RAGS: Research Assistant for Graduate Students

Shaobo Cui
shaobo@umd.edu

Ryan Sullivan
rsulli@umd.edu

Kamala Varma
kvarma@umd.edu

***Abstract*—Academics, and in particular graduate students, often struggle to manage their research goals. Achieving these goals requires strong time management skills, and good judgement of where to allocate time. One particular task that combines these challenges is choosing which papers to read. Graduate students must read papers to keep up with current trends in their field, as well as to gain knowledge of their particular research focus, but they typically have many other obligations and limited time for reading. These students, and likely all academics, would benefit from a strong recommendation system for research papers. Our work proposes the Research Assistance for Graduate Students (RAGS), a personal assistant that helps graduate students to schedule and manage time. The core feature that we present is a research paper recommendation system which suggests papers to users based on their history of previously read papers. We present this application as well as discussions of additional possible features and future research directions.**

***Index Terms*—recommender system, personal digital assistant, context-aware system**

I. Introduction

Academics face a number of unique challenges in their work. On top of the common struggles of scheduling meetings and event, researchers must often manage long term projects with shifting expectations and timelines. The uncertain nature of research makes this planning challenging. Additionally, keeping up with trends and research can be extremely difficult for graduate students, especially those that are exploring a new topic or subfield for the first time. The technical complexity of these topics increases the difficulty of finding relevant papers to read and judging their relevance. We developed the Research Assistant for Graduate Students (RAGS) to address these concerns. In this paper we specifically focus on implementing a recommendation system for relevant research papers, but we also introduce and discuss several potential future additions to address some of the challenges of graduate school. The main contributions of this paper are the design for this context-aware personal digital assistant (PDA) for graduate students, and the implementation of the associated paper recommendation system[1]. Our system is designed to work in conjunction with Zotero, which is a research reference management software wherein a user of our system stores papers that they have read or have interest in saving. We find that the system we develop produces reasonable recommendations for the given context, and discuss several avenues for

[1]Source code and usage guide available at https://github.com/RyanNavillus/CMSC818G-Project. The usage information can be found in the README file.

future improvements.

II. Related Work

To the best of our knowledge, there are only three preexisting student-specific digital assistants. The Insight mobile app comes from the University of California San Diego (USD) and is focused on helping student in prioritizing and tracking tasks. Its design is tailored specifically to USD, which provides a personal and easily applicable tool that maximizes effectiveness for its users. AtlasRTX is a PDA that comes from Purdue University and is primarily motivated by the desire to guide prospective students through online information and resources about the school when there is no human team member available. My Elite Assistant, particularly the version for academics, provides virtual services for professors, graduate students, and researchers. Services include handling email request, formatting, marketing services, proofreading and editing, website management, and other similar tasks. Like these three assistants, RAGS aims to be tailored specifically to a graduate student's lifestyle and needs. The novelty of RAGS comes from the fact that it incorporates active context information into its services, most notably, its research paper recommendation service.

III. User Interface

A core focus of this application is to limit the amount of manual data entering that users must perform. In service of this goal, we created a simple text based interface which allows users to generate replications from their Zotero history with a few minutes of set up and a couple of key presses. All of the necessary data manipulation can be done in Zotero by editing collections of previously read papers.

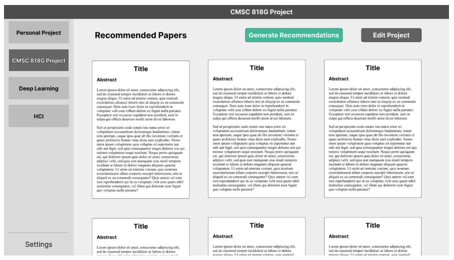


Fig. 1: A mock-up of the graphical user interface that we designed for this system to use, though we did not have time to implement.

The interface allows you to create and select different projects. These can be associated with separate Zotero collections to allow users to get recommendations for multiple different research interests. Users can then generate recommendations using the methods described in section VI.

Although we did not implement it, we also designed a user interface which replicates many of the features of Zotero's user interface, and allows users to view full abstracts of papers (mock-up in Figure). While we feel this interface would be slightly more convenient than the text-based interface, we focused on implementing the context-dependent systems for this project.

IV. Activities

Activities are events or software applications such as research conferences or TensorFlow which can be used as context to recommend reading material. For example, a user that is actively using PyTorch could be recommended relevant documentation, and a user attending a research conference might be recommended papers that are being presented at that conference. This would allow us to draw from a large variety of sources of active context to make event-based recommendations. Accurate activity detection might also enable us to offer different services in addition to paper recommendations. Activities are not included in our current implementation because it is difficult to create a cross-platform solution for detecting active applications.

