Mycology Literature Search

IST 664 December 2024

La Monte Yarroll

Christopher Murphy

Jennifer Balasi

Shintaro Osuga

Project Overview:

The project created a search experience for mycology articles using NLP techniques involving a sentence transformer and text embedding. To enable vector-based searching of scientific articles focused on mycology, the project developed and deployed updates to two pre-existing applications

1. The first application, named the Synoptic Key of Life ("SKOL"), is responsible for creating annotated representations of mycology research articles using classification algorithms. This application prepares the article content for embedding in the second application.
    - SKOL was previously developed by project team member, La Monte Yarroll.

2. The second application, named "MycoSearch" is responsible for embedding end-user search queries, and performing a vector search against the embedded mycology research articles.
    - MycoSearch is an extension of a pre-existing search application developed by Dr. Nick Gisolfi, a researcher at the Auton Laboratory at Carnegie Mellon University. The codebase was forked to allow for development and testing of extensions required to embed the mycology content.

The project successfully extended these applications to prepare, annotate, simplify, and embedded SKOL article content into MycoSearch via transformer architecture. End user queries to Mycosearch are converted to vectors, similar embedded article content is located and presented to the end user. The end solution provides a highly accurate search matching capability on a domain-specific corpus of mycology article content. The intent is for a user's query to be a formal description of a particular specimen, though partial descriptions seem to work well.

<u>About the Data:</u>

For our final project, we worked with a comprehensive dataset of over 7,000 scientific articles focused on mycelia. This dataset, gathered through extensive collaboration between the Imperial Institute of Agricultural Research, the Mycological Society of America, and other respected institutions, includes detailed classifications of fungal species.

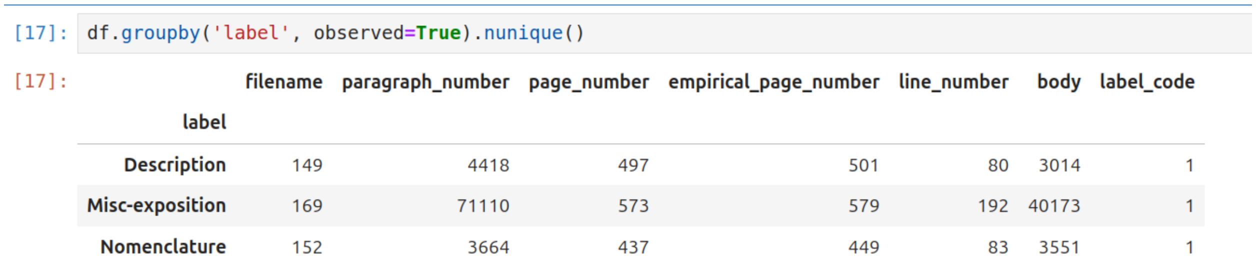
Each article contains precise nomenclature and descriptions in both Latin and English, following a conventional notation system. This consistency allows us to efficiently tokenize the descriptive content and link it with the formal species names.

The following section presents a representative sample of the text format we are working with, illustrating the conventional structure of each research article, which supports our data processing objectives.

Data Preprocessing:

- **Objective and Workflow**:
  - Extract structured information from annotated text files.
  - Standardize labels and prepare data for downstream analysis or visualization.
  - Inspect data at various stages for quality checks and insights.
  - Produce a well-organized pandas DataFrame with categorized labels ready for further processing and modeling.
- **Setup and Initialization**:
  - Import necessary libraries: `glob`, `pandas`, `matplotlib`, and custom modules for file operations and labeling logic.
  - Configure constants: set random seed for reproducibility, define default label (`Misc-exposition`), and specify labels to retain (`Description`, `Nomenclature`).
  - Collect annotated files using `glob` from a directory (`../data/annotated/journals/Mycotaxon/Vol*/*.ann`).
  - Perform initial analysis, noting 169 annotated files in the dataset.
- **Relabeling and Filtering**:
  - Relabel paragraphs using the `target_classes` function:
    - Assign default label (`Misc-exposition`) to paragraphs without matching specified labels.
    - Retain original labels for paragraphs matching the specified labels (`Description`, `Nomenclature`).
  - Filter out irrelevant or noisy data, ensuring consistency in labels.
- **Parsing and Structuring Data (new code)**:
  - Read and parse text files into paragraphs using custom functions.
  - Map annotations (labels, metadata) to corresponding paragraphs.
  - Convert unstructured text data into a structured intermediate representation for further processing.
  - Store parsed data in a list for iteration and/or manipulation.
- **Sanity Check and Review**:
  - Summarize the relabeled dataset:
    - Processed 79,192 paragraphs, with the first 5 paragraphs inspected for assigned labels.
  - Examine specific paragraphs (e.g., 2000th paragraph), including metadata like filename, paragraph number, label, empirical page number, and text content.
- **DataFrame Creation and Statistical Summary**:
  - Convert relabeled data into a pandas DataFrame with structured columns:

        - Filenames, labels, paragraph numbers, page numbers, empirical page numbers, line numbers, text body, and numeric label codes.
    - Categorize and encode labels numerically.
    - Provide a statistical summary of numeric columns (e.g., counts, means, standard deviations, ranges) using `DataFrame.describe()`.
- **Outcome**:
    - The final processed dataset is structured, filtered, and standardized, making it ready for detailed analysis, visualization, or machine learning tasks.



```
[17]: df.groupby('label', observed=True).nunique()
```

[17]:

label	filename	paragraph_number	page_number	empirical_page_number	line_number	body	label_code
Description	149	4418	497	501	80	3014	1
Misc-exposition	169	71110	573	579	192	40173	1
Nomenclature	152	3664	437	449	83	3551	1

Figure A. This illustration displays the counts of unique elements in the dataset. E.g. There are 169 unique filenames of documents with Misc-exposition labels, but only 152 of them contain at least 1 Nomenclature label, and 149 contain descriptions.

Feature Engineering:

La Monte produced the annotated text files about 5 years ago using a text classifier. He hand-annotated a few articles with YEDDA, and then trained a suite of classifiers, picking the best performing classifier. He then ran the classifier on a larger body of articles and hand-corrected the annotations. He repeated this process several times, to produce the set of about 170 journal issues in the current annotated dataset.

The notebook, `skol/jupyter/skol_journals.ipynb`, conducts a comprehensive evaluation of various machine learning classifiers, including BernoulliNB, AdaBoostClassifier, RandomForestClassifier, SGDClassifier, RidgeClassifier, and OneVsRestClassifier. To prepare textual data for model training and analysis, it employs advanced vectorization techniques such as CountVectorizer and TfidfVectorizer, ensuring effective feature extraction and representation.

The new feature engineering for this project includes the construction of the Taxon class, described in more detail below.

Custom Content Object for Embedding: the "Taxon" Class

- Structure:
    - The Taxon classes encapsulate key components of a mycology article, such as:
        - Nomenclature: The scientific naming conventions (e.g. fully notated species names).
        - Descriptions: Detailed textual information about the species. These can be in Latin and English, or just in English.
        - Metadata: Supplementary details such as filename, paragraph number, page number, empirical page number, and line number.
- Purpose:
    - To create a unified object that links structured metadata with descriptive and nomenclatural content.
    - To standardize the data format, ensuring compatibility with embedding processes and vector-based search mechanisms.
    - Provide the information needed to format the embedded content, and provide information for the output step.
- Content Preparation:
    - The Taxon classes combine nomenclature and description text for each species into cohesive objects.

    - It ensures that the relevant content is grouped within a specific proximity (e.g., up to 5–6 paragraphs from the species name).
    - It assumes that nomenclature always precedes the descriptions, which seems to be a good assumption.
- Embedding Process:
    - The Taxon class merges the Description paragraphs for the given Taxon into a single string of text for use in embedding.
    - Text from each Taxon instance is passed to a pre-trained SentenceTransformer model from SBERT (all-mpnet-base-v2).
    - This model encodes the text into 768-dimensional vectors, capturing semantic and contextual meaning.
    - The multi-headed attention mechanism within the transformer enhances the representation by embedding nuances of the text's meaning, context, and position.
- Search Optimization:
    - By encapsulating and embedding content in this structured format, the Taxon classes ensure that searches are highly accurate and contextually relevant.
    - The structured embedding allows cosine similarity measures to rank and return results based on their relevance to user queries.
- Key Benefits
    - The Taxon class simplifies the embedding process by abstracting the complexity of raw article content into a structured, machine-readable format.
    - It enhances the search system's efficiency and accuracy by ensuring all relevant text and metadata are encoded consistently and meaningfully.

