsing through academic journals or regularly visiting research platforms, can be time-consuming and inefficient [1]. In this paper, I introduce PaperWhiz, an NLP-based recommender system that can aid not just researchers and data scientists, but also eager students in discovering valuable machine learning papers to explore. The system is designed to recommend germane papers tailored to a user's specific interests.

1. Introduction

In the modern age of rapid technological advancements, it is crucial for scientific community members to remain up-to-date with the most recent findings and advancements in their particular domains. Disseminating pre-peer-reviewed papers on electronic preprint repositories like arXiv has emerged as a favored approach for AI researchers to share their work beyond conventional publication channels. These platforms enable researchers to circulate their results prior to journal and conference submissions, expediting the process of information exploration. Between 2010 and 2021, the overall count of publications on AI has experienced a substantial growth, more than doubling from 200,000 to nearly 500,000 as shown in Figure 1. In addition, there has been a noticeable shift in the type of publications. Initially, most of the literature was largely limited to theoretical discussions and speculations about AI's potential. However, post-2021, there has been a surge in empirical studies, showcasing practical implementations and real-world impacts of AI technologies. This surge in AI publications underscores the rapidly expanding interest and investment in the field, showcasing its growing importance in today's technology-driven world [2].

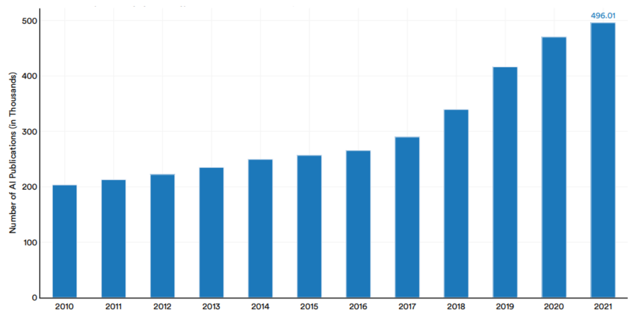


Figure 1. Global number of publications on AI between 2010 and 2021.

The sheer volume of AI publications creates a challenge for AI enthusiasts to keep up with the most relevant and recent findings. This is where Natural Language Processing can play a pivotal role. By leveraging the capabilities of NLP, AI enthusiasts can efficiently navigate through the extensive corpus of papers and identify the most pertinent research to read. One prominent use case of NLP in this context is the development of recommendation systems that suggest research papers to users based on their personal interests. These systems utilize various NLP techniques, such as topic modeling, semantic similarity, and sentiment analysis, to understand the content of the papers and the users' preferences [3] [4]. By analyzing the textual information in abstracts, titles, and even the full text of articles, NLP-powered recommendation systems can discern the relevance of a paper to the user's interests and suggest the most fitting research to read.

Taking inspiration from my own struggles with the overwhelming volume of AI research papers during my internship, I decide to build a system that recommends what machine learning paper to read. For instance, a researcher working on the application of transformers in computer vision can input a set of keywords or a brief description of their interests into the recommendation system. The system then processes the input, evaluates the similarity between the user's interests and the available research, and recommends a list of papers that are most likely to be pertinent to the user's research domain as illustrated in Figure 2.

