5-2020

Meta-analysis of biological research literature

Evan Suggs
University of Tennessee at Chattanooga, evandsuggs@gmail.com

Meta-Analysis of Biology Research Literature

Evan Drake Suggs

Departmental Honors Thesis

The University of Tennessee at Chattanooga

Department of Computer Science

Examination Date: March 27, 2020

Craig Tanis

Assistant Professor of Computer Science

Thesis Director

Joseph Dumas I.I.

UC Foundation Professor of Computer Science

Department Examiner

Table of Content

Abstract

Comparative studies have been powerful tools in generating a broad understanding about the evolution of animal social systems but they currently rely on the slow, manual process of reading thousands of abstracts and papers from research databases.

A web application was created for researchers conducting a comparative survey, in order to speed up their research. This web application automates the retrieval of research papers and their selection process. Using previously obtained data sets on the orders *Artiodacytla* and *Lagomorph*, a machine learning application was created to classify the papers. These techniques and tools should greatly increase the speed at which researchers are able to compile papers and improve progress towards publication and scientific advancement.

Introduction

The UTC Biosearch project discussed in this paper took shape over the Fall 2019 and Spring 2020 semesters. It has two main components, a web application and a background system including web crawling application programming interfaces (APIs) and machine learning libraries. This project was developed on several machines over the course of several months. The final machines used for this project were the SimCenter's Qbert cluster and a VM running Ubuntu 18.04.3 Long Term Support with the former running machine learning training and the later being a server for the web application.

Meta-analysis and comparative studies of biological research literature are concerned with compiling research in a given domain and analysing them for correlations and trends. They are powerful tools in generating a broad understanding about the evolution of biology topics such as animal social systems; however, they currently rely on the slow, manual process of reading each abstract and paper for a large volume of papers [1-6]. For example, if a researcher is searching for papers on field work with the peccary, a traditional web search might yield as many or more papers focused on lab research.

A researcher must read through all the obtained abstracts in order to see which are suitable or unsuitable, then read through the full papers of the subset labeled suitable. For a biological order and its list of species, this process may be done thousands of times before an actual study can be compiled from a final corpora.

Work using this manual analysis of papers is currently being done in the University of Tennessee at Chattanooga (UTC) Environmental Science department by a team overseen by Dr.Loren Hayes. Over the past several years, they have compiled a dataset on papers in the

mammalian orders *Artiodactyla* and *Lagomorph*. The goal of this project is to be able to match the quality of manual work in the UTC environmental science department with software in a time frame of hours or days rather than years.

The original project proposal had two main objectives:

1. Produce a website where data and publications related to Dr. Hayes' research can be uploaded by the scientific community
2. Develop an automated search method based on modern unstructured text extraction and machine learning techniques

The website will take in search jobs (based on biology terminology and species classification) from a user to compile papers from scientific and academic databases to the project's database with the goal of mimicking the discretion of a domain researcher and provide domain researchers with computer science tools to survey and classify research literature

These papers should match the following suitability criteria:

- Concerned with open field research, not lab or other artificial research.
- Concerned with social and familial groups in the mammalian orders, *Artiodactyl* and *Lagomorph*

In the backend, papers will be gathered from these search jobs and their suitability will be predicted by a machine learning model using semantic vector spaces. Together these techniques should greatly increase data collection productivity and improve progress towards publication and scientific advancement.

Methods

Flask and Server application

Work on the server application began after meeting with Dr. Hayes and his research assistant, Ashley Carpenter. The main goal was to create a proof of concept of goal one: produce a website where data and publications related to Hayes' research can be uploaded by the scientific community.

Various considerations changed the form of the application. The time required for a real time search of databases to retrieve papers proved to be too long for a realistic user (about 5-10 minutes for a search of approximately three hundred papers). This was turned into a background job system discussed in the Database section.

This work was made possible via the facilities of the University of Tennessee's Multidisciplinary Research Building also known as the SimCenter. The SimCCenter is a place for multidisciplinary research that houses UTC's high performance clusters and servers. This project uses the Flask framework and runs on a Ubuntu Linux VM running on the SimCenter's servers. Once the Flask server is run, it can be accessed from any computer running on the UTC internet or VPN from an IP, from the SimCenter on port 8080.

Flask is a 'micro' web development framework designed by the Pallets Project and built with their Jinja and Werkzeug libraries. 'Micro' in this case means that many essential features such as Database development are not included in the base library, rather, they must be downloaded separately [19]. However, their functionality is akin to a dependent library, which

creates a relatively smooth programming experience [19]. Flask is used by Netflix for its servers which integrate with both video encoding and machine learning frameworks [20].

The code base ended up composed of 68.6% Python, 26.1 % HTML, and 5.3 % Bash shell scripting with over a thousand lines of custom code. Shell scripting was utilized for server initialization. If the project requires multiple large users making separate comparative studies to use the program, it might make more sense for each to use an individual VM for their individual works. If communication is required, then a central database or communication system might be created.

