Embedded queries and Mining Techniques

Brian Beer and David Overgaard

Abstract—Data mining aims at extracting knowledge from data. Information rich datasets, such as EBSCOhost Newspaper Source [3], carry a significant amount of multi-typed current and archived news data. This extracted information can easily be constructed into a heterogeneous information network. Through use of many mining techniques, deeper comprehension can then be unearthed from the underlying relationships between article, authors, tags, etc. This paper focuses on building on two such techniques, classification and embedding. GNetMine [1] is a common classification algorithm that is able to label entities of different types through a small set of training data. Node2Vec [5] is an embedding approach, using the many nodes and edges in a heterogeneous network, and converting them into a low-dimensional vector space, that can quickly and easily allow for comparison between nodes of any type. The goal of this paper is to combine these methods and compare and contrast the quality of output of Node2Vec with unlabeled data, directly from the heterogeneous network of EBSCOhost sports news data, as well as adding learned classification labels. Can nodes labeled using classification as an input to network embedding, improve the outcome of the embedding results?

I. INTRODUCTION

A. System methodology

The intention of this research is explore the potential improvement to embedding, with the addition of classification labels, complementing an information rich, heterogeneous information network. Our test data is a set of sports articles, that includes their authors, locations, tags, titles, abstracts and section number in the New York Times [2] (retrieved via EBSCOhost). A heterogeneous information network can be constructed using this multi-typed data to link relations between all types of nodes. This network lays the foundation for turning the data into knowledge.

At a high level the system we are constructing has six major blocks, 1) raw EBSCOhost data, 2) data extraction, 3) frequency analysis and labeling, 4) GNetMine classification, 5) Node2Vec and 6) node similarity (evaluation). The raw EBSCOhost data is pulled from a query of 7 year of New York Times articles focusing on sports and is provided in a XML formatted dump. The XML format provides a relatively easy way to move the raw data along to the next stage, data extraction. The raw XML data is parsed, including various text information, such as title, author(s), tags, location and section. Each type of node is then defined with unique IDs assigned. For every combination of two types, a set of relations (an edgelist) and their weights is generated. A series of lookup tables between the uniquely assigned IDs for each type is created in order to extract the original data at any time. At this stage a heterogeneous information network has been formed from the raw data. The next phase involves processing and extracting information from this network.

In a broad sense, classification provides labels to unclassified objects. GNetMine [1] is one such approach to assign a label to each node. The algorithm requires manually annotated training data, in order to the propagate the labels of known entities to unlabeled ones throughout the network. A set of relation files between nodes of various types, representing our heterogeneous network, are fed into GNetMine in conjunction with the manually generated training data (on a small subset). The GNetMine algorithm is then run in an iterative process to produce a steady-state of unchanging, labeled data. In our system we parse all of the network information, both of the original heterogeneous graph, as well as the newly created label files and create a full set of edge-lists of the entire graph. Node2Vec takes the edge-list file(s) and converts the weighted edges between our undirected graph into a low dimensional vector space. This produces an embedding file that has the corresponding vector for each individual node. Finally, we take advantage of a pre-existing Word2Vec [11], Python-based, library that allows for importing an embedding file and provides utility functions for ranked comparisons between nodes and indirectly, between each different network embedding.

![Figure 1: High-level architecture](image)

Evaluation of this method, requires a set of network embeddings to be created that exclude the learned label information from GNetMine, as well as those that include them. This serves as a control in order to fairly assess this technique. The resulting similarities between select nodes can then be compared for each network embedding to understand if
improvements can be realized by inclusion of generated labels. A graphical representation of the complete system can be seen in Figure 1.

B. Paper outline

The paper generally progresses through each component of the system architecture as shown in Figure 1. Section II describes the data extraction used to reformat the raw data into data used to construct the heterogeneous network. Section III discusses different methods to create the best training set for a semi-supervised classification algorithm, in particular GNetMine. Section IV introduces GNetMine which is used to provide additional data about the network through classification. Some observations about the dataset and potential concerns are also highlighted. Section V gives an overview of Word2Vec and how it applies to Node2Vec as well as the query methodology. Section VI talks about Node2Vec which as an extension of Word2Vec and is used to embed the heterogeneous network after it has been projected into a homogeneous space. The results of our experiment are presented in section VII. In section VIII we summarize our conclusions, and in section IX we propose future directions for this experiment to further refine the quality of embedded search results.

