Learning Links Between Low Level and Preferred Medical Terminology with GNNs

Stanford CS224N {Custom} Project

Sophia Kivelson
Department of Computer Science
Stanford University
skivelso@stanford.edu

Abstract

Relational databases like MedDRA can be valuable tools for learning technical medical language. In this project, I am interested in leveraging the tree structure of the MedDRA ontology to learn node embeddings for the natural language phrases contained at each level of the ontology. I begin by inferring a graph structure from the hierarchical relations defined between phrases in MedDRA. I then use a Graph Neural Network (GNN) to learn node embeddings which can be used to predict relationships between "low level terminology" - i.e. the kind of phrases used when doctors are talking to patients - and "preferred terminology" - i.e. standardized technical medical jargon. In this project I explore the effects of modeling the ontology using different graph structures (both homogeneous and heterogeneous). I further explore the effectiveness of pairing the GNN with various Bert models. My clearest finding is that the use of a heterogeneous GNN significantly outperforms a standard GNN in all experimental settings. I find the heterogeneous GNNs are, on average, able to achieve approximately 70% accuracy in predicting the links between low level terminology and the appropriate preferred terminology.

1 Key Information to include

- External mentor: Originally Marie Humbert-Droz, though I have deviated from her project - only sharing a choice of dataset with the original project.

2 Introduction

There are two aspects of this problem that are of interest for different reasons. Firstly is the natural language task I am working on, namely predicting links between 'low level' and 'preferred' terminology in the MedDRA database [1]. MedDRA, or the Medical Dictionary for Regulatory Activities, contains the medical terminology used during the regulatory process for biopharmaceutical product development. It is well known for its internationally utilized coding system, which assigns unique identifiers to each phrase in the database. However, for my purposes, MedDRA is primarily of interest for the medical ontology it provides. Presented as a relational database, MedDRA defines a hierarchy between general, and increasingly specific terminology. For instance, at the top of the hierarchy are system organ classes like "Gastrointestinal Disorders", all the way down to phrases like "feeling queasy." The terms at the bottom of this hierarchy, categorized as "lowest level terminology" or *llts*, are representative of the kind of slightly less technical language doctors may use when speaking directly with patients. However, a subset of *llts* are then designated as specialized "preferred terms," or *pts*, which are standardized technical descriptors for a swath of *llts* (each *llt* in the database is associated with a *pt*). The task I am interested in is learning a mapping from *llts* to their most similar *pts*. Put more broadly, I am interested in learning a mapping between medical language that is more lay-interpretable, or even query language generated by patients, and associated *pts*. From a patient side, this could be useful for creating tools for patients to look up their own symptoms. The

terms in MedDRA are used often to encode patient symptoms during drug trials and so one could further imagine a tool like this being useful for analyzing and or regulating these kinds of trials.

The second aspect of my problem is concerned with my use of GNNs for this task, and my exploration of pairing these GNNs with pre-trained transformers. From a certain perspective, GNNs, using convolutional message passing operations, operate very similarly to transformers where edges between neighbors can be thought of as predetermining what the model will attend to when updating embeddings [2]. However, when there are known connects within your data, a GNN may be specially suited to leveraging those rich sources of relational information. In this project I experiment with training GNNs to predict edges between *llts* and *pts*. I explore beginning training with randomly initialized node embeddings, node embeddings drawn from a frozen BlueBERT model, and node embeddings drawn from a BaseBERT model [3, 4]. I further explore 3 different formulations of the graph representation of the MedDRA data - one which is homogeneous, and two which are heterogeneous. Accordingly, I pair these models with heterogeneous GNNs which compute unique message functions for each unique edge type (where I defined edge types by the type of their source and target nodes). My two most distinct findings are that heterogeneous GNNs significantly outperform standard GNNs on this link prediction task, and that the best models are able to achieve close to 70% accuracy in *llt-pt* link prediction regardless of initialization scheme.

3 Related Work

3.1 GNNs

GraphSAGE is an inductive framework for learning low dimensional node embeddings which was developed by members of the SNAP group at Stanford in 2017 [5]. It has since become more or less the standard starting place for GNN construction because of its flexibility in generalizing to unseen data. The same group has also developed a framework for heterogeneous GNNS using GraphSAGE. The group provides a library, DeepSNAP, with code to implement heterogeneous GraphSAGE message passing. I used this library when implementing my model. I will discuss both the GraphSAGE and heterogeneous GraphSAGE algorithms in greater detail in my approach section.