A typical example of the user experience of the application proceeds as follows. A user registers for an account with username, password and email. Once logged in, the user can submit a search job [Figure 1]. The application stores these searches in a database, where the user search queries are treated like jobs to be completed by the back-end search program. From there the user can find the page [Figure 2] for previously submitted search jobs. If the selected job has been completely processed, then the user can look through the paper display pages where about 10 results are on a page [Figure 3]. For each page, the user can submit the suitability (select box) or unsuitability (leave box empty) for each paper.

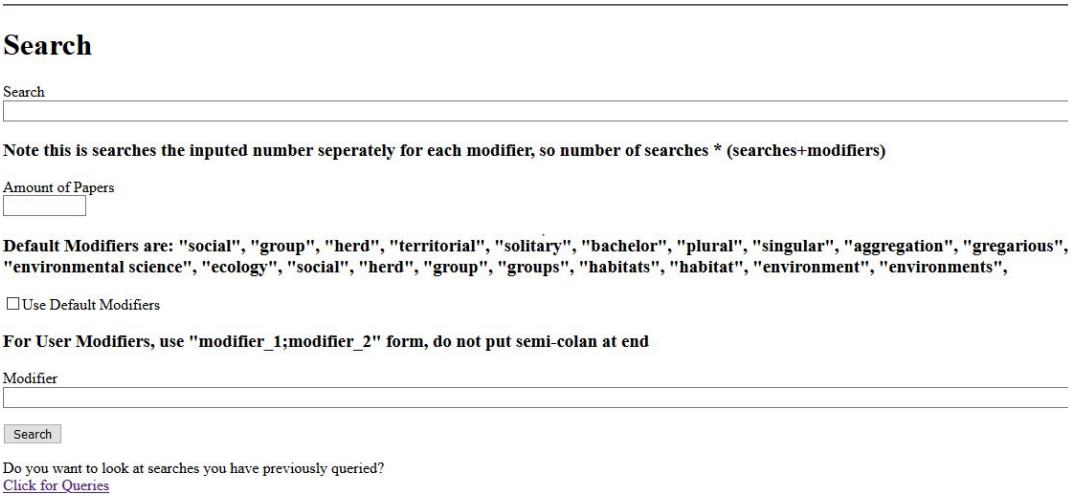

Figure 1. Search submission page

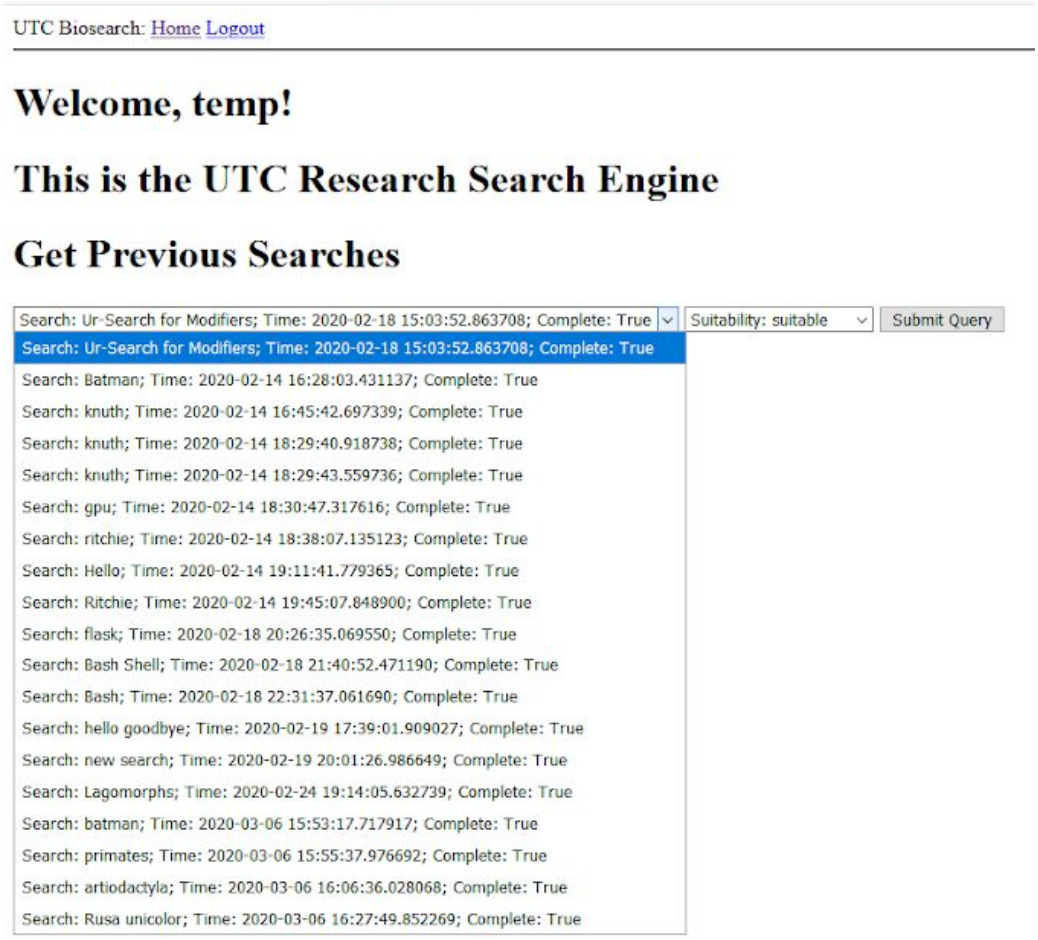

Figure 2. Server query selection page for submitted searches

Figure 3. Papers display page with selected metadata

Search APIs

The earliest work on the project focused on finding useful libraries to find papers and their meta-data in databases focused on scientific and academic papers. Two popular databases of this type are Google Scholar and Web of Science.

Previous work had been composed of a workflow for manual searching. The workflow of Dr. Hayes and his research students can be summarized as follows:

For a list of species in an order,

1. Search with a series of modifiers to collect papers
2. Repeat searches for the categories, behavioral science, zoology and environmental science/ecology
3. Manually mark abstracts/papers that actually report on social organization and do not focus on lab studies and add these to a marked file
4. Search abstracts/papers for terms ['social', 'group', 'herd', 'territorial', 'solitary', 'bachelor', 'plural', 'singular', 'aggregation', 'gregarious']
5. Add the final chosen articles to a complete file

The Hayes project primarily used Web of Science and used Google Scholar and other databases for additional papers. When the project first started, it was assumed that UTC's contracts included access to the full Web of Science API. However, this was not the case, and the project had to continue with just Google Scholar until either a renegotiation or another API is available.

Google Scholar and a useful API, Scholarly were chosen. Conveniently, it seems that each of their publisher databases are roughly identical aside from idiosyncrasies of the respective platforms such as weighted sorting of papers [15-17].

The primary web scraping API was a Python module, Scholarly, available from the standard Python library installer PIP [15]. This package is itself built upon a popular Python web scraping tool, Beautiful Soup [18]. It has the ability to search based on author, keyword or publications. The publication ability is the primary function needed for this project.

Python virtual environments were utilized for all of the work except for the server VM. These were created with Conda, an open source package manager [21]. These virtualize the importing of libraries allowing multiple programs to use conflicting package requirements on the same system.

