Graph Embeddings for Harvard Widener Library Data

A THESIS PRESENTED
BY
XAVIER M. EVANS
TO
THE DEPARTMENT OF COMPUTER SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
BACHELOR OF ARTS
IN THE SUBJECTS OF
COMPUTER SCIENCE AND LINGUISTICS

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS
MAY 2023

Thesis advisor: Professor H.T. Kung Xavier M. Evans

Graph Embeddings for Harvard Widener Library Data

Abstract

In this project, I consider complications with retrieval of resources in a database, particularly library entries and a catalog, and consider how graph machine learning can aid in that retrieval process. Typical retrieval systems are exclusively text-based. This means that they will rely on natural language models only when making determinations about which resources might be relevant for the user. Natural language data is indispensable and at the core of how search engines operate. One can embed the query and all entries in a coordinate space and then find the closest entries based on some distance metric to get the most relevant search results. However, limiting the model to just this natural language data is letting go to waste one of the most powerful tools for understanding the relationships between entries: keywords.

By including keywords in the model, we can understand the hierarchical nature of the relationships between entries and keywords and among the keywords themselves, and this gives us additional information about the extent to which entries might be related when the text might be deceiving at face value. This introduces a new complication, however, as the keywords and entries are related in a graph structure. The keywords and entries are nodes and relations indicate the connection between an entry and a keyword that describes that entry. Since computers cannot look at a graph and make sense of it like humans can, we must use graph machine learning techniques to embed the graph into a coordinate space. Then, we can use the coordinates to make conclusions about the relatedness of entries and the relevance of search results. This gives use a new way to approach retrieval and can be paired with the common natural language model approach to enhance retrieval and return more relevant results to the user.

Contents

I dedicate this thesis to my best friend Jordan and my twin brother Isaiah who have been my rocks throughout this process.

0

Introduction

This research project is an extension of my work as an intern for the Godard Space Flight Center, a NASA facility in Greenbelt, Maryland which specializes in data collection and presentation. During my time working for NASA, I completed a similar project for the GES-DISC search engine. A natural language model was already in place, but researchers were finding that they did that suggestions were relevant on an extremely superficial basis. They were getting data sets that they didn't actually need, and it was clear why they would be suggested, given the query and similar words in the title, for example. When a researcher would search for climate change data, they would find that the data sets returned often had to do with other less related types of change, like radiation levels and concentrations of certain chemicals in the moon's soil. While these might be tangentially related to climate change, the more immediate relevant results would be things directly tied to it and direct consequences of it, like natural disaster data, snow and ice data, or ocean composition data. However, because these were potentially not obvious enough based on the textual descriptions of these data sets, they would not be deemed relevant enough to be recommended highly in the results returned to the user, and the scientist would be forced to dig for the data they needed.

To begin to resolve this issue of extreme literalness when it comes to textual similarity, we need to think on a macro scale—specifically a topical scale. In not grouping data sets by their textual similarity and instead by a combination of textual and topical factors, we are able to then return results that are a part of the general sphere of study they are investigating as well as the ones that a similar textually. This, intuitively, gives a more well-rounded understanding of what data exists that is related to what you search up. This was the motivation for incorporating keywords and, consequently, graph machine learning in this NASA use case.

This method of combining graph machine learning and natural language models to enhance search results showed such promise that I hypothesized that I could be helpful for the Widener Library search tools. During the internship, we were unable to finalize a model, yet we made some progress in the direction of embedding data sets, starting to understand how we learn from these graph structures to create embeddings that can augment natural language embeddings. It was then my hope to have some more conclusive reflections on whether incorporating graph machine learning could prove beneficial in the context of my senior thesis, and that marked the start of this investigation. Henceforth, we will discuss the methods and procedures that resulted in effective embeddings for a subset of the Widener Library data and reflect on their utility and potential improvements moving forward.

