A Survey of Graph Neural Network Approaches to Electronic Health Record Data

David Pitt
Department of Mathematics
Harvey Mudd College
Claremont, CA
dpitt@hmc.edu

Abstract

Graph neural networks (GNNs) have shown promising results for signal extraction tasks on electronic health records. In this paper, I present recent advances in GNN techniques and apply GraphSMOTE, a state-of-the-art GNN framework, to mortality prediction on the eICU Collaborative Research Database. I then present a novel adaptation of GraphSMOTE and review its performance on the same dataset.

1 Background

Almost every event that occurs during an interaction between a patient and a doctor is stored in an electronic health record (EHR). EHR data contains a wealth of information in various formats, from lab results to diagnosis codes to doctors' notes. Data mining techniques can leverage the interconnected nature of this data to create a graph representation of the EHR. Given such a graph, the past four years have seen the development of a variety of machine learning techniques to extract patterns and make predictions at the node level. Recently, the approach of learning representations of medical concepts and events has demonstrated success in tasks such as predicting Alzheimer's diagnoses and patient mortality. In the following section, I will present some recent graph machine learning approaches to EHR problems and compare their performance on the eICU-CRD dataset.

2 Related Work

This project is motivated by recent work in the area of applying graph neural networks to electronic health record data.

To define the problem, consider a set of elements $\mathcal{V}$, each of which is represented by a feature vector v_i. Based on some relationship, the elements are connected by edges from the set $\mathcal{E}$. We can think of a graph as an object containing the set of all our vertices and edges: $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. Graph models are useful for describing many kinds of interconnected systems, from social media networks to power grids. As such, machine learning algorithms for prediction and classification on graph-structured data has the potential to yield invaluable insights across a variety of fields.

Over the past half-decade, the field of machine learning on graphs has seen tremendous growth, as both generalizations of existing ML algorithms and new algorithms designed to leverage the properties of graphs have demonstrated efficacy at classification and prediction tasks.

Such classification and prediction tasks take the form of edge, node and graph-level tasks. Node classification seeks to identify the class of one node in a network. Edge classification aims to characterize a relationship between two nodes. Graph classification places an entire graph into a category - for instance, classifying a molecule based on the features of its "nodes" (atoms) and "edges" (bonds).

Graph neural networks (GNNs) are an application of neural networks to data with a graph structure. All of the GNNs I will discuss in this paper follow the same basic principle. The networks learn to represent each node by an embedding in a low-dimensional Euclidean space, and over time apply transformations to node embeddings that incorporate information from neighbors. The embeddings are then used to classify (through clustering or some other means) elements of the graph at the node or edge level.

The graph convolutional network (GCN) is a GNN algorithm that generalizes the CNN architecture to data without a grid structure. Over time, representations come to reflect information from neighbors with larger graph distances. The final embeddings are often used to cluster nodes into classes.

As a baseline, consider a simple GCN. Given an adjacency matrix, the network learns representations of the features associated with each visit, then learns an embedding of each visit. The network then uses this embedding to classify by treatment or diagnosis.

While this method is effective, more complex architectures are able to take advantage of data that varies in time, as well as generalizing to a broader set of problems. Electronic health records contain many modes of data and exhibit complex information structures, so they make excellent datasets for testing new approaches to subproblems in graph machine learning. In the following section, I present eight models from recent literature and highlight the information-theoretic mathematical advances of each.

[1] (HORDE) approaches the problem by constructing a multi-modal EHR graph, wherein the graph contains multiple classes of node: patient, event, and concept. The graph is structured like a semantic knowledge graph, with the exception of patients. Patients are connected to concepts and events that relate to their hospital visits. The edges that connect patient nodes to the rest of the graph vary over timesteps, corresponding to a sequence of visits.

Each edge is also assigned a numeric weight. To calculate this weight for concept nodes, the authors measure the importance of connections between two concepts. They do so with information from the MIMIC-III dataset, using sliding window analysis on unstructured natural language data (doctors' notes) to detect co-occurrence of the two concepts. The resulting weight is proportional to the conditional probability of a mention of one concept in natural language notes given a mention of the other.