For example, if library x might need Python 2 and library y might require Python 3, then one can activate their respective virtual environment at runtime for which program needs them. However, while a program can be run in different libraries, only one virtual environment can run at a time.

Due to the unofficial nature of the Scholarly API, it waits for a semi-random amount of time (between 5 and 10 seconds) to avoid overloading the Google Scholar server and getting blocked [15, see source code]. This forms the primary speed bottleneck of the project.

Using the Scholarly API along with its popularity in data science (enabling the later decision to use the machine learning frameworks Gensim and Keras) were the primary reasons for choosing to use the Python programming language for the majority of the code [7, see Python FAQ].

The final form of the Scholarly application exists as a back end program for the server application. This application called scholar_cycle runs a database query for search jobs that have yet to be completed. After it has used the Scholarly API to find papers in Google Scholar, it will then sleep for several seconds and run another query. If the query has newly submitted search jobs, it queries Scholarly for them. Otherwise, it will retrieve no jobs, sleep and repeat this process until a search job is found.

A sister application/prototype called cron_scholar is suitable for time scheduling with a utility such as cron. It only retrieves incomplete search jobs once, then terminates execution. However, if two instances of this application are running at the same time, they will complete the same jobs and cause conflicts in the database.

Database

During early research, it became obvious that storing the data only in text files would be unwieldy and time-consuming. The obvious next step was a database using some sort of Structured Query Language (SQL). Early prototypes were created with the SQLite3 library built into Python to add on to experiments with Scholarly. These functioned primarily to mechanically mimic the Hayes workflow. This workflow placed suitable and unsuitable papers into individual database files marked by their species. These experiments mostly used a simple paper schema (a plan for how data is ordered in SQL) including abstracts, author information, etc. for all files, but later it expanded to include more metadata. The original purely SQLite3 plan had many drawbacks such as having no way of automatically converting the data into useful data and repetition of query code [7, SQLite3].

The current iteration under the Flask web server utilizes a SQLAlchemy wrapper communicating with the Flask API. This creates schema models from Python class structures, allowing very simple use and conversions to other data structures such as dictionaries (especially useful for web pages). There is also a wrapper, called migrate, for migrating data between schemas and formats.