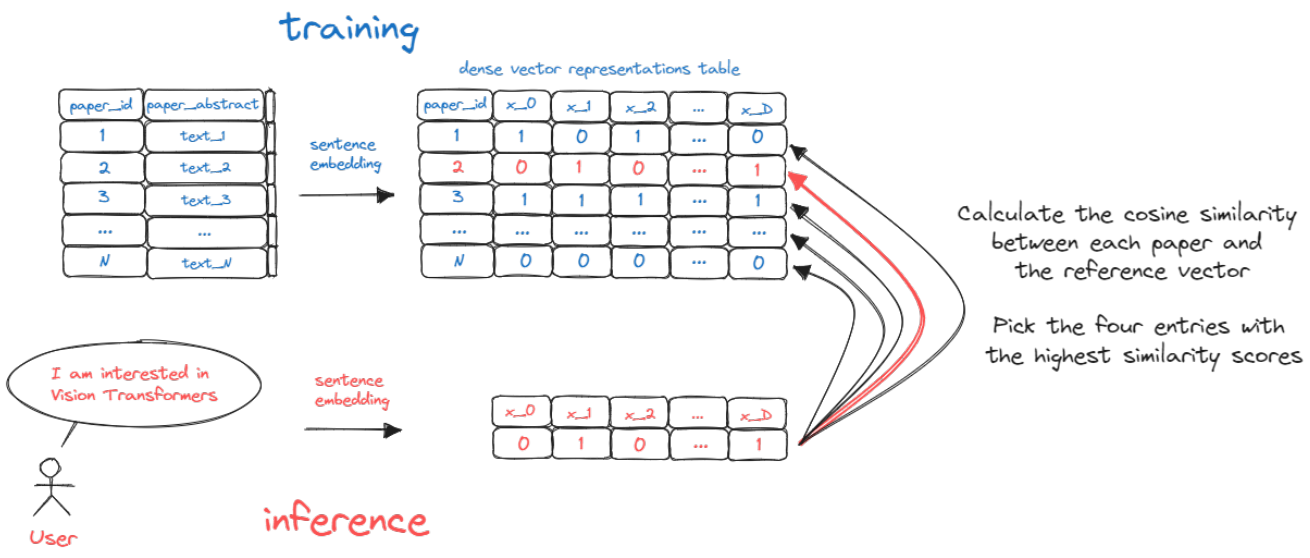

training
dense vector representations table
paper_id
paper_abstract
text_1
text_2
text_3
text_N
sentence embedding
x_0
x_1
x_2
x_D
Calculate the cosine similarity between each paper and the reference vector
Pick the four entries with the highest similarity scores
I am interested in Vision Transformers
sentence embedding
User
inference

Figure 2. Behavioral diagram of the recommender system.

2. Related Work

The field of recommendation systems has seen considerable evolution and refinement over the years, striving to provide personalized experiences for users across various domains. These systems, which aim to predict the preferences of users, have been employed widely in sectors such as e-commerce, social networking, and media streaming, proving their indispensable role in today's digital world. One of the seminal works in the domain was conducted by Resnick and Varian [5], who elucidated the principles of recommender systems and discussed collaborative filtering, one of the earliest and still widely used techniques in the field.

The integration of Natural Language Processing (NLP) techniques has further enriched the capabilities of recommender systems. The advent of NLP has empowered these systems to process and understand unstructured textual data, enabling a more nuanced understanding of user preferences and content characteristics. For instance, topic modeling and semantic similarity measures have been used to map the latent topics in documents and estimate the similarity between different pieces of content, providing a more targeted and relevant recommendation experience. A key contribution in this regard was made by Blei, Ng, and Jordan [6], who introduced the concept of Latent Dirichlet Allocation (LDA), a generative probabilistic model for collections of discrete data such as text corpora. This work revolutionized the way we could extract topics from large volumes of text, and has found extensive use in NLP-based recommender systems.

In more recent times, the application of deep learning techniques in NLP has further augmented the efficacy of recommender systems. Le and Mikolov's work on Word2Vec, a two-layer neural network that processes text, has been pivotal in understanding the semantic context of words, enhancing the ability of these systems to discern user preferences and recommend relevant content [7]. Moreover, the advent of transformer models, as discussed in Vaswani et al.'s groundbreaking paper, "Attention is All You Need", has provided new avenues for understanding the context in language, enabling recommender systems to make even more accurate predictions [8].

3. Materials and Methods

3.1. Data

The dataset was collected using the $arXiv$ Python library that provides a wrapper around the original arXiv API, an open-access repository of electronic preprints (known as e-prints) approved for posting after moderation, but not full peer review. Here's how it works step by step:

1. A list of query keywords is defined which includes various terms related to AI and machine learning research.
2. The $query\_with\_keywords$ function is defined. This function queries the arXiv API for research papers matching a given query. It filters the results to include only those whose primary category is among: "$cs.CV$", "$stat.ML$", "$cs.LG$", "$cs.AI$". For each result matching these categories, it appends the categories (terms), title, summary (abstract), and URL of the paper to their respective lists. These lists are then returned by the function.