To calculate embeddings of time-invariant nodes (events and concepts), the authors use a simple graph convolution. For the l-th layer of the network, this is represented by the equation:

$$h_i^l = \varphi\left(\sum_{v_j \in \mathcal{V}_{inv}} \frac{h_j^{l-1} W_{inv}^l}{|N(v_i)| \cdot |N(v_j)|}\right),$$

where $\mathcal{V}_{inv}$ represents the set of all invariant nodes and φ is a nonlinear activation function.

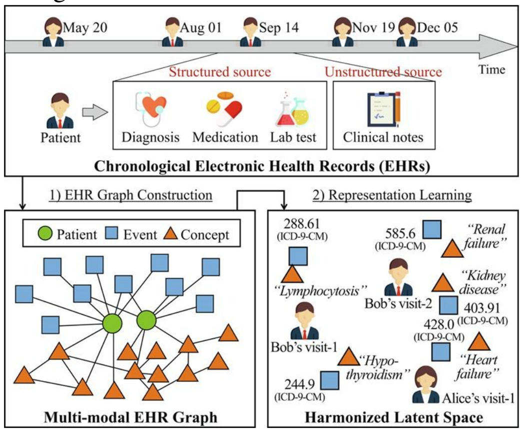
For time-variant nodes, the computation is essentially the same, but the embeddings are recomputed for the graph at each timestep. The full sequence of embeddings of those nodes are then passed through an LSTM to produce the final embedding for time-variant nodes.

To compute loss for the model, the authors define separate cost functions for the time-variant and time-invariant nodes, and take the total loss function to be the sum of the two. The loss here is defined as the central moment of a negative sampling distribution.

After tuning, the authors use this model to classify patient severity and mortality on the MIMIC-III dataset.

One problem plaguing many graph datasets is class imbalance. MPNN classifiers tend to perform pooly when one class of node is heavily underrepresented. [2] (LSTM-GNN) attempts a solution. To account for the fact that the majority of diseases and procedures occur relatively infrequently, the authors apply reasoning similar to that used by doctors when diagnosing patients. They do so by considering classification on the "case" level: they construct a patient graph with adjacency matrix $\mathcal{M}$, where edges are weighted by the relatedness of the patients' diagnoses. Relatedness score is calculated via a weighted sum of shared diagnoses between each pair:

Figure 1: Illustration of the HORDE model



$$\mathcal{M}_{ij} = a\sum_{\mu=1}^{m}(\mathcal{D}_{i\mu}\mathcal{D}_{j\mu}(d_\mu^{-1}+c)) - \sum_{\mu=1}^{m}(\mathcal{D}_{i\mu}+\mathcal{D}_{j\mu}),$$

where $\mathcal{D} \in \mathbb{R}^{N\times m}$ is a diagnosis matrix, where N is the number of patients and m is the number of diseases present in the dataset. Weights are inversely proportional to the rarity of a shared diagnosis, which implies for the model that two patients who share a rare diagnosis are probably more similar. $\mathcal{M}$ is filtered using the k best neighbors for each node to establish a fixed neighborhood size. Note that k, c, and a are hyperparameters that the authors later tune.

Each node (patient) has time-series features and static features. The time-series features are fed through an LSTM, and static features through a traditional GCN. They then concatenate their generated embeddings and feed the combined result through a dense layer. The cost function is a combination of loss after passing the outputs of each component through dense layers and the loss after the final combined output is passed through a dense layer. This model is used to predict in-hospital mortality and length of stay.

Sometimes, the adjacency matrix for a particular network is not known a priori. [3] (GCT) is a solution to this type of graph classification task. Instead of working on a predetermined EHR graph, the authors use a self-attention mechanism to determine which connections within the EHR graph are relevant. They do so by combining the transformer architecture (previously used in sequential data analysis, e.g. sentence translation) with a graph structure to weight edges according to the attention calculated from the concatenated embeddings of two corresponding nodes.

The algorithm is initialized with a matrix P that represents the conditional probability of one event given an occurrence of the other. This matrix is constructed through simple statistical analysis of the frequency and co-occurrence of features in the dataset. The authors also introduce a mask matrix M that removes forbidden connections from the adjacency matrix.

From there, attention is used to reconstruct the adjacency matrix. For the j-th block of the network:

$$\hat{A}^{(j)} = \text{softmax}\,(\tfrac{Q^{(j)}K^{(j)T}}{\sqrt{d}} + M),$$

where $Q^{(j)} = C^{(j-1)}W_Q^{(j)}$, $K = C^{(j-1)}W_K^{(j)}$, C represents the output of an encoder with single-head attention, and $W^{(j)}$ are trainable weight matrices for the block.