V. Handling Paper Data

A. Building a Corpus of Papers

In order for our recommendation system to find papers to recommend to a user, there needs to be some dataset of papers that the recommender will look through and find the best recommendations within. To populate this, we find paper titles that Google Scholar identifies as related to the papers that a user has in their Zotero library. Google Scholar does not provide any detail regarding their method for finding related papers. From these related papers, it is possible (though not currently used in our system) to add additional papers by adding related papers associated with the related papers. This additional branching would not only grow the corpus, but it could also lead to titles that are slightly less directly related to those that a user has already added to their library and therefore offer opportunity for the user to encounter a wider array of topics. Our system can also optionally (though it is not doing this currently) add papers that are cited by a user's library papers and can continue to branch to more papers through the cited papers' cited papers. The current system implementation is able to store this corpus in a JSON file. As an example, Appendix A lists three initial paper titles along with a list of papers that were added to a corpus due to Google Scholar identifying them as being related to the the three papers. Cited by papers were not included in this example.

B. Paper Information Storage and Retrieval

A user of our system is expected to use Zotero for storing paper information. This can include papers that they have read or that they have interest in reading/saving and it aids the paper recommendation system in building user profiles and understanding user interests. We use the pyzotero API[2] to interact with a user's Zotero library. We can also automatically add recommended papers directly to a user's Zotero library. Unless a paper's data is available in a user's Zotero library, we access it through Google Scholar using the scholarly API[3]. scholarly provides basic paper information such as title and authors. It also provides a URL for a paper's publication page and a PDF URL for the full paper text. Our system is able to use either of these links to try to extract the paper's abstract. Currently, this is done by scraping the entire page and using certain text and layout patterns to try to determine which portion of the full page text corresponds to the abstract.

VI. Recommendation System

A. Content-based filtering

Content-based filtering [1] is the most commonly seen paper recommendation system in existing literature [2]. The basic idea is to rank papers in the corpus by similarity to those captured in the user profile, and recommend those ranked top by the similarity score.

We implemented a content-based filtering system for paper recommendation, using a BERT model [3] to gauge the similarity of papers. Our system uses a BERT model based on one pretrained on scientific texts [4] to convert the text of a paper into a vector

[2]https://github.com/urschrei/pyzotero
[3]https://github.com/scholarly-python-package/scholarly

recommendation. The user profile is accumulated as the user adds paper to his library, with the BERT vectorization stored into the profile. Each paper in the corpus is also vectorized with the same model.

When a recommendation is requested, the system computes the cosine similarity of each paper in the profile with a paper in the corpus, and add them up to yield a total similarity score between the paper and the user profile. The papers with the highest similarity scores are then selected and presented to the user as recommendations.

If different weighting of papers in the user profile is desired, the weights can be applied to the cosine similarity score of each paper in the profile, so that papers assigned a higher weight will be more influential to the recommendations made.

Our system allows the user to create different profiles for different projects. Users can specify keywords related to each project. When a paper being read shares a keyword with that of a project, it will be added to the profile specific to that project in addition to the generic profile. Users can select a specific profile from which they can request relevant recommendations.

To save memory consumption, we run the paper through a BERT summarization model [5] before computing the vectorization as the system we test our software runs out of memory when attempting direct vectorization of the full text. An alternative option provided is to replace the full text with the abstract, but either method will sacrifice recommendation quality in some way. A less memory-intensive method might be desirable improvement.

B. Other potential methods

Our system used a pure content-based approach, which works even with only one user of the system, but alternative recommendation methods or hybrids are possible, and a potential improvement is to combine the content-based method implemented with other recommendation algorithms to improve performance.

One potential paper recommendation scheme is based on collaborative filtering, where users rank the paper by some criteria. This might be an explicit request of rating, or a piece of data that is collected implicitly after obtaining user consent, such as whether the user read the paper or skipped it after reading a small part, or whether particular papers are viewed together with other ones. The information collected about users could form a profile, and when recommendation is requested, the system can look papers based on the history of other users with a similar profile. This method, however, requires multiple users to work, and a large number of users is needed to achieve high accuracy of recommendations. It is still possible to compute a recommendation score based on these criteria and combine that with the similarity score formed by content-based filtering in an attempt to improve the recommendation made by the system if there's a small number (but more than one) of users.