II. DATASET AND EXTRACTION

A. Overview

Before any data mining can occur the raw data had to be transformed into a heterogeneous network. To simplify this process a preprocessing dataset extraction step was added to generate a set of node and edge files that would simplify constructing the network for any algorithm. For this experiment GNetMine [1] and Node2Vec [5] use these files to construct the networks.

B. Dataset description

The raw dataset for this evaluation is metadata from sports related articles, over last seven years, published in The New York Times [2]. This data was extracted from the EBSCOhost Newspaper Source [3] in XML format. The dataset includes 23,861 articles, 21,040 titles, 1,174 locations, 22,477 tags, 6 sections, 2,944 authors, and 23,727 abstracts. The articles are not fully populated and may be missing one or all of the listed types.

![Figure 2: Schema of a typical article](Image)

Each article can essentially be modeled as the star schema [4] as shown in the Figure 2 where outer nodes often end up connecting to other articles. While all of the data types were not used throughout this project they were all processed to allow for greatest flexibility when exploring the data.

C. Extraction methodology

To allow for the greatest flexibility of graph construction and analysis for each type pair combination, a file describing the association between entities of those two types is created. This excludes the inverse relation i.e. if a node to node edge file is created for article-author then author-article would not be created. It is assumed to be simple for downstream components to either parse the file using opposite terms, or if stored in a large matrix, to use transpose. Each node is represented by a unique index per type and the pairs describe a link with a third field which includes the frequency of the relation to provide information about the weight of the association.

A verbose description of each entity index is also provided for each type, such that it is trivial for either the user or an algorithm to decode the original data.

Currently the slowest component of extraction is parsing the raw xml file rather than the manipulation of data into the desired format. The extraction component was tested on a dataset 3 times larger than the set used in this paper and it completed in less than 10 minutes.

D. Data Extraction Tips

Data extraction especially from a large dataset can be very time consuming. A toy example (0.1-1%) of the dataset can be used to validate key parts of extraction and also provide a small output file which can be used for similar verification by later steps. This helps to speed up cycles of development before producing the final product.

It is beneficial to use time wrappers or debug statements to profile key parts of the program and helps identify slow modules. This also assists in quantifying the potential benefit of any optimizations. For example this project required random access printing of a value stored in a list along with its associated index. This was first done by printing the value and then looking up the associated index (value assumed unique). This was acceptable for the toy dataset, but when expanding to the full input file, this operation was identified as a performance bottleneck. Rather than finding the index in the array, a dictionary was created to do this lookup.

Our optimization reduced processing time from approximately an hour to less than 3 minutes. This makes sense since the average case list index is O(n) to the number of elements in the list and after being reorganized to a dictionary key-value lookup is O(1) [7]. The specific functions and classes that are expensive will change based on the language in use, however the general practice can be applied to any language.

We then generated all possible combinations of node-edge files, for each type relation, to enable exploration and development without unnecessary constraints or delays due to missing data. This technique was used to debug software downstream.
III. FREQUENCY ANALYSIS AND LABELING

A. Labeling overview

The objective of labeling a portion of the dataset is to provide training data for GNetMine [1], ultimately creating labeled article, author and tag nodes for Node2Vec [5]. After manual inspection of a portion of the tags and abstracts, 15 categories were identified: baseball, basketball, football (American), soccer, tennis, fishing, cycling, track and field, winter sports (skiing, snowboarding, etc.), hockey, swimming, boxing/mixed martial arts, motorsports, golf, and Olympics.

Although the network contains many data types, author, tag, and article id were determined to be the most relevant for use with GNetMine [1]. One of the challenges with labeling these entities is that some articles are highlights from several sports and do not have a clearly fit a single label. Perhaps the biggest struggle though, is manually labeling the training data is exceptionally time consuming. For example, data types such as author require internet research and each article requires careful reading before a label can be assigned. This methodology is not easily scaled for large datasets.

For these reasons, techniques were developed to efficiently label a sufficient number of entities, such that quality labels could be generated for each article, author and tag. These are described in detail in the following subsections.

B. Methodology

1) Frequency analysis

With tens of thousands of entities in the network and limited ability to label nodes, the decision had to be made on which nodes to label. It was decided that node selection, on authors would use frequency counting. In particular, we look at the number of articles written, by a given author, as a way to identify the authors of greatest influence on the network. This methodology loosely aligns with the concept of selective labeling discussed by Sheng et al. [6], demonstrated to have consistent improvement over randomly labeled entities.