Then, the output of the block is calculated by:

$$C^{(j)} = MLP(PC^{(j-1)}W_V^{(j)}) \text{ when } j = 1,$$

$$\text{and } C^{(j)} = MLP(\hat{A}^{(j)}C^{(j-1)}W_V^{(j)}) \text{ for all following blocks.}$$

For the loss function, the authors use KL-divergence between one block's adjacency matrix and the previous adjacency matrix as a regularization term to penalize less general adjacency matrices, and add this to a prediction loss term (e.g. BCE) for total cost.

This method results in a generalization of the GCN to graphs in which the adjacency matrix is unknown. Given a bag of nodes, the model constructs an adjacency matrix and applies simple graph convolutions to the resulting graph to create node embeddings, which are then used to predict heart failure and ICU admission via binary classification.

Many graph models also lack generality, and exhibit discrepancies between training and testing performance. [4] (VGNN) uses a generative element to resolve these issues. This model is an extension of the graph autoencoder. It takes in a node feature matrix X, and first creates embedding matrix H using a lookup algorithm. As this occurs, a K-head attention module predicts an adjacency matrix $\widehat{A}$. Attention coefficients are calculated as:

$$e_{ij} = \sigma(a^T[Wh_i \mid\mid Wh_j])/\sqrt{dim(h_i)}.$$

Then, $\widehat{A}$ is calculated by:

$$\widehat{A}_{ij} = \frac{exp(e_{ij})}{\sum\limits_{p \in N(i)} exp(e_{ip})},$$

and the embeddings of the encoded graph at each layer are calculated by:

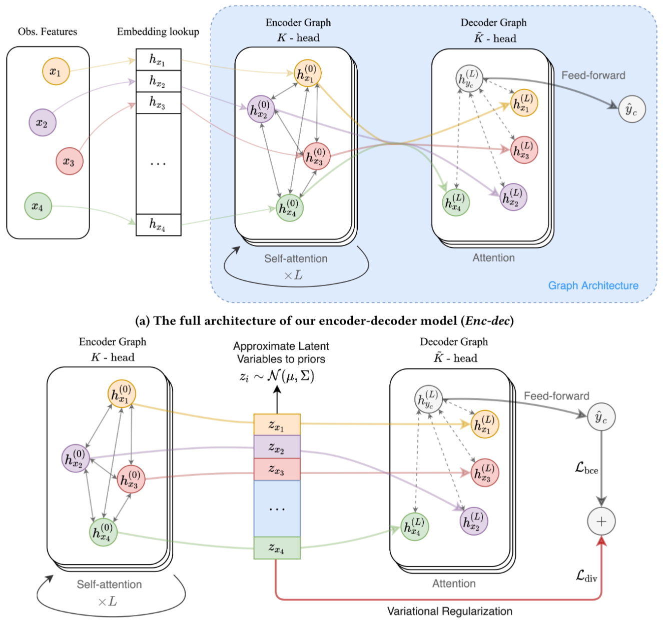
$$H^{(l+1)} = \text{FFN}\ [\widehat{A}^{(l)}(H^{(l)}W^{(l)} + b^{(l)})],$$

where W, b form a linear layer.

The output of the encoder module is a graph with nodes embedded a latent space, where new embeddings are represented by Z. From there, the model assumes that latent variables are distributed according to a Gaussian prior, $p(z_i \mid h_i)$. From there, the model learns a distribution $q(\hat{h_i} \mid z_i)$, and resamples from this distribution to obtain new node embeddings. The h_i are then fed to a linear layer to predict y_i.

To penalize over-specificity in the distribution $q(\hat{h_i} \mid z_i)$, the authors use KL-divergence as a regularization term in the cost function. This is combined with cross-entropy loss to train the model. They then apply this model to mortality and Alzheimer's diagnosis tasks.