The basic relationships for the database are shown in [Figure 4]. The first table is a simple User account table for the server. This has a one to many relationship with the searches table. The TimeStamp field takes in a date time object and as it is obfuscated from the user, it should always reflect the time when entered into the database. This is based on and compatible with the 1989 C time standard [7, see datetime]. The Complete field tracks whether or not the

search has been processed by the background job system and the number_of_searches field gives a flexible paper limit for the Scholarly API to stop searching.

The Modifier Table is based on the modifiers from the Hayes workflow. When it is accessed, each search can have many modifiers and the backend program includes those in its queries. This takes the form of a boolean search. For a given search, for all modifiers, it conducts a boolean search of the form, “search+’ AND ‘+modifier.”

The Paper Table is mostly composed of fields for paper data obtained from the Scholarly search API. These include title, authors, and abstracts (note that not all fields need to be filled, as sometimes the databases do not include certain fields). There is also a back reference to the original search and a field to store the suitability of the abstract. The suitability field is a string rather than a boolean in order to flexibly introduce additional modes of suitability later. Once stored, the server application can find papers from a selected search [Figure 2] and then display the papers from said search [Figure 3].

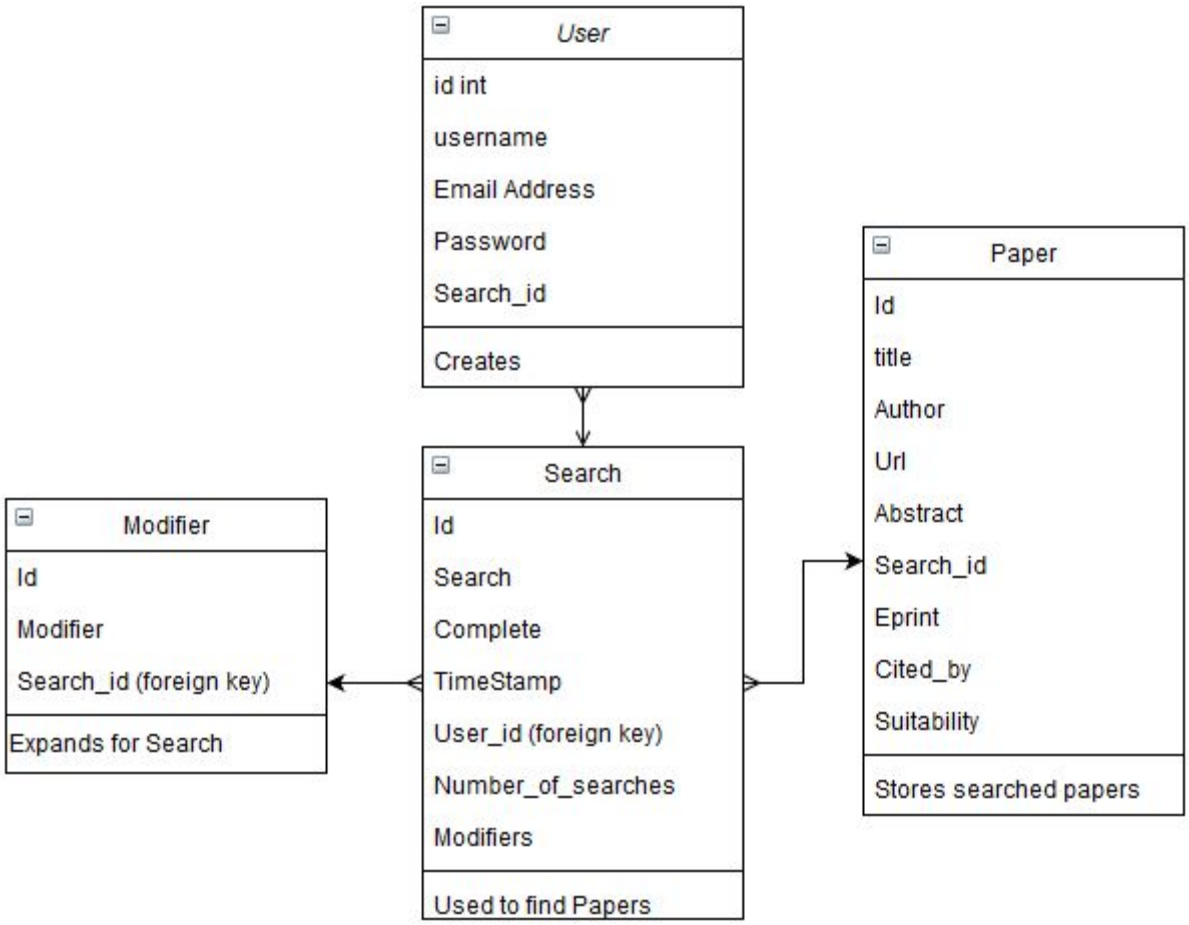

Figure 4. Simple Model of Database

Brief History of Natural Language Processing

Traditional machine learning techniques and natural language processing have a long history reaching back to the beginning of computer science in the 1940s [11, see chapter 1.2]. However, much of this work was theoretical and dedicated to machine translation and knowledge representations [13, p. 4]. While much of this came at the expense of neural networks theory, that field was already at a disadvantage due to the performance of contemporary systems and the expense of storing the large amounts of data necessary to train a suitable system [11].