Graphing the number of articles written by an author relationship revealed a power law distribution for this association as seen in Figure 3. This implies that a majority of the edges connecting with authors can be partially labeled by labeling a small set of prolific authors. This observation is another example of edges in a network following a power law distribution; not only for the web, city populations, and protein interaction [8, 9], but also the number of articles written by a reporter. This observation was confirmed by extracting the similar data pertaining to authors of the Toronto Star [10].

2) Unambiguous labeling

Labeling authors, articles, and most object types for this dataset is very tedious, as previously described, but tags have many entities which are unambiguous and easy to label. Tags are essentially preprocessed keywords that are available for this dataset and generally 1-5 words in length. These tags have many direct references to the 15 target categories. For example, there are 308 tags (1.4%) which contain the word ‘football.’ By searching the tags for exclusive mention of one target category, ~7% of tags were labeled with minimal effort. This accounts for ~3.4% of the total number of nodes used by GNetMine (author, article id, and tag), sufficient to achieve reasonable classification results.

Although tags are specific to this experiment, each type should be examined for any dataset to identify ‘easy to label’ entities. This allows more focused effort to be spent labeling the most influential nodes, as described in the previous section. The result is quality results of the query type, as fast as possible.

3) Commercial services and repeated labeling

Services such as Rent-A-Coder, Amazon’s Mechanical Turk, and CloudFlower provide inexpensive human labeling, however the labels provided can be noisy. Although this technique was not used for this paper, it is another feasible solution to generate training and test labels. To make effective use of this noisy labeling data, the dataset should be labeled several times. Either a round robin or preferably a selective repeated-labeling algorithm should be used to provide high quality results, as shown by the work of Sheng et al. [6] on this topic.

C. Future plans for labeling

More time could be spent comparing the performance of different labeling techniques such as selective frequency based labeling on different object types. This approach was not pursued further, given that it is not the focus of this paper. Getting the full dataset labeled using one or more of the commercial services would be required for this evaluation. Which data type is most influential could also be investigated, to see if this further improves the selective labeling technique.

IV. CLASSIFICATION WITH GNETMINE

A. Classification overview

The purpose of classification at this stage is to provide article, author, and tag labels to Node2Vec [5]. With additional labeling information it is expected that the quality of Node2Vec embedding and in turn node similarity calculations and comparisons will be improved. GNetMine [1] was selected as
B. GNetMine overview

GNetMine [1] is a transductive classification method that supports heterogeneous information networks. Supporting multi-typed networks helps to capture domain specific semantics and allows for an intuitive propagation of labels throughout the network. GNetMine is a semi-supervised algorithm which requires some entities to be seed labels. For our dataset the seed labels were tag nodes.

The heterogeneous nature of the algorithm allows labels from one data type to propagate to other data types. This allowed the labeled tag nodes to provide classification for the other types and nodes. As information is propagated from labeled to unlabeled objects there will be some objects such as ‘Paper P2’ in Figure 4 which have multiple potential labels. To resolve this issue, the probability that a node is assigned to each label is determined. Classification with the highest probability is selected as the final label.

This algorithm is a single label model and does not support partial node labeling. Relation and diagonal matrices are built between each combination of types as described by the user defined meta-paths. The process of label propagation is repeated iteratively until it converges and the system is stable.

For our experiment associations between article-author, article-tag, and author-tag were selected as meta-paths. This fully connects the nodes for our classification dataset and allows for relations, such as co-authorship and tag association, to fully propagate the labels from seed nodes.