Many never realize they always had the key in their pocket, so they die at the locked door, never reaching deep inside to pull it out.

Anthony Liccione

1 Tools

Before we get into the good stuff, it is worth mentioning how I set up the infrastructure for this project and the tools I made use of. The coding was done in Python, specifically in the Jupyter Notebook environment, and I made significant use of Torch machine learning tools and NLTK natural language processing tools. I was able to run the script locally, however when increasing the size of the subsection of data we analyze into the tens of thousands, this would likely be infeasible. Besides the finiteness of computational power, another limiting factor was the API itself, which is documented at the Harvard Wiki. I needed to be careful about the frequency of the calls to the API and the number of calls I was making within fixed time intervals. Considering these limitations, I was careful in the design of my code to allow cooldown time to avoid upsetting the API. Then, when it was working reliably, I could retrieve resources in XML format using the following scheme:

```
url = 'https://api.lib.harvard.edu/v2/items.json?resourceType=text&' +
      'subject={subject}&limit={limit}&cursor={next_cursor}'
```

The subject field allows us to choose a topic for the returned results as opposed the simpler search term function. This means that we are not using their built in search engine but instead searching for entries related to this subject. This is the most important field, and this is our chose method for subsecting the massive database into a manageable section for analysis in this iteration of the project.

The limit field allows us to moderate how many results we consider at a time. If the limit is lower, fewer entries will appear in the resulting XML file, and if it is higher, we get more entries returned to us. I set this at an arbitrary moderate number of fifty, simply so I wasn't biting off more than I could chew at any given iteration of reading all the entries for the given subject.

The cursor is where the magic of iterating over all the entries happens. If we consider fifty entries to start with, we need to know how to find the next fifty without needing to generate one hundred; after all, our goal is to keep the XML file to a manageable size, and even if this were out of sight, it would be computationally taxing to generate all the entries leading up to the range we want to consider at a given range. By setting the cursor value, we are able to choose where exactly we start from, and by shifting the cursor to the next cursor returned by the XML at each iteration, we can keep sliding the window over the entries until we parse all of them.

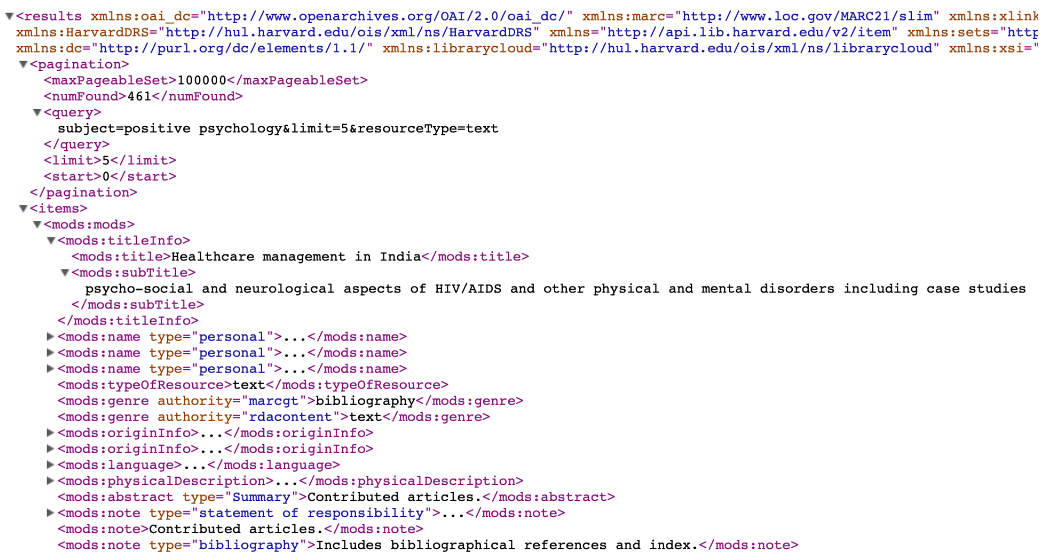
This is a bit more context for how the API works and how I made use of it and the extent of the resources that I needed to make this project work. Then, it was time to make sense of the data and get it into a reasonable form for graph machine learning.