Visualizations

To pick the maximum distance between a Nomenclature paragraph and a matching set of Description paragraphs, we visualize the gaps between distinct kinds of paragraph. See Figures B and C for this characterization.

```
[39]: df2.loc[:,'diffs'] = diffs
```

```
[40]: df2.plot(kind='scatter', x='paragraph_number', y='diffs')
      x = 290
      plt.xlim(x,x+1000)
      plt.ylim(0, 100)
      plt.show()
```

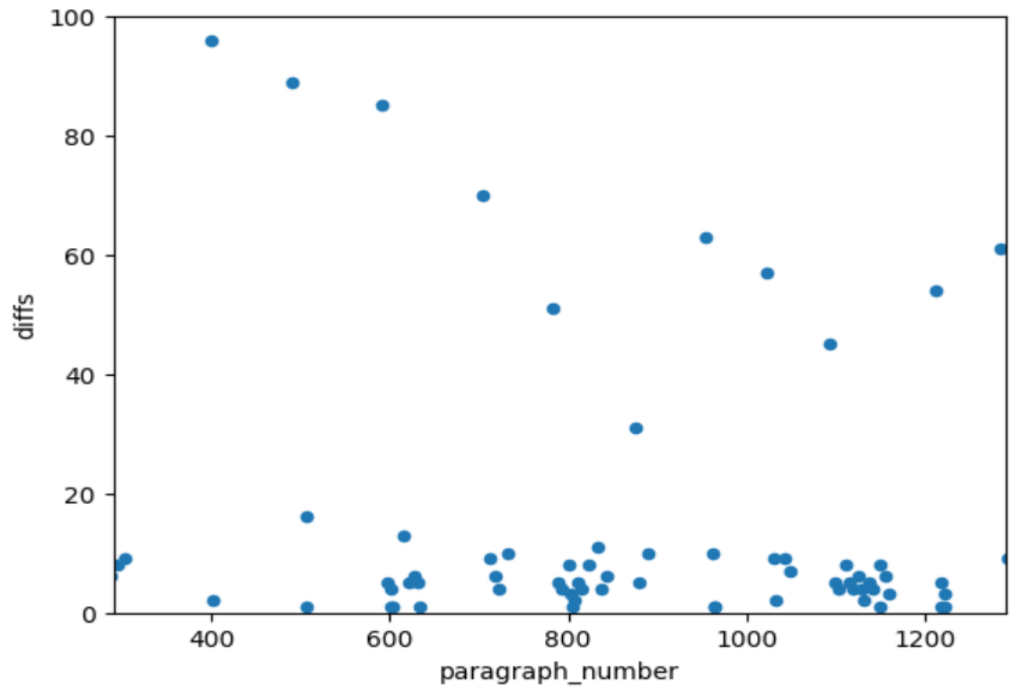


Figure B. *The upper dots represent the distances between species descriptions within an article, indicating how far the search progresses between potential matches. The lower dots highlight species descriptions closely associated with species names. Larger distances between descriptions suggest a lower likelihood of relevance to the search criteria. Typically, the search extends up to 5–6 paragraphs before stopping, with increasing distances between descriptions signaling diminishing relevance to the query.*

```
[43]: diffs = df2['paragraph_number'].diff()
      diffs[diffs < 20].hist(bins=40)
      diffs.max()
```

```
[43]: np.float64(6313.0)
```