![GNetMine Network Classification](image)

**Figure 4: GNetMine Network Classification [1]**

C. Performance

To evaluate the quality of the classification some of the labeled tags were retained as test labels. A handful of articles were also semi-randomly selected from the dataset and manually labeled as test data. The results of this effort are shown in Figure 5.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Entity nodes as train</th>
<th>Entity nodes as test</th>
<th>% of network as train</th>
<th>Accuracy on test nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tags</td>
<td>0.74%</td>
<td>6.62%</td>
<td>0.34%</td>
<td>33%</td>
</tr>
<tr>
<td>tags</td>
<td>3.68%</td>
<td>3.68%</td>
<td>1.68%</td>
<td>71.02%</td>
</tr>
<tr>
<td>articles (tag nodes)</td>
<td>7.35%</td>
<td>0.14%</td>
<td>3.35%</td>
<td>81.82%</td>
</tr>
</tbody>
</table>

**Figure 5: Test classification results**

Given the small percentage of training data the accuracy on test nodes, was sufficient to evaluate our hypothesis of improving Node2Vec embedding by including labeling as additional information. The articles which were labeled were semi-random in that several articles have ambiguous labels and where not used in the test set. For this reason the measured accuracy on articles may be too high, but full evaluation will require commercial labeling of the entire dataset.

All available tag labels (total of 3.35% of nodes in the network) were used to produce the labeled article, tag, and author files. These files are the additional node and edge data which was used as additional information for Node2Vec embedding.

D. Limitations of dataset and GNetMine

The biggest challenge with this data set in terms of classification is that many entities are mixed and do not have a single clear label. For example, the articles which were misclassified from the test set shown in Figure 5 were written by an author who also wrote articles in many different disciplines. For example, a soccer article written by Jere Longman became classified as track and field, a fishing article by Zach Schonbrun was classified as football. In addition to soccer, Jere Longman also wrote about track and field, Olympics, baseball, and basketball. Zach Schonbrun wrote about most collegiate sports including basketball, football, baseball and tennis. Similarly, there are generalized tags such as ‘SPORTS – News briefs’ which are applied to essentially every sport.

Another observation is that the misclassified items tend to gravitate toward the more popular sports: baseball, football, and basketball. This could be the result of these sports representing the majority of the network which in turn have a greater influence on author and tag labeling than the less frequent sports. Generalized tags such as ‘SPORTS – News briefs’ could also be labeled as the most popular sport and then spread that classification bias to other tags and articles.

Lastly, a disparity in the number of labels between types, can also be seen. The most popular sports; baseball, basketball, football, have many more entities than fishing or boxing. This makes it difficult to provide a sufficient number of quality labels for each data type.

E. Output for Node2Vec

The final output for this stage of the program is a collection of node edge files connecting article, author, and tags to one of the 15 new label nodes. Each of these label nodes has many edges and should help to differentiate articles and their attributes (star schema of Figure 2) such that, after mapping into a lower dimension space with Node2Vec, the vector comparison between two entities will be more clear.

Several support files were also passed to later steps to help decode the network index numbers used to denote a specific node back into the author name, tag words, etc. This greatly simplified the final parsing of query results into human readable format.
**F. Future plans for Classification**

Limitations in the dataset, particularly the mixed entities, show the need for partial labels assignments to authors, based on context. An algorithm which supports content aware noisy entity typing such as PLE [12], AFET [13], or CoType [14] could be used to address the conflicting labels for an author or tag. This likely would improve the classification quality since there are numerous entities with overlapping types.

It would also be interesting to observe how different meta-paths for GNetMine may the change the final query results. Potentially meta-paths which align more closely to the type of user desired queries would further improve the quality of the results.

**V. EMBEDDING - WORD2VEC**

**A. Introduction of Method**

Word2Vec was designed to take advantage of the continuous Skip-gram model in order to efficiently learn a large number of syntactic and word relations. Skip-gram works very well on large amounts of unstructured data, in this case, a single machine can easily train on more than 100 billion words per day [11]. The general idea behind this algorithm is, with a target word, find the probability of getting the context. This can also be looked as a context prediction of the surrounding words in a sentence or document. The resulting matrices are stored for each word. A visual overview of the skip-gram model can be seen in Figure 6.

**Figure 6 - Skip-gram model architecture [11]**

**B. Vector Similarity Measurements**

Once a set of matrices are generated for each word in the corpus, the linear structure of the skip-gram model’s output allows for simple vector arithmetic operations to perform analogical reasoning. This includes generating similarity scores between the vector representations of two words. In turn that enables generating a ranked list of the most similar words to any query word. Because of this property, once an entity is in vector space, the arithmetic nature of the matrices is the same for any kind of data, including nodes. This will be further discussed in the next section, covering Node2Vec [5].