Out of clutter, find simplicity. From discord, find harmony. In the middle of difficulty lies opportunity.

Albert Einstein

2 Data Wrangling and Exploration

Now we can make sense of the API results and start constructing a data structure for the entries. The API returns results in XML format, and a page—like this example for the subject Positive Psychology—can take the following form:



```
▼<results xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/" xmlns:marc="http://www.loc.gov/MARC21/slim" xmlns:xlink
 xmlns:HarvardDRS="http://hul.harvard.edu/ois/xml/ns/HarvardDRS" xmlns="http://api.lib.harvard.edu/v2/item" xmlns:sets="http
 xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:librarycloud="http://hul.harvard.edu/ois/xml/ns/librarycloud" xmlns:xsi="
  ▼<pagination>
     <maxPageableSet>100000</maxPageableSet>
     <numFound>461</numFound>
    ▼<query>
       subject=positive psychology&limit=5&resourceType=text
     </query>
     <limit>5</limit>
     <start>0</start>
   </pagination>
  ▼<items>
    ▼<mods:mods>
      ▼<mods:titleInfo>
         <mods:title>Healthcare management in India</mods:title>
        ▼<mods:subTitle>
           psycho-social and neurological aspects of HIV/AIDS and other physical and mental disorders including case studies
         </mods:subTitle>
       </mods:titleInfo>
      ▶<mods:name type="personal">...</mods:name>
      ▶<mods:name type="personal">...</mods:name>
      ▶<mods:name type="personal">...</mods:name>
       <mods:typeOfResource>text</mods:typeOfResource>
       <mods:genre authority="marcgt">bibliography</mods:genre>
       <mods:genre authority="rdacontent">text</mods:genre>
      ▶<mods:originInfo>...</mods:originInfo>
      ▶<mods:originInfo>...</mods:originInfo>
      ▶<mods:language>...</mods:language>
      ▶<mods:physicalDescription>...</mods:physicalDescription>
       <mods:abstract type="Summary">Contributed articles.</mods:abstract>
      ▶<mods:note type="statement of responsibility">...</mods:note>
       <mods:note>Contributed articles.</mods:note>
       <mods:note type="bibliography">Includes bibliographical references and index.</mods:note>
```

We can instead display this in JSON format to make it easier to import into Python. Upon doing so, the result will look more like the following:

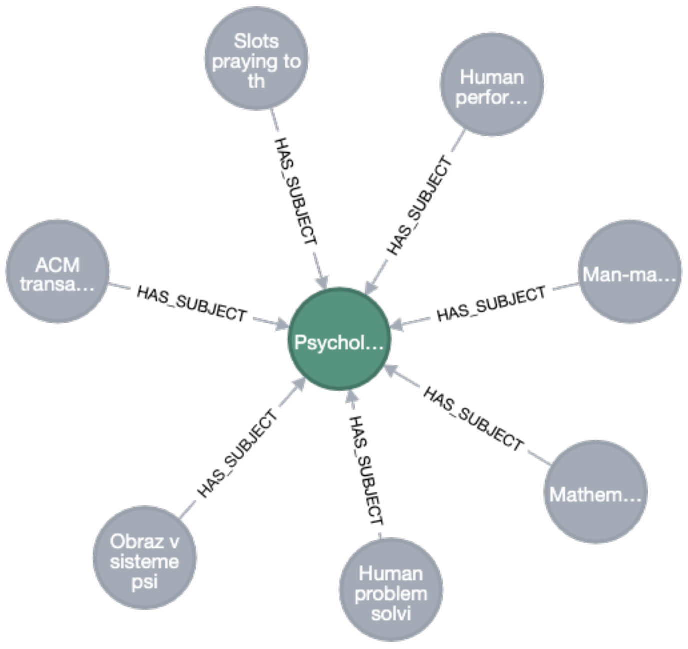
```
[
    {
        "titleInfo": {
            "title": "Guidelines for counselling about HIV infection and disease"
        },
        "name": {
            "@type": "corporate",
            "namePart": "World Health Organization"
        },
        "typeOfResource": "text",
        "genre": [
            {
                "@authority": "marcgt",
                "#text": "bibliography"
            },
            {
                "@authority": "marcgt",
                "#text": "government publication"
            },
            {
                "@authority": "2",
                "#text": "Guideline"
            }
        ],
        "originInfo": {...},
        ...
    },
    ...
    {...}
]
```