3.2 GNNs Used with Transformers

In 2019, Shang et al. published *Pre-training of Graph Augmented Transformers for Medication Recommendation*, in which they present a hybrid model consisting of a GNN which feeds into a transformer. This group looks at the longitudinal electronic health records (EHRs) for a number of patients across multiple doctor visits, and uses the temporal links between these visits to infer a diagnosis hierarchy from the data. Pointing out that traditional transformers would fail to adequately leverage this information rich hierarchy, they propose $G - BERT$, in which the data, formulated as a graph, is first passed through a GNN to get initial encodings which are then passed to a BERT (Bidirectional Encoder Representations from Transformers) model [6]. Using this model the group was able to achieve state of the art performance on medical recommendation tasks. This paper is of interest for my purposes as it is representative of one of the ways people are already looking at combining GNNs and transformers for medical tasks. Furthermore, its success corroborates my proposal that explicitly leveraging the graph structure of medical ontologies/ hierarchies extracts useful additional information beyond what simple transformers may be able to glean on their own. In its current form our endeavours are not directly comparable as they are trained on different tasks with different datasets, however I think interesting future work could be to train my model - which reverses the order having the transformer feed into the GNN - on a similar EHR dataset.

There are numerous other examples in which groups are combining transformers and GNNs [7, 8]. For instance, Yun et al propose a new Graph Transformer Network which can be used to infer new graph structures directly from data by learning to propose candidate adjacency matrices[8]. This paper emphasis that heterogeneous graph representations are significantly more powerful than homogeneous representations. Furthermore, the success of their work, which allows the automatic generation of heterogeneous graphs from data, is promising from my perspective as it could be used to create new use cases for methods like mine which learn from heterogeneous graphs.

4 Approach

4.1 Data Modeling

MedDRA is organized as hierarchy with 5 levels as shown in 1. At the highest level are system organ classes (*soc*), followed by high level group terms (*hlgt*), then high level terms (*hlt*), preferred terms (*pt*) and finally lowest level terms (*llt*). There are 27 *soc*'s, 337 *hlgt*'s, 1,738 *hlt*'s, and 81,885 *llt*'s, of which *pt*'s are a special subset. (Note that the data is not a perfect tree, LLT's and PT's can be related to one another in the database).

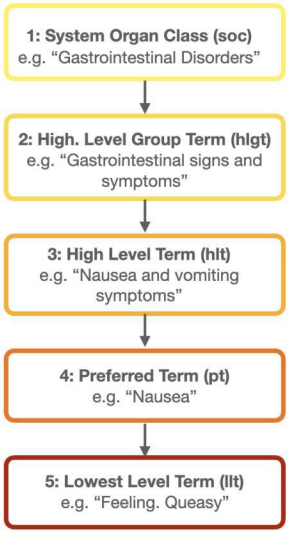


Figure 1: Example of a path in MedDRA.

Using this data, I construct a graph in which each term is a node and edges exist between nodes whenever the terms (which the nodes represent) are associated in the relational tables defined in the MedDRA system. I treat the resulting graph in one of three ways. In the homogeneous case, all nodes are assigned type '0'. In both heterogeneous cases, I treat *pt*'s as distinct from *llt*'s (i.e. if a node is both an *llt* and a *pt*, I drop the former type). In the first heterogeneous case, each node is assigned a type based on its depth in the tree, where *soc*'s are type '1' and *llt*'s are type '5' as labeled in 1. In the second heterogeneous case, nodes of type *soc*, *hlgt*, *hlt* and *pt* are all labeled type '1' and *llt*'s remain as type '5'.

Edges in the each graph are named according to the format 'source-node-type target-node-type.' For example in the first heterogeneous case, there are 8 edge types in the graph: '12', '21', '23', '32', '34', '43', '45', '54'. In the second heterogeneous case, there are 3 edge types: '11', '15', and '51.'

4.2 Model Architecture and Training

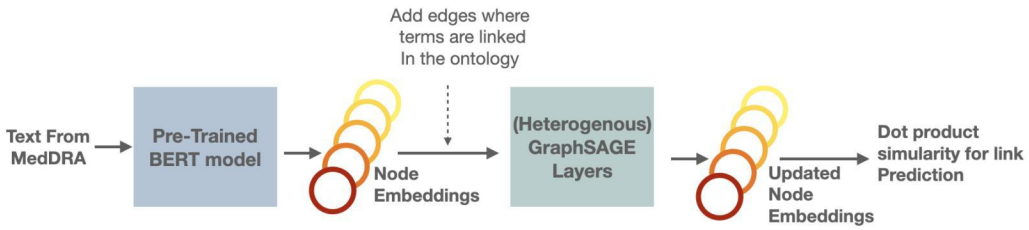


Figure 2: Model Architecture: (In two of the training cases) I use frozen BERT models to get initial node embeddings from text. When the appropriate edges are added, these embeddings are tuned by a series of GraphSAGE message passing layers. I then compute link predictions as described in the section bellow.