**C. Module for Vector Comparison**

Gensim [15] (https://radimrehurek.com/gensim/) is a Python library that implements many topic modeling algorithms and provides APIs for loading data, modeling and evaluation. For this research we are interested only in the evaluation portion of the Word2Vec module. As noted earlier, Node2Vec generates embedding files that are formatted with each node ID and a corresponding NumPy matrix. The Word2Vec package within Gensim allows for importing these embedding files and simply maps the node ID to the corresponding NumPy matrix. Once imported, it is simple to use the built-in comparison functions directly on the nodes. In particular, three functions are of interest. The first, ‘similarity(a, b)’ can provide a numerical similarity (a float 0 to 1) between two nodes. Secondly, and the main focus of our evaluation, is ‘similar_by_word(a, b)’ can provide a numerical similarity (a float 0 to 1) between two nodes. Lastly, ‘most_similar(positive, negative)’ provides means of adding (positive) sets and subtracting (negative) sets of nodes, providing a ranked-list of the most likely nodes pertaining to that criteria. This provides a framework for evaluating the network embedding files against each other. Next we will take a deeper dive into the embedding method used in our research.

**VI. EMBEDDING - NODE2VEC**

Node2Vec is an extension of Word2Vec, predicting network node vectors [5]. While there are differences in the approach and construction of the low dimensional vectors, the overall idea and more importantly, comparisons between vectors are very similar, hence the brief introduction into Word2Vec [11] in the section prior.

Node2vec, in a similar spirit as Word2Vec, seeks to learn the mapping of nodes into a low-dimensional space. This occurs on nodes in place of words. The output vectors maximize the likelihood of preserving network neighborhoods of nodes. Similar to word2vec, feature representations are learned using stochastic gradient descent. Word2Vec performs this under the consideration that words in similar contexts have similar meanings [11]. Analogous to this, Node2Vec can be seen as an ordered sequences of nodes when sampled from the underlying network. Node2Vec does not have a strict definition of the distances or hops that compose a neighborhood. Networks can be traversed by either breadth first search or depth first search (BFS and DFS respectively) [5].

As with Word2Vec, feature learning is performed by stochastic gradient descent. Predication is based on two kinds of similarities, homophily and structural equivalence. Homophily state that nodes that are highly connected and belong to similar clusters should be embedded closely together. In contrast, structural equivalence states that nodes that have similar roles, rather than connectivity, should be embedded closely together. BFS generally leads to embeddings that emphasize structural equivalence. Homophily is more naturally represented by DFS. In order to provide a balance between BFS and DFS search strategies, a random walk procedure is used. Groupings of nodes have flexible neighborhood sizes by simulating biased random walks to alternate between breadth
and depth first searching in order to accurately represent real networks as a mixture rather than either extreme [5]. The random walk approach taken by Node2Vec can be observed in Figure 7.

![Random Walk Procedure](image)

**Figure 7 - Random Walk Procedure [5]**

A. Vector Similarity

As with Word2Vec, the vector-space output allows for easy comparison between objects, or nodes in this case. Simple vector arithmetic operations can again be used to perform analogical reasoning between nodes. Generating similarity scores between two nodes can easily and quickly be calculated. Most importantly a ranked list of the most similar nodes to any query node (that exists in the embedding) can be generated on the fly.

B. Project Module

This paper makes use of the publically available Node2Vec source code ([https://github.com/snap-stanford/snap/tree/master/examples/node2vec](https://github.com/snap-stanford/snap/tree/master/examples/node2vec)). Both Python and C++ based source is available, however from sampling both, the compiled binary produced from C++ source provides a much higher performance solution for our dataset. Cloned via the git repository, we compiled the Node2Vec binary on Mac OS X with GCC. Our Python code creates the input and analyzes the output of the Node2Vec binary.

Data entry for Node2Vec is created in the form of an edge-list. An edge list is essentially the same input that goes into the GNetMine algorithm. It is a text representation of the network delineated by the two nodes (with unique IDs) and an optional weight parameter to create a single edge. Our heterogeneous information network is undirected and weighted, thus our edge-list and program arguments must reflect this in order to get the expected embeddings. A custom Python program takes our typed input information and performs remapping of each node to a unique ID for Node2Vec processing, we call this the node’s “embedding ID.” The limitation of the Node2Vec algorithm/source, is that every node, of all types, must be mapped into a single unique integer embedding ID space. Our program maps the unique IDs, specific to each type, into the embedding IDs. For our specific dataset, an example of this mapping is located in Figure 8.