Then, it is rather straightforward to extract the desired information from the list of dictionaries. After we have extracted the information in the Data Wrangling section of our code, we get a list of dictionaries that have the following form:

```
[
    ...
    {
        'title': 'Positive Carers',
        'authors': ['Mayho, Paul.'],
        'editors': [],
        'abstract': 'There is not one positively identified case...',
        'subjects': ['Diseases',
            'Medical personnel',
            'Law and legislation',
            'psychology',
            'Health and hygiene',
            'prevention & control',
            'Infectious Disease Transmission, Professional-to-Patient',
            'Ethics, Medical',
            'HIV Infections',
            'HIV-positive persons',
            'Health Personnel',
            'HIV Seropositivity']
    }
    ...
]
```

Then, we are able to create Cypher queries based on all of the subjects and entries. Nodes of type `:Entry` are defined by their title and nodes of type `:Subject` are defined by their names. In the example above, we would create an entry node of the form `(entry:Entry {name: "Positive Carers"})` and at most twelve new subject nodes of the form `(subject:Subject {name: "Diseases"})`, `(subject:Subject {name: "Medical personnel"})`, and so on. The relationships go in the direction from entry to subject with the type `:HAS_SUBJECT`. Upon creating all the queries for node creation and then those for relation creation, we have a complex example graph database of 10632 nodes and 22042 relations. To have a sense of which subjects might be particularly popular, we can do some exploration with the neo4j Graph Data Science Playground. When using the subject

"Machine" as input for the API, we can run the page rank algorithm to see which subjects are most popular among the entries. "Machine learning," unsurprisingly, is by far the most popular with a score of 41.9, followed by "Machine translating (19.4)," "Machine-tools (15.7)," and "Machine theory (15.3)." For me, this simply serves as a sanity check that the subjects that we would expect to be extremely popular for a given search are indeed present and well-connected in the graph. This is also just an effective tool for checking that we don't have an epic error, like shifting everything by one such that the index of the ancestor node is accidentally pointing to a node that actually has an index of one higher than it is supposed to. By inspecting neo4j graph, we are able to spot these sorts of mistakes, and if everything seems congruent, then it is likely that we have accurately and effectively created the edge index, and we are ready to do some graph machine learning. Let's inspect it together.



In this example, we looked up the node with the name “Psychology” since the original subject for the API call was “Positive Psychology.” I then expanded it to display all adjacent nodes, and I wanted to confirm that it would make sense for all of them to have “Psychology” as a subject. Just a few examples of the names of those nodes are “Human performance and ergonomics,” “Man-machine simulation models psychosocial and performance interaction,” and “Human problem solving in dynamic environments understanding and supporting operators in large-scale, complex systems.” Since these all have to do with psychology, I feel confident in proceeding with machine learning knowing that we have linked up the entries with their corresponding subjects accurately.

For the things we have to learn before we can do them, we learn by doing them.

Aristotle

3 Machine Learning

Now we can start doing what we came here for: graph machine learning. Again, considering that the goal is to express the graph in a way that is interpretable by a computer, particularly quickly since it will be used in high pressure situations like making search engine recommendations, we want to embed the graph. It would be horribly inefficient to run graph algorithms each time we wanted to make a conclusion about the graph, so by training a model such that similar nodes are closer together and distinct ones are distant, we can precompute this essentially and save time later on.