Figure 2: Diagram of the VGNN model



Obs. Features
Embedding lookup
Encoder Graph
K - head
Decoder Graph
$\tilde{K}$ - head
Feed-forward
Self-attention
$\times L$
Attention
Graph Architecture
(a) The full architecture of our encoder-decoder model (*Enc-dec*)
Approximate Latent Variables to priors
$z_i \sim \mathcal{N}(\mu, \Sigma)$
$\mathcal{L}_{\text{bce}}$
$\mathcal{L}_{\text{div}}$
Variational Regularization

Some models structure certain information more explicitly in order to mimic human knowledge-based approaches to classification (in the case of EHR data, diagnosis). [5] (GRAM) is another self-attention representation learning model. However, instead of one undirected graph, this model constructs a directed acyclic graph (DAG) to encode the hierarchical nature of "ontology information" (concept classes and definitions). This "knowledge DAG" has a parent-child structure, where lower levels represent higher specificity. Each concept node is given an initial embedding, and embeddings are updated using attention coefficients and the embeddings of parent nodes. Attention coefficients are calculated using softmax operating on a "compatibility function" that represents the output of an MLP on the basic embeddings of a parent and child node.

From there, the authors construct an embedding matrix for all the codes in the knowledge DAG, which is then multiplied by a multi-hot vector for each visit to create a visit embedding consisting of only the revelant codes for that visit. Visit embeddings are then passed through an RNN, and the final hidden state is passed through a softmax layer to predict the disease codes of the next visit.

[6] also approaches the class imbalance problem. The authors are chiefly concerned with classification in situations where one node class is much less present in the dataset than others. This often causes classifiers to underperform on the underrepresented classes in the dataset. To solve this problem, the GraphSMOTE algorithm creates synthetic nodes from the minority class, predicts edges for those nodes, and performs classification on the augmented balanced graph. The model is comprised of four main components: an encoder, a synthetic node generator, an edge generator and a classifier.

The encoder block of GraphSMOTE is a GraphSage encoder block. To avoid potential overfitting problems, the authors use only one GraphSage layer. A forward pass through this layer represents a nonlinear activation of the node's features concatenated with its edges multiplied by weights. From there, after embedding each node, the model generates new nodes as samples from the minority classs. They do so using the pre-existing SMOTE algorithm, which interpolates nodes from the minority class with synthetic nodes between the target nodes and their nearest neighbors (in the latent space) in the minority class. First define the nearest neighbor of a node as

$$nn(v) = \text{argmin}_u \; ||h_u^1 - h_v^1|| \text{ s.t. } Y_u = Y_v,$$

for node $v \in \mathcal{V}$. Each new sample is a linear combination of a target node and its nearest neighbor:

$$h_{v'}^1 = (1 - \delta) \cdot h_v^1 + \delta \cdot h_{nn}^1,$$

where $\delta \sim$ Uniform(0,1). In cases with more than one minority class, SMOTE is applied to every minority class, and a hyperparameter is used to determine the number of synthetic nodes generated.

Once we have generated synthetic nodes in embedding space, the new nodes are attached to the raw graph $\mathcal{G}$. To do so, the authors train an edge generator model on $\mathcal{G}$ to predict which pairs of nodes are connected, and use this generator to predict edges between every pair of nodes in the graph. The edge generator is defined below as

$$E(u, v) = \text{softmax}(\sigma(h_v^1 \cdot S \cdot h_u^1)),$$

where σ is a nonlinear activation, and S is the parameter matrix that encodes node relatedness. This generator is trained by re-generating A from scratch, and the loss for the generator is defined as

$$\mathcal{L}_e = ||E - A||_F^2$$

Using the trained edge generator, the authors then calculate the edges for each synthetic node. There are two strategies for adding the edges to A: first, edges could be added with a value of 1 if the edge generator returns above a certain threshold value; second, edges could be added with weights taking on any value on [0,1].

The new node representations and edges are then added to $\tilde{H}$, the set of all node representations, and the adjacency matrix A to create a new balanced augmented graph $\tilde{\mathcal{G}}$. This augmented graph is then used to train a GNN-based classifier.