Natural Language Processing also has a long history of versatile, simple and useful tools, such as GREP and regular expressions. GREP is used by many programmers every day, but it was created to analyze the text of the *Federalist Papers* [12]. Unlike many of the older systems of classical machine learning, these tools are still in use today [11-13].

With the rise of the Internet and the availability of data such as Google Scholar, the corpora and datasets necessary to create useful models are available to all researchers [11]. This project is an attempt at creating useful recyclable tools for research interests such as GREP using modern machine learning techniques while possibly creating something long-lasting.

Dataset

This portion of the project is based on objective 2: develop an automated search method based on modern unstructured text extraction and machine learning techniques. This is the portion of the project that is in the least state of sophistication. Ideally, the final portion should be able to take an abstract from an academic database as input, then guess how suitable it is for the project based on the Hayes dataset. Most of this work was created with the Gensim and Natural Language Toolkit libraries for Python [14, 25].

As discussed in the introduction, the Loren Hayes dataset is composed of the metadata for approximately 20,000 papers compiled for their comparative studies and meta-analysis of the mammalian orders, *Artiodactyl* and *Lagomorph* [Figure 5]. The *Artiodactyl* dataset has 238 suitable papers and 15,967 unsuitable papers, and the *Lagomorph* has 10 suitable papers and 4,448 unsuitable papers.