![Type to embedding ID mapping]

**Figure 8: Type to embedding ID mapping**

After conversion to embedding ID, an edge-list can easily be generated from the desired input relations (edges and weights) between node types. Multiple edges-lists were generated with various combinations of relations, both with the corresponding labels generated by GNetMine (modeled as a new type of node) [1] and without. These will be visited in greater depth in the results section. Node2Vec is then ready to take the input edgelist(s) and generate corresponding embedding file(s). Each edgelist was run through the compiled Node2Vec binary and generated a network embedding file. The embedding file now contains the embedding ID for all nodes in the edgelist and is accompanied by a low-dimensional vector.

Evaluation of the raw vector data is performed using the Gensim [15] Word2Vec module, because of its ability to load arbitrary embedding file and perform several comparisons. Detailed descriptions of these are outlined in the previous section. Our Python code loads multiple embedding files and generates results for the top nodes most similar to a set of nodes that we query. The output of such a query is a ranked list of node embedding IDs and the similarity score. Through the use of look-up tables (reverse direction from Figure 8) our code is able to extract the original type and type-specific node ID from our heterogeneous information graph. Moreover, for any node in the embedding, we are able to retrieve either the top most similar nodes of any type, or filter on nodes only of a certain type.

For flexibility of comparison, and analysis, multiple network embedding files were generated, each with different sets of relations included. We selected the relations listed in Figure 9. Every chosen network has a version both with and without the GNetMine produced labels. For each, a corresponding edge-list and ultimately embedding file was generated. Our code imports the resulting network embedding files and runs a set of similarity searches, that includes the top ranked nodes for the 1) most published authors, 2) a random set of articles, 3) all labels, 4) a set of random locations and 5) a random set of tags. The final output is a ‘result.csv’ file that includes the top ranked nodes, their type, unique type-specific ID and the human-readable data specific to that node (ex. name for author, title for article, etc.) for every network embedding. In the next section we will be analyzing these results.

VII. RESULTS

A. Overview

Results are displayed by embedding the same dataset with different combinations of the base data types. Then an identical network embedding is created, with the addition of
A common data mining task might be to rank the similarity a particular author node, who has published in our dataset. This begs the question, what authors should we look at and intuitively who should be the most similar authors to them? During evaluation we decided to select the top 50 authors, by number of articles published, in the sampled New York Times/Sports dataset. Richard Sandomir had the largest number of published articles. Figure 10 examines the top 3 authors from various Node2Vec network embedding files, generated using our program. As a reminder, each odd embedding number exclude the GNetMine labels, while subsequent even numbers include the GNetMine labels.

The most interesting observation can be spotted by considering the top ranked result for all embeddings. Those containing GNetMine labels (even network embedding numbers) all have ‘SANDOMIR, RICHARD’ as the top ranked result. It can easily be understood that many authors appear be the same person, although there are inconsistencies in capitalizations, inclusion of middle name (or initial), order, abbreviation, etc. A reasonable conclusion follows, if our solution is able rank a different spelling (or capitalization) of the same author as the most similar author node (with a different author ID), it proves that it does a good job at ranking in general. In this way, network embeddings 2 and 4 (with the addition of labeling) show an improvement over embeddings 1 and 2 respectively (without labeling). GNetMine labels serve to promote the author with the different capitalization to the top ranked author results. In contrast, both embeddings 5 and 6, share similar results with and without labeling. In conclusion 2 of the 3 Node2Vec embeddings, with GNetMine labels, show an inherent improvement in the most similar author to our dataset’s top published author node.

2) Karen Crouse (Golf reporter)

Ms. Crouse is a reporter who primarily covers golf. This can easily be seen by reviewing the articles she has published. Using our methodology, the query results show that adding label information consistently improves the quality of the results. Embedded networks 4 as well as 6 (results for 6 are not shown for brevity – both networks are without label data) list a book about Harper Lee as being most similar to Ms. Crouse which has nothing to do with golf. The same networks with label nodes added provides results related to golfers and tournaments as shown in Figure 11. The book about Harper Lee does’t appear within the top 10 results for these networks.

C. Article query

SPORTS BRIEFING is a title used by many articles that cover the highlights from a variety of sports. Although network 4 gives a couple of semi-general tags; GYMNASTICS competitions and SPANIARDS in sports, when labels are added the top results are all articles with the same title. Returning articles with the same title is a better result because SPORTS BRIEFING in its nature covers multiple sports. Without a tag or label to describe a combination of all sports, providing articles that also cover several sports is better.