I experiment with three different node embedding initialization schemes: 1. Random initialization, 2. passing the associated text through frozen (base) BERT model, 3. passing the associated text through frozen BlueBERT model [3, 4]. BlueBERT is architecturally similar the the base BERT model with the exception that where base BERT is trained on general natural language corpora, the BlueBERT model I use is pre-trained on PubMed abstracts and clinical note (MIMIC-III)[9].. For each embedding initialization scheme, we tokenize the input text using the tokenizer associated with each model, feed the tokens into the corresponding transformer, and extract the [CLS] token embedding as our node embedding. [1]

[1]For each model, I downloaded the frozen weights and tokenizers from Huggingface.

Once the node embeddings have been initialized, and the links between nodes added, I pass the resulting graph through a series of GraphSAGE message passing layers, which operate as follows: For each node v in the set $\mathcal{V}$ of vertices in the graph we denote its features after l rounds of message passing as h_v^l. We denote the neighborhood of v as $\mathcal{N}(v)$, and a set of learnable weights at layer l as W^l. Lastly we define some function AGGREGATE$_l$ which is used to aggregate the embeddings of a given set of nodes at layer l. In this case I use the common mean AGGREGATE function by simply taking the mean of the neighbors' embeddings. Using this terminology we can express the node feature update function at layer k as follows [5]: In the heterogeneous case, the algorithm operates in

Algorithm 1: GraphSAGE embedding forward function

for $v \in \mathcal{V}$ **do**
  $h^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{h_u^{k-1}, u \in \mathcal{N}(v)\})$
  $h_v^k \leftarrow \sigma\big(W^k \cdot \text{CONCAT}(h_v^{k-1}, h^k_{\mathcal{N}(v)})$
end for

much the same way, except that, rather than having one set of weights W^l at layer l, the model will learn a set of weights $W^l_{e1}, ..., W^l_{en}$, where each ei is a distinct edge type in the graph. At each layer, when node v is receiving incoming messages from its neighbors, each message will be transformed by the W_{ei} specified by the edge type connecting the nodes. Thus adding more edge types, adds more expressivity to the model. Thinking about it with the vocabulary of transformer architectures, increasing the specificity of the edge types allows the model to, in essence, attend to neighboring nodes features differently depending on the context of their node type.

The task I train my GNN on is link prediction. In order to do this before training begins, the edges in the graph are split into train and validation sets. During training, the edges in the train set are masked out leaving it to the model to predict their existence. The model is evaluated on the complementary set for validation. Additionally a series of negative samples are tested on in each case - i.e. the model is asked to predict the non-existence of a set of edges as well. (Note that the plotted accuracies in the following section reflect the accuracy in predicting both the positive and negative samples.)

The existence (or otherwise) of an edge is established by normalizing over the dot products of node embeddings. More concretely, we define a series of edges with a 2 by m matrix E, where m is the number of edges we are testing on. (Note that in the heterogeneous case we have a distinct E for each edge type). Each column of E defines one edge; the first row designates source nodes while the second row designates target nodes. For each source target pair, we take the dot product of their associate node features, then normalize over the the whole set of dot products using sigmoid function σ. Finally we threshold at 0.5, and for all pairs whose normalized dot product is greater than 0.5, we output 1 - i.e we predict that the edge exists, and in all other cases we output 0 - i.e. we predict that the edge does not exist. We then can use binary cross entropy loss to compare the model predictions with ground truth labels. [2]

In essence, the loss function I use is binary cross entropy loss with logit loss (BCEWITHLOGITSLOSS). For batch size N, BCEWITHLOGITSLOSS can be expressed as follows:[5, 10]

$$\ell(x, y) = L = \{l_1, ..., l_N\}^T, \tag{1}$$

$$l_n = -w_n[y_n \dot{l}og\sigma(x_n) + (1 - y_n)\dot{l}og(1 - \sigma(x_n))] \tag{2}$$

$$\ell(x, y) = mean(L) \tag{3}$$

One benefit of this kind of analysis, is that this normalized dot product similarity test can be used on text embeddings even if they are not drawn from a graph structure. This allows us to perform a baseline by taking dot products between the node embeddings returned by each of the frozen BERT models. It turns out that this is a very weak baseline, as, for both base BERT and BlueBERT, the models achieve only about 5% accuracy in predicting *llt - pt* links. A similar accuracy is achieved if one uses nearest neighbors to propose links between *llt's* and the *pt's* with the minimal euclidean distance from one another in the BERT models' embedding spaces.