3. An arXiv API client is created with 20 retries and a page size of 500 results
4. For each keyword in the $query_keywords$ list, the $query\_with\_keywords$ function is called with the keyword and the client. The results are added to master lists for titles, abstracts, terms, and URLs.
5. After all keywords have been processed, a pandas DataFrame is created from the master lists.

3.2. Methods

To embed textual data into a numerical space, typically for natural language processing tasks, Transformer models from the Hugging Face library are commonly used. Transformer models are based on the self-attention mechanism and have been very successful in various tasks, including language translation, text generation, and language understanding.

- **Sentence Transformers:** Sentence Transformers is designed to generate semantically meaningful sentence embeddings efficiently. Unlike word-level transformers such as BERT that generate embeddings for each word in a sentence, Sentence Transformers generate a single fixed-length vector representation for the entire input sentence. It does this by fine-tuning transformer models on specific tasks that encourage the model to encode the semantic meaning of sentences into their embeddings [9]. The main advantage of Sentence Transformers is that it allows for the computation of semantically meaningful sentence embeddings, which can be compared using simple distance metrics such as cosine similarity or Euclidean distance. This makes it useful for a variety of tasks, including semantic textual similarity, clustering, information retrieval, and others.

- **Cosine Similarity:** Cosine similarity is a metric that is often used in tandem with Sentence Transformers to determine the semantic similarity between different sentences. When Sentence Transformers encode sentences into vector representations, or embeddings, these embeddings capture the semantic content of the sentences in their dimensions. Thus, two sentences that have similar meanings should result in embeddings that are close together in the high-dimensional vector space. This is where cosine similarity comes into play. By calculating the cosine of the angle between two sentence embeddings, we can quantify how closely related the sentences are in terms of their semantic content. A cosine similarity of 1 indicates that the sentences are extremely similar (or even identical), a score of -1 indicates that they are dissimilar, and a score of 0 indicates that they are unrelated. The cosine similarity between two vectors (A and B) is calculated using the following Formula 1:

$$cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} \tag{1}$$

  Here, $A.B$ denotes the dot product of vectors A and B, and $||A||$ and $||B||$ denote the magnitudes (or lengths) of vectors A and B, respectively.In the context of sentence embeddings, each vector would represent the sentence embedding of a sentence, and each dimension in the vector corresponds to a feature learned by the Sentence Transformer model.

  In practice, this allows us to perform tasks like information retrieval (finding the most similar sentence in a database to a given query), clustering (grouping together sentences that have similar meanings), and duplicate detection (finding sentences that express the same idea in a corpus), among others.

4. Model

MiniLM is a smaller variant of the BERT model which has been designed to provide high-quality language understanding capabilities while being significantly smaller and more efficient. The "all-MiniLM-L6-v2" model refers to a specific configuration of the MiniLM model [10]. Here are some reasons why I have chosen this model for my project:

- **Efficiency:** MiniLM models are smaller and faster than full-size BERT models, which can be a major advantage if you're working on a project with limited computational resources or if you need to process large amounts of data quickly.

- **Performance:** Despite their smaller size, MiniLM models often perform at a comparable level to full-size BERT models on a variety of NLP tasks. This means that you can often use a MiniLM model without sacrificing much in the way of performance. In fact, the 'Performance Sentence Embeddings' metric which is the average performance on encoding sentences over 14 diverse tasks from different domains is 68.06 for the "all-MiniLM-L6-v2 model", which is very good to start with.

- **Ease of Use:** If you're using a library like Hugging Face's Transformers, it can be relatively straightforward to load a pre-trained MiniLM model and finetune it for your specific task.

- **Lower Memory Requirements:** Given its smaller size, MiniLM requires less memory for training and inference. This could be a crucial factor if you're working with limited hardware resources.