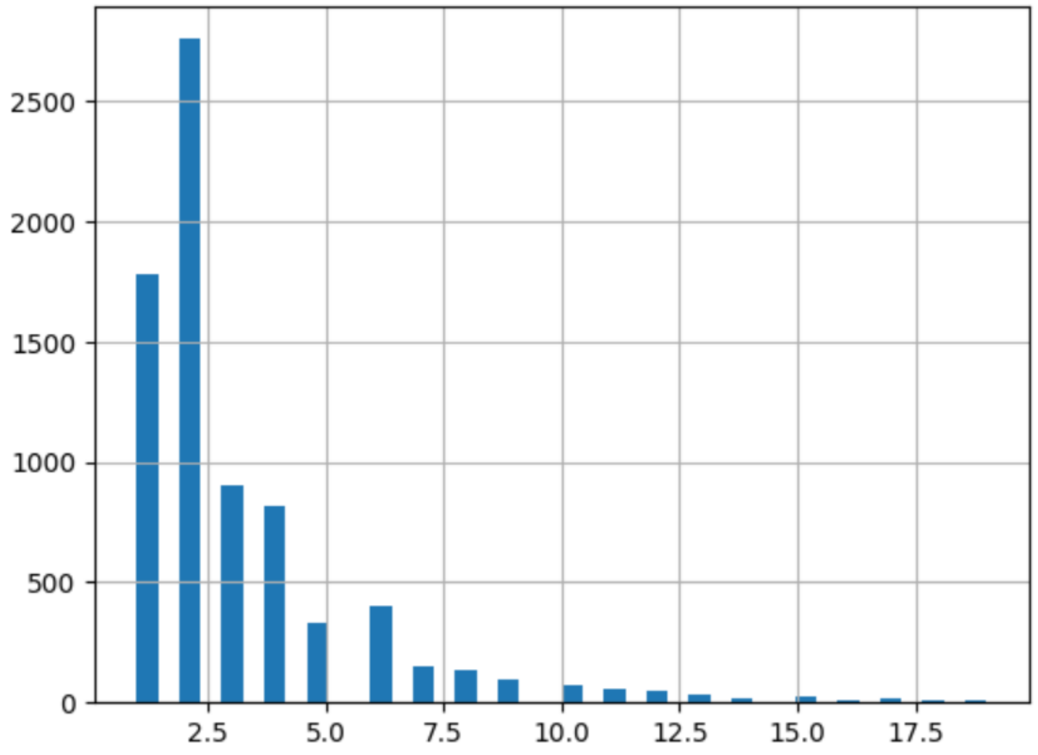


Figure C. *The distribution diagram above shows the count of articles (y-axis) and the number of paragraphs that exist between the species name and the species description (x-axis) in the physical layout of an article document. This information was used to validate the design of the taxon object used to format a simplified version of the article content for embedding.*

SBERT Text Embedding:

The SBERT pretrained SentenceTransformer creates a text embedding of our Taxon object. The SBERT model is based on Transformer Architecture and the Attention Mechanism.

The multi-headed attention embeds the text into query, key, and value vectors which each help represent the data and explain the meaning, context, and position. This information helps encode the text and all its information into an embedding vector which encapsulates the non-apparent text information.

The query search from the user is also encoded in the same way utilizing the sentence transformer.

New Work

The authors extended the SKOL and Dr. Drafts (MycoSearch) projects. We extended the Taxon class (`skol/taxon.py`) for use in embedding. We wrote a script, `dr-drafts-mycosearch/src/get_skol.sh`, which imports the data from SKOL into MycoSearch. We added the SKOL class to `dr-drafts-mycosearch/src/data.py`, which prepares the Taxon objects for embedding. We added the similarity metric to `dr-drafts-mycosearch/src/sota_search.py`.