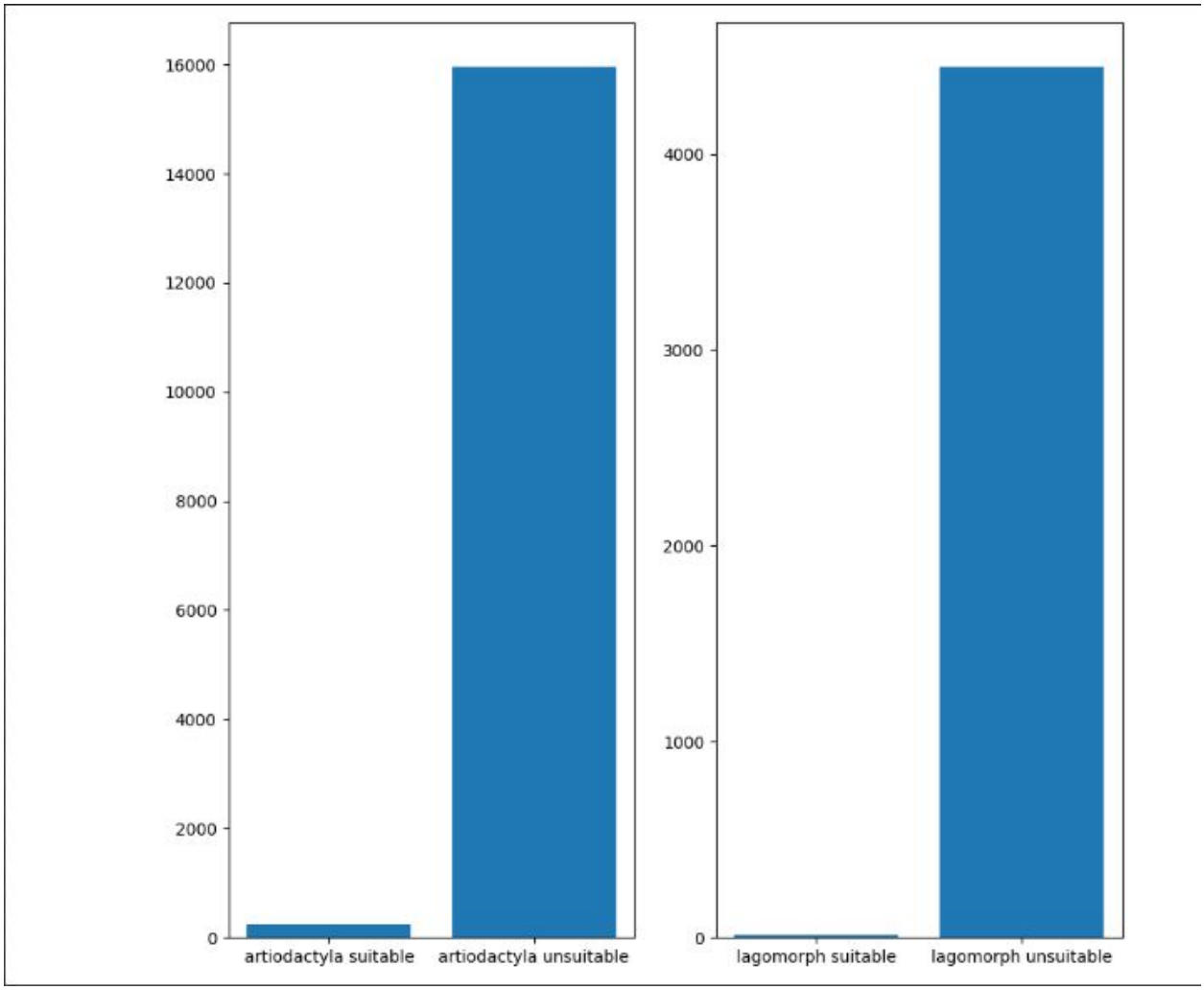

Figure 5. *Artiodactyl* with 238 suitable papers and 15,967 unsuitable papers and *Lagomorph* with 10 suitable papers and 4,448 unsuitable papers

The dataset is large for human compilation, but not as big as other datasets for training such as GloVe from Stanford which contains 400K words [20]. In addition, the dataset only has +approximately 250 suitable papers versus 20,000 unsuitable papers which can lead to a classifier biased towards identifying unsuitable papers. However, the goal of the project is less to accurately highlight every suitable paper, and instead, cut the amount of papers necessary for humans to read, meaning that false suitable papers represent an outsized waste of researcher time and resources.

Document Vectors

Document Vectors, and the related Word Vectors, are useful and relatively straightforward as training tools. Whereas a Recurrent Neural Network processes words and documents as vectors in a hidden and complicated way, vector representations are relatively straightforward for anyone with a knowledge of linear algebra [11]. They represent words or documents (as a group of words) as a vector and store them in a vector space. For example, “hello, world” and “goodbye, world” in English have a relationship analogous to the French phrases “bienvenue le monde” and “Au revoir le monde” on a model trained for machine translation [23-24].

Thesaurus and GloVe

Early tests were to create a thesaurus learning model with the goal of searching the abstracts obtained from Google Scholar for the most similar words based on searches and their modifiers, which could help give weight to more useful papers.

Many predefined models and datasets are included in Gensims repositories, the one found most useful for this project was the Stanford GloVe Word Vectors [14, 22]. More a family of models than one specific model, this project focused on the GloVe model trained using 300 Gigaword of English Wikipedia articles [22].

Using the most similar words function of the Gensim document vector model, given the word ‘social’ the GloVe model gave the related words: welfare, education, political, cultural, health, while the Hayes model gave the related words: epizootical, health, modulation, pre-eradication, placental. The Hayes dataset is focused entirely on scientific vocabulary, but these results are too specific and do not match the actual domain of research. The GloVe results are overly broad. Despite these results, this experiment was useful for introducing the GloVe model to the project.

Tests and Logistic Regression

Gensim was chosen as the library to use for programming the Document Vector models [14]. Since a vector model composes all entered items into a plane, there should be a global divider (a flat plane or another shape) dividing suitable and unsuitable. The simplest goal for testing was to find a global plane in the vector space to divide suitable and non-suitable papers [11, p. 137]. Logistic regression is designed to compute the line between binary classes in a plane so logistic regression was chosen for classification and processed with Scikit-Learn [11, 27].

A main model made with the majority of Dr. Hayes’ dataset, minus 100 from both the suitable and unsuitable datasets for testing. Each document was processed as a vector of 300

floats to maximize information. The original training trained over the dataset for 30 runs through the dataset or iterations of training. The iterations were raised to 50 in order to monitor for overfitting or when a machine learning model has begun to fit the dataset exactly and reduces the generality of the model [11].

For the scores shown in Figure 6, the average accuracy was ~0.58 for standard accuracy and for F1 accuracy score ~0.52 for testing based on the 200 suitable and unsuitable entries. The F1 score is a score computed between the subsets of the data that are false positives and those that are false negatives [11, p. 411]. After 100 iterations we see overfitting on the dataset and a decrease in accuracy in both performance metrics, with the F1 score decreasing even more. This indicates that the model is increasingly detecting false positives and/or negatives.

	Standard Accuracy	F1 Accuracy
30 iterations	0.58	0.5414346544382574
80 iterations	0.62	0.57664884
130 iterations	0.585	0.545442098633588
180 iterations	0.55	0.4357366771159875

Figure 6. Training solely on Hayes dataset

Another run of experiments run on merging the Hayes dataset model with Stanford's GloVe pre-built model based on 300 gigawords of Wikipedia articles. A Wikipedia-based corpus was chosen on the hypothesis that it resembled academic literature more than Twitter or news-based corpora included in GloVe.