5. Observations

5.1. Distribution of titles' length

The average text length of a title is 73 characters, and its maximum length is 217 characters. Choosing a sentence-transformer model with a Max Sequence Length capability of over 217 characters would be ideal. The 'all-MiniLM-L6-v2' model has a $MaxSequenceLength$ of 256, which is more than enough to process the entire title.

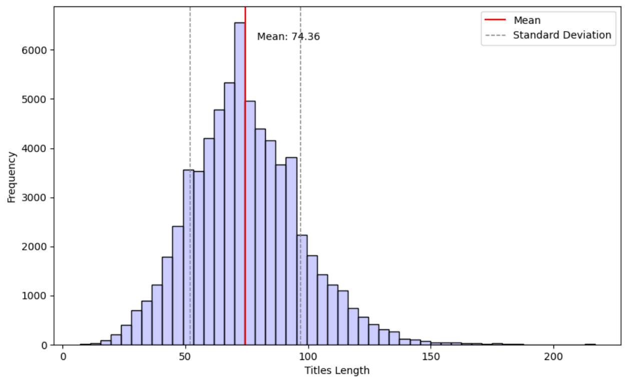


Figure 3. Titles length's distribution.

5.2. Distribution of abstracts' length

The average text length of an abstract is 1237 characters, and its maximum length is 2790 characters. So the best choice is to use a sentence-transformer model with a Max Sequence Length capability of over 2790 characters. Unfortunately, most pre-trained models have a Max Sequence Length of 512. The transformer model just won't be able to process the entire abstract at once due to its max length constraint, so it processes as much as it can, which in this case is the first 512 words.

A naive approach would be to split the document into chunks, encode each chunk separately and then combine these encodings for a final document-level representation. For example, split a document into sentences, encode each sentence independently and then combine these sentence vectors (e.g., averaging, max-pooling, etc.) for a document representation.

Another approach would be to use a "sliding window". Instead of just taking the first 512 words, I could apply a "sliding window" approach where I first process the first 512 words, then the next 512 words (perhaps with some overlap), and so on until I've processed the whole abstract.

6. Proposed Approach

The main aim is to build an end-to-end machine learning product where the objective is to help not only researchers

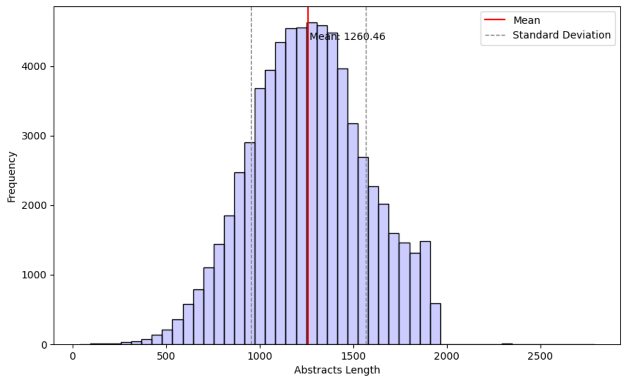


Figure 4. Abstracts length's distribution.

and data scientists, but also eager students, by recommending germane papers specifically aligned with their interests. This idea was sparked by the observation that much of academic research was scattered and often unorganized, and navigating it was a daunting task for many, especially newcomers.

6.1. Problem Framing

The first step involved defining the problem and identifying the target audience. The issue at hand pertains to the realm of academic research. As research expands exponentially, it has become difficult for stakeholders such as researchers, students, and science enthusiasts to keep up with the latest findings and advancements in their areas of interest. The conventional methods of tracking research papers, like going through academic journals or frequenting research platforms, are turning out to be inefficient and time-consuming. The primary machine learning problem to tackle here is the recommendation of research papers. Specifically, the ML problem could be categorized as a content-based recommendation system, where the goal is to recommend research papers based on a user's specific interests.

6.2. Feature Pipeline

The heart of this content-based recommendation system is the feature pipeline as shown in Figure 5. This pipeline is specifically designed to streamline data collection and storage, and it plays a pivotal role in this project. I established a data pipeline that uses the arXiv API to scrape the latest research papers from the arXiv website on a monthly basis. A Jupyter notebook was developed to automate this process, providing us with a consistent stream of new data, which is key for keeping the recommendations current and relevant.