VII. Evaluation Methods

We came up with two potential methods for evaluating the quality of our recommendations. Due to the amount of data or participants that these studies would require, we were unable to use them for this paper.

First, we could complete a user study by asking users to rate the quality of our recommendations based on their Zotero collections. We can generate recommendations based on their individual Zotero libraries, and ask participants to rate the quality of the recommendation on a variety of relevant metrics, such as novelty, relevance, paper quality, and overall recommendation satisfaction. However, this would require a significant number of participants that actively use Zotero.

Second, for a sufficiently large Zotero collection, we could separate the previously read papers into a recommendation and evaluation set. The recommendation set is used to produce the recommendations as usual, and the evaluation set is used to judge the quality of the recommendations. We then take the set of recommended papers and the evaluation set, and compute their overlap. a High quality recommendation method should recommend most of the papers in the evaluation set, as those are the papers that the user judged to be relevant.

Although these methods would have been ideal evaluations, we instead chose to focus on qualitative results which were easier to compute and analyze within the constraints of this project.

VIII. Results

In general, it is difficult to evaluate how good the recommendations are without a user study. However, we believe that the recommendations made by our system at least makes some sense. We tested the recommendation algorithm by forming two different user profiles. The first one involves a user reading papers about robust federated learning, with the following papers initially in their Zotero library:

- Attack-Resistant Federated Learning with Residual-based Reweighting
- Certifiably-Robust Federated Adversarial Learning via Randomized Smoothing
- Data Poisoning Attacks Against Federated Learning Systems

The other is of a researcher examining literature about programming languages, with the following papers initially in their Zotero library:

- A Relational Theory of Effects and Coeffects
- A Formal Foundation for Symbolic Evaluation with Merging
- Concurrent Incorrectness Separation Logic

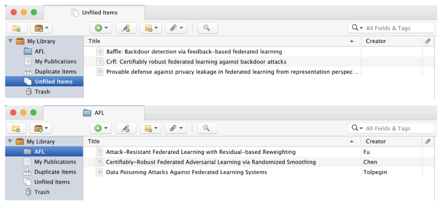
We requested recommendations for each profile with the same corpus, and the system recommended the papers about robust federated learning for the first profile, specifically:

- Baffle: Backdoor detection via feedback-based federated learning
- Crfl: Certifiably robust federated learning against backdoor attacks
- Provable defense against privacy leakage in federated learning from representation perspective

All papers recommended for the second profile are related to type systems, specifically:

- Effectful Program Distancing
- From Enhanced Coinduction towards Enhanced Induction
- Oblivious Algebraic Data Types

Figure 2 shows the papers originally in the Zotero library and the recommended papers that are automatically added to Zotero. These results imply that



Title	Creator
Baffle: Backdoor detection via feedback-based federated learning	
Crfl: Certifiably robust federated learning against backdoor attacks	
Provable defense against privacy leakage in federated learning from representation perspec...	

Title	Creator
Attack-Resistant Federated Learning with Residual-based Reweighting	Fu
Certifiably-Robust Federated Adversarial Learning via Randomized Smoothing	Chen
Data Poisoning Attacks Against Federated Learning Systems	Tolpegin

Fig. 2: Top: Papers initially in a user's Zotero library. Bottom: Papers that were recommended and automatically added to Zotero by RAGS. Note that they are listed as unfiled items because, by default, RAGS will not choose or create folders for paper storage in Zotero and we leave any reorganization to the user to do manually.

our system is able to recommend papers relevant to the user profile instead of random papers available in the corpus.

IX. Limitations

The main limitation of the current implementation is that it solely relies on scholarly, a Google Scholar API, for accessing any paper data that is not already accessible through Zotero. This is problematic because scholarly is relatively new and new issues are presently being opened in the GitHub repository, which indicates its unreliability. For example, the function that access all papers that some paper is cited by is currently unreliable, which is why its use can be toggled on and off in our corpus-building implementation. Furthermore, if our system issues too many calls to the API, it will no longer be able to access Google Scholar due to a max tries exception that comes from scholarly thinking our system is a bot and consequently blocking it. Using proxies to make calls is not a useful solution to this because proxies are often flooded.

The approach we use to extracting abstracts from publication page URLs or full paper text PDF URLs is also unreliable because it relies on strict assumptions about the way an abstract is positioned in the text (e.g. prompted by the word 'Abstract' and followed by white space) that is scraped from these URLs.