With all of the types of nodes included, network 10 was able to return a single SPORTS BRIEFING article but when labels are added two of them were returned. It appears
that with all of the nodes added the search results degraded somewhat since they include ‘TAVERAS, Oscar.’ Further investigation may be warranted to investigate which features add value and which degrade the query results.

### D. Location query

<table>
<thead>
<tr>
<th>Network No.</th>
<th>Result position</th>
<th>Result type</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>location N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>location N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>location N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>location N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>location GERMANY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>location GERMANY</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 13: Location 'Berlin (Germany)' query results**

In certain situations, no subjective difference between embeddings can be observed. One such example is a ranked list of locations, found by querying a location node, “BERLIN (Germany).” As seen in Figure 13, all embeddings (that include location) rank “GERMANY” as the top most similar location node. In this instance, we see good overall performance on the most similar location node for all Node2Vec embeddings, regardless of labeling. The results thereafter are mixed in a few of the network embeddings. Reviewing Figure 13, we can see that we did not produce a GNNetMine classification/labelling of location, rather only article, author and tags. Intuitively we did this with the understanding that, at a given location, many different sports will be played. For example, most countries will contain most, if not all, of our sports categories. Thus, it seems inaccurate to classify these locations with our labels. While more specific towns and cities may certainly lean much more towards a single label, we have no way to tell the difference. We then must conclude that classification data, particularly when applied to types other than the labeled types, do not show any noticeable improvement in our embedding space.

### E. Label queries

Querying on the label nodes themselves produced some interesting observations as well. For popular sports such as basketball and football (shown in Figure 14) networks with most or all of the types gave high quality results. The networks with only a few node types (network 5) gave mostly useless results. This shows that the more data types included in the network, the higher the quality of the label results even after projection into a single homogeneous network. Boxing is an example of one of the less frequent sports and its results are mostly meaningless. The first result is in fact a boxer, subsequent results are miscellaneous, but potentially sponsors of some boxing event. Although the embedded networks were able to give reasonable results for the popular sports, ones with few nodes and edges performed poorly.

### VIII. Conclusions

In this dataset there are many entities which are miscellaneous and do not fit well into any category. If these noisy entities are classified into one of the smaller sports, it could have a larger impact on the embedded vectors. Less prevalent sports do not have enough nodes to overwhelm the miscellaneous entities. With more data in the network it is likely that boxing would also yield better results, but this would require either a more balanced dataset or a much larger dataset.

### IX. Future Directions

Following this research, a few approaches stand out as potential focus areas for development. From a data extraction standpoint, it would be interesting to try this approach on networks of different domains. Additional cleanup of networks, such as removing entities that do not apply to any of the target labels, could potentially further improve results. In order to produce high-quality output, we must have high-quality input to our system.

Applying tiered classified categories, would be another interesting future direction. For example, entities could be labeled into sports, but each article could also be associated with the appropriate level of competition; recreational, professional, amateur.
collegiate, semi-pro, pro, etc. Multiple layers of classification could provide even more data into the network to reduce sparsity, resolve noisy labels and return higher quality search results.

If using just a single level of classification or if using multiple, it would be interesting to use an algorithm which supports soft labeling to improve label quality, which may help to further resolve the noisy entity mentions. Candidate algorithms for this approach are PLE [12], AFET [13], or CoType [14].

Classification is a time intensive task; as new datasets emerge old training data becomes stale and likely will perform poorly. This requires an updated training set. Human annotation of such data is expensive in both time and effort and simply does not scale. For this reason, we believe it would make sense to investigate learning algorithms that do not require training data. Clustering algorithms seem like a natural fit, given that they are able to segment the network into naturally learned buckets without training data. Once such example on heterogeneous networks is NetClus [16]. Learned clusters could then be fed into Node2Vec and resulting network embeddings compared in a similar fashion.

In conclusion, there are several different paths that may be taken to further improve results. Additional investigation in these areas, could lead to a more refined system, capable of creating higher quality network embeddings.

X. REFERENCES

[3] EBSCOhost Newspaper Source. Provided by CARLI University Library, University of Illinois at Urbana-Champaign. <http://sfx.carli.illinois.edu/sfxuiu/az>