Every month, the notebook is triggered to run via Github workflows. This notebook executes several tasks. Firstly, it sends requests to the arXiv API, receiving a JSON response with details of newly published papers. The features

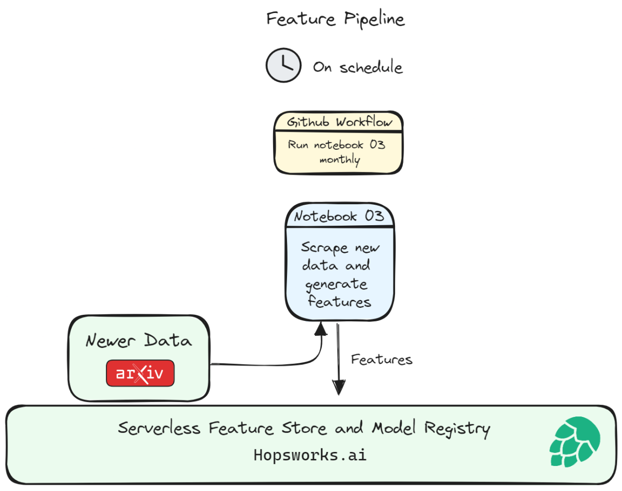


Figure 5. Feature Pipeline.

we extract from the response include metadata such as title, summary, and links to the paper, along with information about the categories the paper belongs to. This way, we can categorize and recommend papers based on different fields of study.

Once the data is scraped and cleaned, it is stored in the HopsWorks Feature Store, our chosen storage system. The Feature Store offers a multitude of benefits for this project. Being serverless, it allows us to scale up or down depending on our needs, thereby efficiently managing resources. It also allows us to define and manage our features at one place, making the features reusable across different machine learning projects. Importantly, this centralized system enables us to maintain data consistency and ensure high data quality.

By automating the data collection and storage process, I have built a feature pipeline that consistently provides fresh data for the machine learning model. This strategy ensures that the recommendation system stays dynamic and adaptable, capable of serving up-to-date recommendations to users who are following rapidly evolving research fields.

6.3. Embedding Pipeline

The embedding pipeline forms the next crucial stage in this machine learning project, transforming our raw data into a format that can be effectively understood and processed by our recommendation model. See Figure 6. This pipeline has been carefully designed to leverage the power of modern Natural Language Processing (NLP) techniques, transforming text data into meaningful and contextually rich sentence embeddings.

This transformation process begins with a Jupyter notebook extracting the required features from the Hopsworks Feature Store. The primary feature of interest for the embedding pipeline is the textual data from the papers, such as the title and the abstract. These features contain rich information about the paper's topic, methods, and findings, making them highly valuable for generating accurate and contextually-aware recommendations.

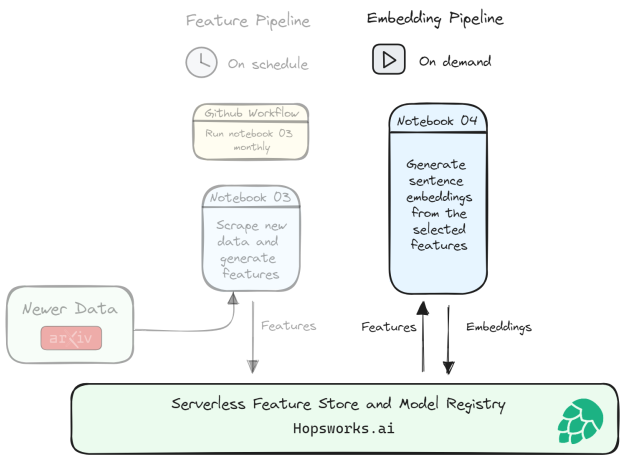


Figure 6. Embedding Pipeline.

Once the features have been retrieved, we employ a Sentence Transformer to convert the raw text data into sentence embeddings. The Sentence Transformer is a powerful model that uses transformer-based architectures like BERT, RoBERTa, or DistilBERT, to generate high-quality sentence embeddings. These embeddings are dense vector representations that capture the semantic meaning and context of the sentences. By representing the text data in this way, we can efficiently compare different papers based on their semantic similarity.