Another limitation of our work is the limited experiments we performed on our system. Due to timing and manpower constraints, we weren't able to design a full evaluation scheme for the system and only relied upon a small number of tests to make sure that the system is working. However, this does not facilitate comparison with other existing systems, and though the experiments reflect upon the quality of recommendations made by the system, it is not enough to convince people that it works better than a generic recommender.

The system also consumes a considerable amount of computational resources due to the size of the vectorization model. One way of resolving this is to implement a client/server infrastructure where all the heavy workloads are performed on a very fast server, so that we can support users without expensive hardware and also accumulate information about all users of the system in a more complex recommendation scheme, instead of relying upon only the data from the local users.

X. Future Work

Our current system relies on the use of Zotero for a user's paper storage and the use of Google Scholar for accessing new papers and their associated data. There are other citation managers, such as Mendeley, Paperpile, and EndNote, which students may prefer and already be using, so it would be helpful to integrate support for these in our system. Similarly, there are many online sources for finding publications apart from Google

Scholar, which sometimes contain publications that are not available on Google Scholar, so somehow incorporating access of other digital libraries could be beneficial.

It is also possible for us to make use of more context information to make using the system more convenient. For example, our system makes different recommendations for different projects, but the user has to manually select the project for which they're requesting recommendations. A potential way to incorporate more information is to detect activities currently performed by the user and infer the project currently being worked on, so that relevant papers can be shown first instead of generic recommendations based on the full user profile.

This paper focuses on a paper recommendation system for graduate students, but the ultimate goal is to build a personal digital assistant that can be helpful in a variety of ways. For example, we imagine another function of interest for graduate students to be providing assistance with scheduling and planning, particularly with respect to organizing conference attendance and research project timelines. Additionally, it can provide tools for note taking, note organization, and other class-related tasks. The paper recommendation system could be extended with the incorporation of author and collaborator information to either improve the paper recommendations or to offer recommendations for collaboration. Combining data from multiple users of our PDA could be another way to facilitate collaboration and potentially improve our recommendation system, most notably by allowing for the use of collaborative filtering.

References

[1] R. Van Meteren and M. Van Someren, "Using content-based filtering for recommendation," in *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*, vol. 30, 2000, pp. 47–56.

[2] J. Beel, B. Gipp, S. Langer, and C. Breitinger, "Paper recommender systems: a literature survey," *International Journal on Digital Libraries*, vol. 17, no. 4, pp. 305–338, 2016.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pre-trained language model for scientific text," *arXiv preprint arXiv:1903.10676*, 2019.

[5] D. Miller, "Leveraging bert for extractive text summarization on lectures," *arXiv preprint arXiv:1906.04165*, 2019.

Appendix A
Corpus Building Example

Below is an example of three initial papers that come from a user's Zotero library and their related papers that are found through Google Scholar and added to the corpus of papers to draw recommendations from.

Initial Papers in Zotero Library
Attack-Resistant Federated Learning with Residual-based Reweighting
Certifiably-Robust Federated Adversarial Learning via Randomized Smoothing
Data Poisoning Attacks Against Federated Learning Systems
Google Scholar-Identified Related Papers Added to the Corpus
Attack-resistant federated learning with residual-based reweighting
Mitigating backdoor attacks in federated learning
Baffle: Backdoor detection via feedback-based federated learning
Can you really backdoor federated learning?
Learning to detect malicious clients for robust federated learning
Robust aggregation for federated learning
Crfl: Certifiably robust federated learning against backdoor attacks
Dba: Distributed backdoor attacks against federated learning
Backdoor attacks and defenses in feature-partitioned collaborative learning
Certifiably-Robust Federated Adversarial Learning via Randomized Smoothing
Federated learning in adversarial settings
Blockchain for federated learning toward secure distributed machine learning systems: a systemic survey
Mitigating data poisoning attacks on a federated learning-edge computing network
A new ensemble adversarial attack powered by long-term gradient memories
Adversarial training in communication constrained federated learning
Provable defense against privacy leakage in federated learning from representation perspective
Fat: Federated adversarial training', 'Data poisoning attacks against federated learning systems
Local Model Poisoning Attacks to Byzantine-Robust Federated Learning
Mitigating backdoor attacks in federated learning
Data poisoning attacks on federated machine learning
Can you really backdoor federated learning?
The limitations of federated learning in sybil settings
Mitigating sybils in federated learning poisoning
Learning to detect malicious clients for robust federated learning
Dba: Distributed backdoor attacks against federated learning