[2] My code for this is inspired by the DeepSNAP link prediction tutorial.

5 Experiments

5.1 Evaluation method

My primary metric when comparing models is their accuracy in predicting *llt - pt* connections in a held out validation set of edges. We define p to be the number of positive connects the model predicts (i.e. the number of connections it predicts exist) and we define n to be the number of negative connections the model predicts (i.e. the number of times the model predicts no edge exists between two nodes). In order to ensure computational tractability, we limit n to be a set of randomly sampled negatives. The number of negatives is held constant across all experiments. We then let p_c and n_c be the number of each prediction type which is correct. Expressed in these terms we formally describe accuracy as follows:

$$accuracy = \frac{p_c + n_c}{p + n} \tag{4}$$

I also look into finer grained analysis of the predictions: the *true positive rate* given by $\frac{p_c}{p}$, the *true negative rate* , $\frac{n_c}{n}$, the *false positive rate*, $\frac{p-p_c}{p}$ and the *false negative rate*, $\frac{n-n_c}{n}$. Lastly I look at *positive accuracy* - the percent of true positive edges the model accurately predicts -and *negative accuracy* - the percent of true negative edges the model accurately predicts.

5.2 Experimental details

I used the deepsnap framework to create a transductive split of edges in the input graphs (i.e. to randomly mask out some edges to predicted in train and val respectively). I then train my models on the link prediction task defined in the previous section. I ran a number of experiments in which I varied the following variables in a controlled manner: **embedding initialization**: random, BlueBERT or base BERT - **graph type**: homogeneous or one of two heterogeneous configurations - **dropout**: 0.0 or 0.5 - **learning rate**: 0.01, 0.001, 0.0001 - **weight decay**: 0.0 or $5e-4$ - **use of a scheduler** for learning rate annealing: yes, no - **activation**: leakyrelu, relu, elu or tanh - **hidden size**: 12, 32 or 64 or 128, and **number of message passing layers**: 2-5.

5.3 Results

Firstly, I found that across hyper parameter values, the results on regarding graph type were fairly consistent. Recalling that the three options were: homogeneous - in which all nodes were assigned the same type, heterogeneous 1 - in which each type in the MedDRA hierarchy was treated as a distinct node type, and heterogeneous 2 - in which all nodes were assigned to one of two node types - *llt* and not *llt*, the results can be summarized as follows:

Best and average accuracy: initialization schemes

Type	Average Accuracy	Accuracy Upper bound
homogeneous	45%	50%
heterogeneous 1	60%	66%
heterogeneous 2	64%	69%

Another result that was fairly consistent is the need for non-zero dropout probability; I found that the model consistently overfit if dropout was not used. The results regarding learning rates were likewise fairly stable, suggesting that with or without the use of a scheduler 0.001 was the sweet spot.

The effects of changing the hidden size varied most notably with model initialization, though a hidden size of 32 was consistently the highest achiever when using non-Random initialization. (Note that in these and the results which follow I report results for the second heterogeneous graph construction as it was likewise the most successful.)

Best accuracy: initialization scheme vs. hidden size

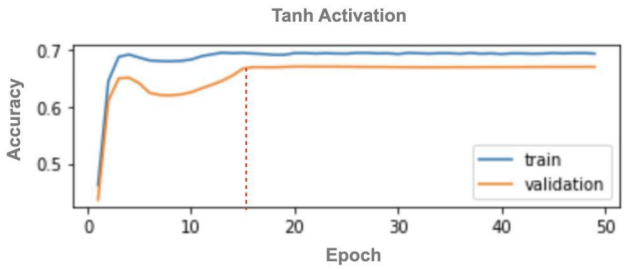
	12	32	64	128
BlueBERT	57%	69%	64%	50%
base BERT	41%	68%	66%	51%

Similarly for number of message passing (MP) layers, for both non-random initialization schemes I found correlated dependencies; both performed best with 3 MP layers.