After generating the sentence embeddings, the notebook stores the resulting model in the Hopsworks Model Registry. The Model Registry provides a central location for managing and versioning our models, ensuring that we can reliably and easily access the appropriate model when generating recommendations.

An important aspect of our embedding pipeline is its on-demand nature. Unlike the data collection phase, which is automated to run monthly via Github workflows, the notebook responsible for generating sentence embeddings and storing the model is triggered manually. The decision to opt for an on-demand process stems from the nature of the task. While the data collection needs to be performed regularly to keep up with the continuous influx of new research papers, the generation of sentence embeddings is a more resource-intensive process and doesn't necessarily require the same frequency. By triggering this process on-demand, we can manage resources more efficiently and have more control over when and how the embeddings are generated. This approach also allows us to incorporate any significant updates or improvements to the Sentence Transformer or any other part of the embedding process as soon as they become available.

Through this embedding pipeline, we are able to convert our vast and complex dataset into a highly effective, compact, and contextually-aware format, significantly enhancing the quality and accuracy of our paper recommendations.

6.4. Inference Pipeline

The final piece of our machine learning product is the inference pipeline, responsible for making the actual paper recommendations to the user. This pipeline is designed to be interactive and user-friendly, and it utilizes a Streamlit application to deliver a seamless experience, as shown in Figure 7.

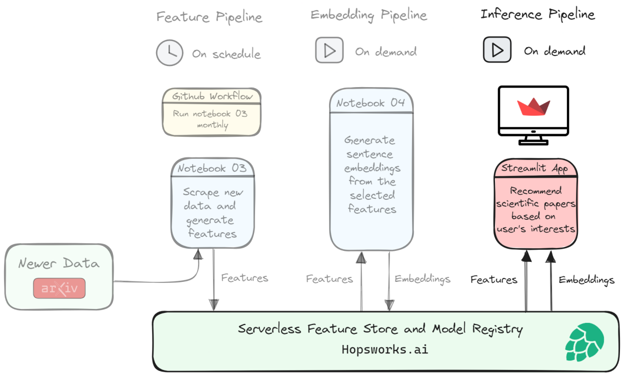


Figure 7. Inference Pipeline.

The inference pipeline is set into motion when a user makes a request for paper recommendations through the Streamlit application. The application then retrieves the necessary features and the most recent model from the Hopsworks Feature Store and Model Registry, respectively. This ensures that the recommendations are made using the most up-to-date data and model.

The main task of the inference pipeline is to calculate the similarity between papers. This is achieved using cosine similarity, a measure that calculates the cosine of the angle between two vectors. In our case, these vectors are the sentence embeddings of the titles and abstracts of the papers. Two separate similarity scores are calculated - one by comparing the titles and another by comparing the abstracts.

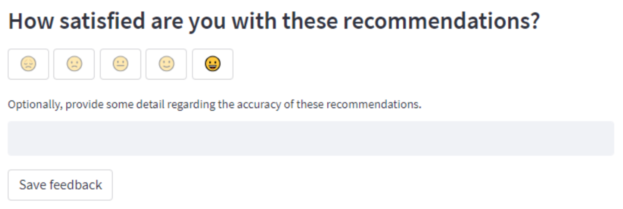
Once the similarity scores have been computed, the Streamlit application recommends the top four most similar papers to the user. These recommendations are made based on a combination of the title and abstract similarity scores. The recommended papers are presented to the user along with their abstracts and links to the full text, enabling users to quickly access and review the papers of interest.

To close the feedback loop, I also implemented a way for users to provide their feedback on the recommendations. Using the Trubrics feedback component within Streamlit, users can rate the relevance and usefulness of the recommended papers. This feedback will be invaluable for further refining and improving our recommendation system. However, the details of this feedback collection and how it will be incorporated into the system will be elaborated on in the next section of the report.

Through the inference pipeline, our end-to-end machine learning product fulfills its aim of helping researchers, data scientists, and students navigate the vast world of academic papers and discover the ones most relevant to their interests.