To accomplish this, we make use of Stanford's GraphSAGE, a framework for inductive representation learning on large graphs. GraphSAGE is used to generate low-dimensional vector representations for nodes, and is especially useful for graphs that have rich node attribute information.[1] Conveniently, since entries have abstracts and subjects sit at the confluence of many entries, each and every node has a lot of rich text data from which to craft node features. A common choice for this is a bag of words representation, which is based on the counts for each word in a text.[3] This is a solid option as a first resort, but it doesn't account for the fact that the word "the" being present many times should not "count for as much" as the word "preposterous" if former is incredibly common among all the text data and the latter is not. This brings us to another strategy for node feature creation, which is term frequency–inverse document frequency—tf–idf for short. The name says it all. The frequency of the term itself is considered, but we also way that against how frequently it appears in the other documents. If it is common in this document but you hardly see it elsewhere, then it has a higher tf–idf score than something that is frequent here and frequent everywhere and even higher than something that is infrequent here yet frequent elsewhere. With this more sophisticated cousin of the bag of words method, we can now create node feature vectors to be the tf–idf representation of the text for each node in the graph. We use sklearn to accomplish this efficiently, and then we are ready to create the model.

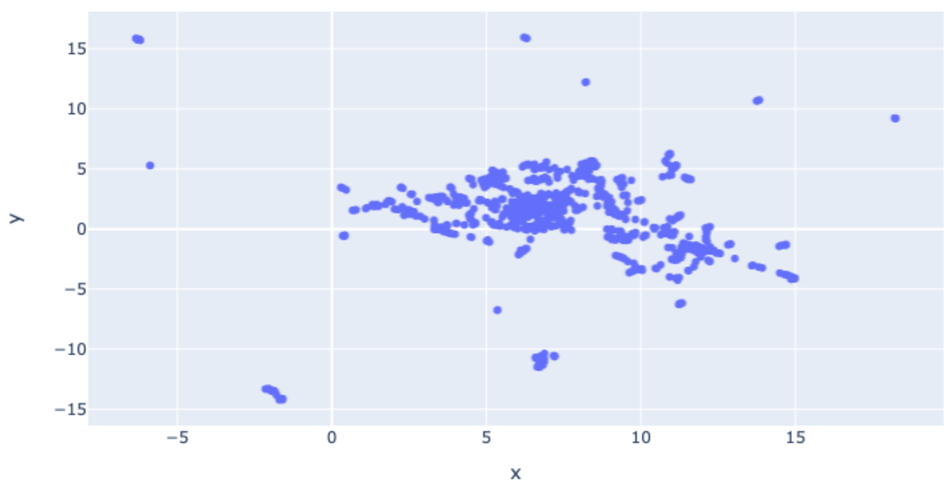
Then, we have node features and an edge index to represent the connections between nodes in COO format. We create the node classifications based on whether they are a subject or entry, meaning y takes the form of a vector comprised of ones and zeroes exclusively. We create a PyTorch Geometric Data object of the form `data = Data(x=x, y=y, edge_index=edge_index, train_mask=train_mask, val_mask=val_mask, test_mask=test_mask)`. The masks simply arbitrarily partition the nodes into a training, validation, and test set for tuning the model. Using neighbor sampling and Stanford's GraphSAGE, we are able to learn embeddings (see code).

Reflective thinking turns experience into insight.