Best accuracy: initialization scheme vs. number of MP layers

	2	3	4	5
BlueBERT	66%	69%	64%	61%
base BERT	65%	68%	62%	66%

Furthermore, I found that for all initialization schemes, using a tanh activation between layers consistently either achieved the highest, or tied for the highest accuracy achieved by each scheme, with LeakyReLU as the second most reliable performer. Interestingly though, using tanh also greatly speed up training. The plot bellow, showing accuracy per epoch, is characteristic of the speed and directness with which models using tanh consistently converged.



I left the case that initialized node embeddings randomly out of the last few tables as it behaved somewhat uniquely. Firstly, I found that training the same model parameters, but with different random initializations could lead to a spread of about 13% in accuracy. For instance, in one trial with hidden size 12 and 3 MP layers, the model never achieved higher than 53% accuracy, while in a subsequent trial that spiked to over 68%, a score which rivals the best performance by any model across initialization schemes. However, if either the hidden size was pushed to 128, or tanh was used, I found that the models trained on randomly initialized embeddings could achieve much more stable performance, consistently achieving accuracies in the range $64\% - 68\%$.

6 Analysis

That the second heterogeneous graph formulation is the most successful seems to me to indicate two general points. Firstly, that the heterogeneous cases in general were the most successful, corroborates the intuition that the extra expressivity gained from learning multiple message functions significantly increases a models ability to learn from relational data. Furthermore, that the second heterogeneous formulation outperformed the first one also seems intuitive when one considers that there are over 80k connections between *llt*'s and *pt*'s while there are less than 500 connections between $soc's$ and $hlgt$. It is perhaps unsurprising that the utility of learning a distinct message function for a given edge type is inversely proportional to the number of edges of that type in your dataset.

Ultimately, models using each of the three initialization schemes were able to achieve similar maximal performances - about 68% accuracy. Considering that the baseline accuracy coming from the frozen BERT models was only 5%, this this is an exciting result. Furthermore, that the randomly initialized models were able to do comparably well to the others is in and of its self a surprising and exciting result. Compared to many NLP corpora, the dataset I used was fairly small. Thus it is encouraging to see that, leveraging no other pre-training, the GNN was able to extract significant amounts of information about the relationships between terms in the ontology.

On the other hand, initializing the node embeddings using a pre-trained BERT model lead to increased stability and reliability during training. It also gave results that were much more interpretable; while it was harder to glean a pattern from looking at the predictions of the models using random initialization, there are clear trends in the model's predictive abilities when paired with BERT. In particular, both models of this type were able to predict the connections between phrases with shared words. For instance, both models would successfully predict that the *llt* "Infusion site reactions" ought to be

connected to the *pt* “Infusion site urticaria." On the other hand pairings like “’Foreign body in throat" and “Chest and respiratory tract injuries NEC" where harder for the models to predict.

7 Conclusion

I was able to achieve close to 70% accuracy in predicting the connections between *llts* and *pts* in the MedDRA database. I found that this predictive ability was largely invariant to the node embedding initialization scheme used. I believe this is indicative of the richness of the information that can be gleaned from the structure of hierarchical data alone. This makes me think that involving both transformer and GNNs in a more end to end training scheme could be very promising.

Linking the GNN and transformers however is a non trivial task. Indeed one limitation of the work I have done here is that it is not immediately obvious of the models I have trained could be used to integrate new nodes into the graph. Looking more specifically into this problem would be an interesting next step. Furthermore, there are many other, both graph and medical NLP specific tasks to which I would like to try to apply this kind of heterogeneous GNN.

References

[1] Elliot Brown. *Medical Dictionary for Regulatory Activities (MedDRA®)*. John Wiley Sons, Ltd, 2006.

[2] Chaitanya Joshi. Transformers are graph neural networks, 2020.

[3] Yifan Peng, Shankai Yan, and Zhiyong Lu. Transfer learning in biomedical natural language processing: An evaluation of bert and elmo on ten benchmarking datasets, 2019.

[4] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface’s transformers: State-of-the-art natural language processing, 2020.

[5] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[6] Junyuan Shang, Tengfei Ma, Cao Xiao, and Jimeng Sun. Pre-training of graph augmented transformers for medication recommendation, 2019.

[7] Edward Choi, Zhen Xu, Yujia Li, Michael W. Dusenberry, Gerardo Flores, Yuan Xue, and Andrew M. Dai. Learning the graphical structure of electronic health records with graph convolutional transformer, 2020.

[8] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks, 2020.

[9] Yifan Peng, Shankai Yan, and Zhiyong Lu. Transfer learning in biomedical natural language processing: An evaluation of bert and elmo on ten benchmarking datasets, 2019.

[10] Pytorch documentation of torch.nn.bcewithlogitsloss.