	Standard Accuracy	F1 Accuracy
10,000 Wikipedia articles, then 5,000 trained for 30 iterations each	0.605	0.53197
5,000 for 30 additional iterations	0.595	0.5319766580763647
GloVe-Hayes merged model	0.51	0.3551783129359126

Figure 7. Accuracy of models trained with Wikipedia-based models

First, a model trained on the Hayes dataset for 30 iterations was given additional training using the wiki-english-20171001 dataset included with Gensim [14]. As shown in Figure 7, this model had slightly better standard performance than just the Hayes dataset model but worse F1 accuracy, meaning that the rate of false negatives and positives is relatively higher. Additional training caused overfitting, however not as substantially as in the pure Hayes dataset model.

Finally a model was made by merging the vocabulary and weights of Gensim's

implementation and the model trained on the Hayes dataset. The merger caused issues with continued training and had the worst accuracy of any model. It is possible that this is due to the added vocabulary enlarging the total vector space making Hayes dataset harder to test with logistic regression.

Results and Further Goals

The logistic regression testing confirmed that the dataset and the Document Vectors alone were not enough to make reasonable results for the project. However Document Vectors might be a good way of processing and managing the dataset for more later models. Especially as Gensim has compatibility with other machine learning frameworks.

Early research began on the topic of capsule networks for establishing relationships around the meanings of papers. This was removed for being difficult to get effective results in a reasonable timeframe. As the capsule network is designed to mimic the correlations in the human mind, it might be useful to mimic what researchers find more suitable than Document Vectors [26].

Conclusions

Discussion

The main bottleneck of the project is the Scholarly search API. It must wait five to ten seconds for each search item in order to avoid resembling a malicious bot. The lack of real-time performance is not preferable, but may be considered tolerable for a research user. Parallelizing the application runs an even greater risk of being denied service by Google Scholar, but it may

be possible to run multiple searches across different academic databases, *i.e.* running the same search on both Google Scholar and Web of Science, when and if that is integrated.

Due to the file based nature of SQLite, if the database is to scale up in size, it is necessary to migrate to another type of database. Likely options include MariaDB/other miscellaneous SQL types or a NoSQL option such as MongoDB or Redis. For the former, it would seem to only require some migration work, but this could result in a reduction in storage size and substantial speed boost. The latter would require more work and possibly discarding substantial portions of the codebase, but along with the benefits of the former it would remove dealing with SQL and result in user-readable data in a format such as JSON.

The database has a problem where duplicate data may result from searches. If a single paper shows up on two searches, then it will get placed as two distinct entries in the database. This is useful as it obfuscates different users and searches from each other. One of the ideas for final deployment, if the project reaches deployment for multiple projects, is to restrict access to one server VM per project, obfuscating projects that way. If papers were collated to a central database in addition to individual VM databases, that might allow redundancies to be culled at set times for all users.

While useful and vital to the project, the dataset proved to be difficult to work with in some respects. In previous work, It was discovered that a dataset is more effective if the data partitions are roughly the same size (*i.e.* within an order of magnitude) for the same label. If projects other than Hayes and his team are introduced, it might be useful to include their dataset in training. However, this leads to issues if their projects are dissimilar to the mammalian orders.

Further Work

In addition to work from discussion, further work is planned as the project progresses and becomes a project for graduate school.

Work on integrating the Web of Science and other academic databases should begin soon after this semester. The project has access to a restricted version of the Clarivate Web of Science API but this does not include abstract information, which must be obtained from other places. Combining this and obtaining the abstracts from another program/API such as Scholarly form an interesting starting point.

Final Thoughts

This project will continue over the summer and next year as a Graduate Project for Mr. Suggs. The web application and server portion of the project is the closest to being suitable for research. Automated searching is functional and useful for researchers. Additions and alpha testing with Dr. Hayes and his research student, Ashley Carpenter should begin in April 2020 and continue through summer.

However, the process for classifying retrieved papers requires additional work. This project has the ability to increase scientific productivity by decreasing the amount of time for domain researchers to conduct their work by using computer science tools. Furthermore, the machine learning portion requires a substantial difference in approach and possibly additional data to achieve satisfactory results.

Overall, the results were satisfactory and promising for an undergraduate thesis. It constitutes the beginning of a much larger project and an enriching cooperation between the UTC computer science and environmental science departments.

Appendices

Appendix A: Acknowledgements

Special Thanks to Craig Tanis, Loren Hayes, Ashley Carpenter, Thomas Wiegand, Hannah Margavio, Aaron Crawford, Mike Ward, Chris Dowell, Joseph Dumas and Andrew Nguyen.

This project would also like to thank the Center of Excellence in Applied Computational Science and Engineering (CEACSE) for providing a grant for this project along with the UTC Department of Biology, Geology and Environmental Science for their cooperation.