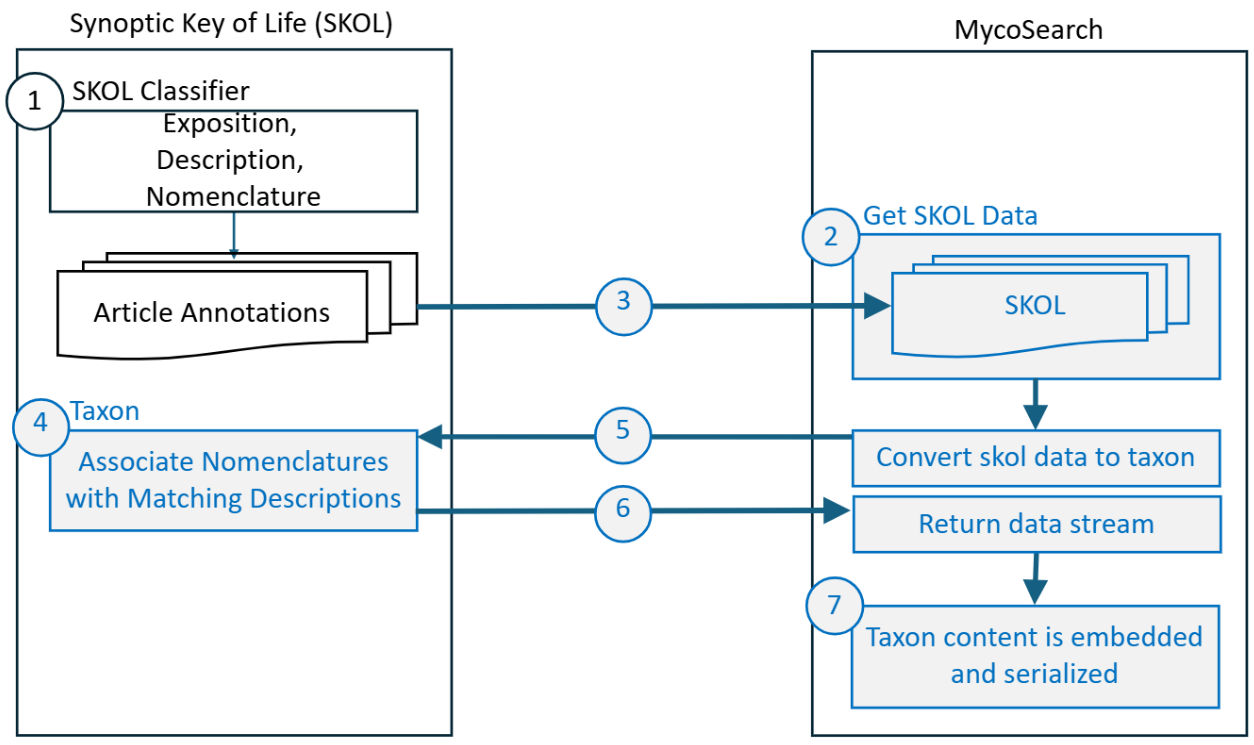

Synoptic Key of Life (SKOL)
MycoSearch
1
SKOL Classifier
Exposition, Description, Nomenclature
Article Annotations
2
Get SKOL Data
SKOL
3
4
Taxon
Associate Nomenclatures with Matching Descriptions
5
Convert skol data to taxon
6
Return data stream
7
Taxon content is embedded and serialized

Figure D. *The flow diagram above depicts the process of transferring the annotated mycology articles from SKOL to MycoSearch, converting them to Taxon format, and* *embedding the content into MycoSearch. The light blue items (2-7) are new or modified code.*

Literature Matching:

The embedding for the query vector is created using the same embedding transformer as the article content. The embedding process employs mathematical representations of data as vectors in a 768-dimensional space. This process ensures that the vectors for the query can be compared to the vectors for article content and ensures that similar text representations are embedded in proximity in the higher dimensional spaces of the model.

The query vector is used in a cosine similarity function to identify the most similar mycology article embedding. Cosine is used to measure the angle between these vectors to determine the similarity in their direction. A cosine similarity measure of 1 indicates word vectors are identical, a measure of 0 means the vectors have no similarity, and a measure of -1 indicates opposites. This search matches a query vector and an article vector when the cosine similarity measurement is closest to 1.

This vector-based search method using sentence transformer encoded text enables highly accurate retrieval of information based on semantic or contextual relevance due to how the embedding maps similar content to the higher dimensional space representation of the text.

Vector-based search performs better than relying solely on keyword matching or just simply embedding the text and searching the embedded space. This approach is particularly effective in processing complex or unstructured datasets, ensuring precise encoding and identification of the taxon object.

This similarity value is then calculated for all nomenclatures, and they are ranked and the top N highest similarity score text is returned to the user. The strength of correlation between the query and the returned text is given but also represented by color with magenta having the highest similarity and light grey having the lowest similarity score. Along with the text and similarity score, the journal which it came from along with the URL where it can be found is returned to the user.