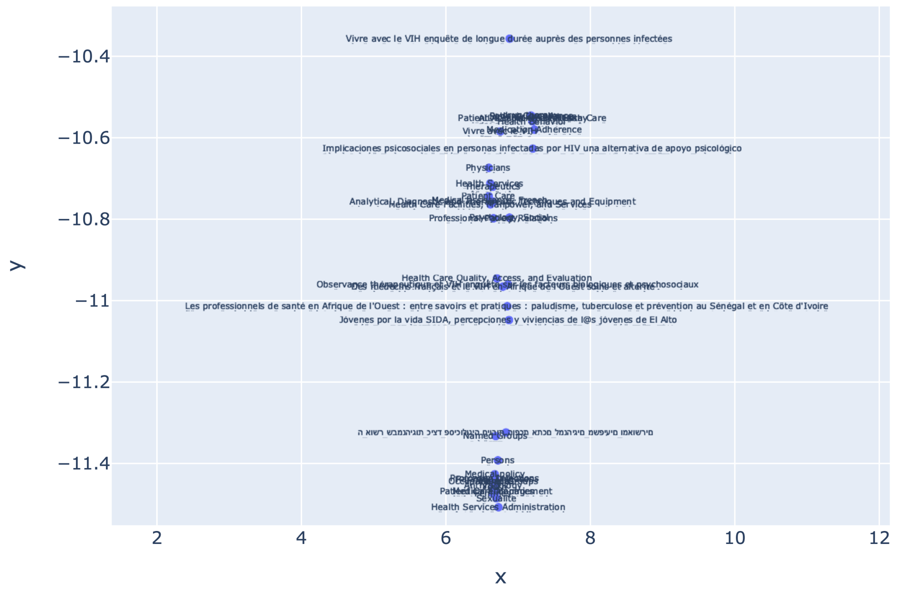
John C. Maxwell

4 Analysis

Now that we have learned embeddings for the entries and subjects, we can project them down onto a two-dimensional space and analyze whether the positions make sense. We do this using the dimensionality reduction algorithm UMAP, short for Uniform Manifold Approximation and Projection[2]. This results in the following visualization:



We proceed to investigate some clusters to see if the embedding seems accurate.



This comes from the bottom center of the original graph. We see that all of these have to do with health, and toward the top, I see many names that translate to "living with HIV" or some variation about that. We also see a title that translates to "the psychological implications of people infected with HIV." We also see the subjects of "Physicians" and "Health Services." To me, this is clear that nodes that are extremely similar are embedded close to each other, and so this indicates that the embeddings are accurate.

This result is quite promising. Now that we know it is possible to embed the entries in the Harvard Widener Library database using the subjects to create a graph structure, we know that we can leverage this important additional piece of information to create even richer embeddings. Relying solely on text data is still quite effective, but by adding this additional layer, there is promise of further improving the retrieval process to return even more relevant results.

If you don't think about the future, you cannot have one.

John Galsworthy

5

Implications for Future Study

Now that we have embedded the entries, as this project evolves into the spring, we would proceed to merge the natural language model with the graph machine learning model to leverage both of these effective technique simultaneously. We would also use a language model that allows us to instantly embed a search query into our embedding space. This is all endgame, so there aren't many remaining steps to get to that point, but these final steps would allow us to begin testing intensively and make conclusions about whether we do indeed get more relevant results when we incorporate graph machine learning on top of natural language.

6 Conclusion

Through doing this project, we have seen that it is possible to effectively leverage the subjects that are associated with each of the entries in the Widener Library database to create a graph, upon which graph machine learning can be conducted to learn rich embeddings. These embeddings can be used to enhance retrieval and return more relevant search results, but more testing is needed to confirm that there would be a significant improvement. In the spring, I will continue work on merging the two types of models and finalizing the structure of the search feature to begin doing rigorous testing. Still, I have laid significant groundwork for data collection and wrangling and the entire scaffold for the machine learning that will be relevant for creating these graphing embeddings. I am eager to continue this work and enter the next phase of this research.

References

[1] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *NIPS*.

[2] McInnes, L., Healy, J., & Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction.

[3] Rajendran, C. (2020). Text classification using the bag of words approach with nltk and scikit learn.