Bibliography

[1] Schradin C. Comparative studies need to rely both on sound natural history data and on excellent statistical analysis. Royal Society open science. 2017;4(9):170346.

[2] Chak STC, Duffy JE, Hultgren KM, Rubenstein DR. Evolutionary transitions towards eusociality in snapping shrimps. Nature ecology & evolution. 2017;1(4):0096.

[3] Gonzalez J-CT, Sheldon BC, Tobias JA. Environmental stability and the evolution of cooperative breeding in hornbills. Proceedings of the Royal Society B: Biological Sciences. 2013;280(1768):20131297.

[4] Schradin C, Hayes LD, Pillay N, Bertelsmeier C. The evolution of intraspecific variation in social organization. Ethology. 2018;124(8):527-36.

[5] Aureli F, Schaffner CM, Boesch C, Bearder SK, Call J, Chapman CA, et al. Fission-fusion dynamics: new research frameworks. Current Anthropology. 2008;49(4):627-54.

[6] Langbein J, Thirgood SJ. *Variation in mating systems of fallow deer (Dama dama) in relation to ecology*. Ethology. 1989;83(3):195-214.

[7] Python Software Foundation. Python Language Documentation, version 3.8.2. Available at https://docs.python.org/3/.

[8] Calvillo, E.A. & Padilla, Alejandro & Muñoz, Jaime & Ponce, Julio & Fernandez-Breis, Jesualdo. (2013). *Searching research papers using clustering and text mining*. 78-81. 10. 1109/CONIELECOMP.2013.6525763. Available at https://ieeexplore.ieee.org/abstract/document/6525763

[9] Flemming, K. and Briggs, M. (2007), *Electronic searching to locate qualitative research: evaluation of three strategies*. Journal of Advanced Nursing, 57: 95-100. Available at

https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2648.2006.04083.x doi: 10.1111/j.1365-2648.2006.04083.x

[10] Bird, S. and Klein, E. and Loper, E. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc, 2009.

[11] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[12] Kernighan, Brian and Sean Riley. "Where GREP Came From-Computerphile," University of Nottingham, July 6, 2001, 10:06, https://www.youtube.com/watch?v=NTfOnGZUZDk

[13] Sparck Jones, Karen. *Natural Language Processing: A Historical Review*. University of Cambridge, https://www.cl.cam.ac.uk/archive/ksj21/histdw4.pdf.

[14] ŘEHŮŘEK, Radim and Petr SOJKA. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks*. Valletta, Malta: University of Malta, 2010. p. 46--50, 5 pp. ISBN 2-9517408-6-7.

[15] Cholewiak, Steven A. "Scholarly." Available at https://pypi.org/project/scholarly/. Accessed on March 12, 2020.

[16] Google Software. "About Google Scholar" Available at https://scholar.google.com/intl/en/scholar/about.html. Accessed on March 15, 2020.

[17] Web of Science Group. "Web of Science platform: Web of Science: Summary of Coverage." Clarivate Analytics, March 13, 2020. Available at https://clarivate.libguides.com/webofscienceplatform/coverage. Accessed on March, 15, 2020.

[18] Richardson, Leonard. “Beautiful Soup: We called him Tortoise because he taught us.” Crummy, 2020. Available at https://www.crummy.com/software/BeautifulSoup/. Accessed on March 10, 2020.

[19] Pallets Project. “Flask: Web Development, One Drop at a Time.” Pallets, 2010. Available at https://flask.palletsprojects.com/en/1.1.x/. Accessed on March 5, 2020.

[20] Netflix Technology Blog. “Python at Netflix.” Medium, April 29, 2019. Available at https://netflixtechblog.com/python-at-netflix-bba45dae649e. Accessed on March 17, 2020.

[21] *Anaconda Software Distribution*. Computer software. Vers. 2-2.4.0. Anaconda, Nov. 2016. Available at https://anaconda.com.

[22] Manning, Christopher D., Jeffrey Pennington and Richard Socher. “GloVe: Global Vectors for Word Representation.” Stanford, 2014. EMNLP. 14. 1532-1543. 10.3115/v1/D14-1162. Available at https://nlp.stanford.edu/pubs/glove.pdf. Accessed on March 15, 2020.

[23] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." In *International conference on machine learning*, pp. 1188-1196. 2014.

[24] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In *Advances in neural information processing systems*, pp. 3111-3119. 2013.

[25] “Natural Language Toolkit.” NLTK Project, March 08, 2020. Available at https://www.nltk.org/index.html. Accessed on March 10, 2020.

[26] Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." In *Advances in neural information processing systems*, pp. 3856-3866. 2017.

[27] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. "Scikit-learn: Machine Learning in Python. In *Journal of Machine Learning Research*, vol. 11. pp. 2825--2830. 2011.