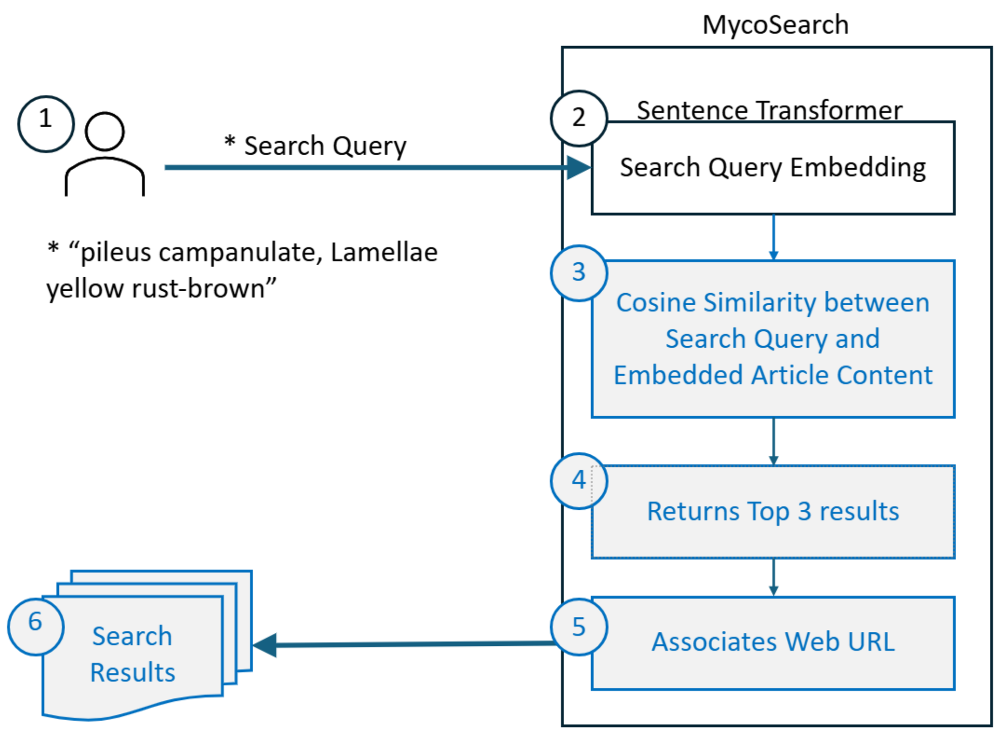


Figure E*. The flow diagram above depicts how the MycoSearch application performs vector-based searching for mycology article content.*

Results:

The literature search's results are presented in a visually engaging and intuitive manner using ANSI escape codes for stylized, color-coded console outputs, emphasizing descriptions, rankings, and key insights. Results can be displayed directly in the console or exported to a CSV file for further analysis. A dynamic color-coded ranking system visually represents the relevance of results based on cosine similarity, making it easier to understand performance metrics at a glance.

The notebook, `skol/jupyter/skol_journals.ipynb`, sorts and ranks results to identify the best-performing classifiers for various tasks and labels, offering valuable insights into the effectiveness of different vectorization and classifier combinations for specific use cases. To further refine the analysis, the dataset is divided into subsets categorized by labels, enabling detailed performance evaluations for each category.

Moreover, the Experiment class simplifies the process of conducting new searches by allowing for easy instantiation and execution with customized prompts and datasets. This comprehensive and flexible framework ensures users receive tailored and actionable insights that align with their specific needs.

Participation Breakdown:

The development process was a highly collaborative effort facilitated through a combination of real-time and asynchronous tools. Our contributions to the existing code project were made during quartet programming sessions conducted via video calls, with the code seamlessly integrated into the codebase through GitHub. The accompanying PowerPoint presentation, which articulates key aspects of the project, was collectively created by the team during a Zoom meeting. Additionally, this final project document, which provides a comprehensive overview of the project, was a group effort. This document covers the project statement, details about the data, data preprocessing techniques, text embedding methodologies, feature engineering processes, and the approach to literature matching, ensuring a well-rounded and detailed explanation of the work completed.

Because we worked simultaneously via video calls throughout the process, individual contributions were interwoven, making it difficult to attribute specific tasks to any one team member. This collaborative approach reflects the team's collective effort and shared responsibility for all aspects of the project.

Future Development:

As we have grown attached to this project, we will be continuing to work on the interface. We will integrate the completed project with the existing Python-based web application hosted in Azure to ensure seamless functionality and optimized performance within the cloud environment allowing users the ability to search easily.

Other improvements include:

- Adding a language model which understands Latin.
- Incorporating more journal articles.

Links

The original Dr Draft's SOTA Literature Search can be found on github.

Our fork can also be found on github at https://github.com/piggyatbaqaqi/dr-drafts-mycosearch.

The SKOL Project is also on github.