7. Feedback Component

The incorporation of the Trubrics Feedback Component represents a critical element in this end-to-end machine learning product. Trubrics is a pioneering user insights platform designed specifically for AI, allowing us to collect, analyze, and manage user feedback on our AI model in a streamlined and efficient manner.


How satisfied are you with these recommendations?

😞 🙁 😐 🙂 😀

Optionally, provide some detail regarding the accuracy of these recommendations.

Figure 8. Trubrics Feedback Component.

There are several key reasons why we value and prioritize the collection of user feedback in our machine learning process:

- **Improving model performance:** Feedback from users provides valuable insights into how well our model is performing from the end-user's perspective. Understanding our model's strengths and weaknesses through this lens allows us to make necessary adjustments and fine-tune the model, resulting in enhanced performance over time.
- **Enhancing user experience:** User feedback is a rich source of information for understanding user preferences and identifying any potential pain points. This knowledge can guide improvements in our user interfaces and usability, ultimately contributing to a superior user experience.
- **Increasing user engagement:** Actively seeking and valuing user feedback signals our commitment to creating the best possible user experience. This demonstration of care can boost user engagement, satisfaction, and loyalty, fostering a robust user base that is

more likely to regularly use and advocate for our product. See Figure 8.

- **Growing responsibility of the ML group:** User feedback collected in a live production environment enables us to keep improving our machine learning model. We can respond to user needs and preferences, fix bugs, and ensure our model remains effective and relevant as user behaviors and needs evolve.
- **Continuous model improvement and monitoring:** User feedback is a rich source of information for understanding user preferences and identifying any potential pain points. This knowledge can guide improvements in our user interfaces and usability, ultimately contributing to a superior user experience.

In conclusion, the Trubrics Feedback Component is not just a feature in this product, but an essential tool that facilitates continuous learning, improvement, and user-centric design in the machine learning product.

8. Conclusion

In the rapidly advancing technological world, staying updated with the latest findings in one's field, like AI, has become imperative. With the dramatic increase in AI-related publications from 2010 to 2021, sifting through this massive volume of work is a daunting task. This challenge prompted the development of an end-to-end machine learning product using Natural Language Processing (NLP) techniques.

This product, born out of personal experience navigating vast AI literature, is a recommendation system tailored to the user's interests. By inputting a set of keywords or a brief description of their research interests, users receive a list of relevant papers from our system. It leverages NLP techniques like topic modeling, semantic similarity, and sentiment analysis to evaluate the similarity between the user's interests and available research.

Through continuous data updates, model fine-tuning based on user feedback, and a commitment to user-friendly experience, this machine learning product aims to assist researchers, data scientists, and students navigate the vast world of AI research effectively.

Code Implementation

The GitHub repository of this project is made publicly available at: https://github.com/yassine-rd/PaperWhiz.

Acknowledgement

I would like to thank arXiv for use of its open access interoperability. I would also like to express my gratitude to the Trubrics team for offering their feedback component and granting me permission to test it out on my project.

References

[1] P. Mongeon and A. Paul-Hus, "The journal coverage of web of science and scopus: a comparative analysis," *Scientometrics*, vol. 106, pp. 213–228, 2015. 1

[2] N. Maslej, L. Fattorini, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, H. Ngo, J. C. Niebles, V. Parli, Y. Shoham, R. Wald, J. Clark, and R. Perrault, "The AI index 2023 annual report," AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, Stanford, CA, Tech. Rep., Apr. 2023. 1

[3] P. Lops, M. Degemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," in *Recommender Systems Handbook*, 2011. 1

[4] H. Wang, N. Wang, and D. Y. Yeung, "Collaborative deep learning for recommender systems," *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014. 1

[5] P. Resnick and H. R. Varian, "Recommender systems," *Commun. ACM*, vol. 40, no. 3, p. 56–58, mar 1997. [Online]. Available: https://doi.org/10.1145/245108.245121 2

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003. 2

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. 2

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. 2

[9] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084 3

[10] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," 2020. 3