The GNN classifier takes $\{\tilde{H}, A, \tilde{\mathcal{V}}\}$ as input. In this implementation, the authors pass the inputs through a GraphSAGE classifier block:

$$h_v^2 = \sigma(W^2 \cdot CONCAT(h_v^1, \tilde{H}^1 \cdot \tilde{A}[:, v]))$$

The outputs of this step $\tilde{H}^2$ are passed into a linear layer to predict the final class of each node in the balanced graph:

$$P_v = \text{softmax}(\sigma(W^c \cdot CONCAT(h_v^2, \tilde{H}^2 \cdot \tilde{A}[:, v]))),$$
$$\hat{Y}_v = \text{argmax}_c P_{v,c}$$

where W represents weight parameters, $\tilde{A}[:, v]$ represents all edges connected to node v in the augmented graph $\mathcal{G}$, and P_v represents the normalized vector of probabilities that node v belongs to each class.

The authors use standard cross-entropy loss to optimize the GNN classifier.

When all the blocks are in sequence, the final cost function for the model is expressed as

$$\min_{\theta,\phi,\psi} \mathcal{L}_{node} + \lambda \cdot \mathcal{L}_{edge},$$

for parameters of the encoder, edge generator and GNN classifier θ, ϕ, ψ.

While some graphs suffer from too little data, others suffer from fraudulent or erroneous entries. EHR data is estimated to contain 20% erroneous information. [7] introduces a method for learning which nodes are most likely fraudulent and deleting them from a graph dataset to improve predictive accuracy.

To start, the model constructs a more complicated graph that encodes multiple types of relationships between nodes. The authors define a multi-relational graph as the set $\mathcal{G} = \{\mathcal{V}, \mathcal{X}, \{\mathcal{E}_r\} \mid_{r=1}^{R}, Y\}$, where $\mathcal{X}$ represents the matrix of feature vectors, $\mathcal{E}_i$ represents the set of all edges of relation type $r = i$, and Y represents the class of each node.

To model the extra domain knowledge used by humans in identifying financial fraud and diagnosing diseases, the authors introduce a similarity measure between nodes $\mathcal{S}(u, v)$ at each network layer to ensure that nodes with similar labels are similar in embedding space:

$$\mathcal{S}(u, v) = 1 - \mathcal{D}(u, v),$$

where $\mathcal{D}(u, v)$ is the scaled l^1 embedding distance between two node representations. The results of the label-aware similarity function are also into the network's final loss function to penalize dissimilar embeddings among a class of nodes.

Each layer in the model constitutes one Recursive and Scalable Reinforcement Learning module, or RSRL. The RSRL module's goal is to learn to reconstruct the graph such that it contains as little erroneous/fraudulent data as possible. It does so through a Markov Decision process, where the action space represents actions on the input graph (such as node deletion) and the reward function for MDP for each class represents the average similarity $\mathcal{S}(u, v)$ between nodes of the class.

After pruning erroneous nodes, the model aggregates information from each node's relevant neighbors. To select relevant neighbors of node v_i, the model picks the nodes with similarity above a threshold to aggregate information. From there, the learning process proceeds as in an MPNN to classify each vertex. This model is applied to several datasets, including an EHR diagnosis task.

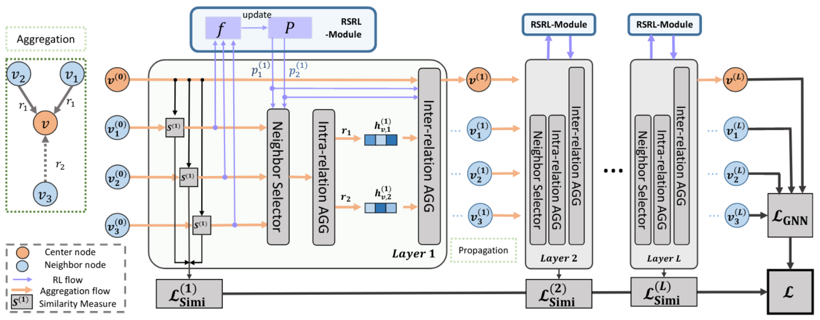
Thus far, all the models presented have been simple MPNNs. However, some models take different approaches to modeling the flow of information.

The key advance of [8] is in its application to recommender systems on EHR graphs. Instead of using message-passing neural network (MPNN) layers to propagate information through the graph, the authors construct a mathematical object (agent) to walk along the graph to simulate the flow of information. The model is trained using a reinforcement-learning approach, and over time learns to walk across the graph to find the most likely disease nodes associated with each patient.

Procedurally, the authors define a Markov Decision Process with state space $s_t = \{p_e, e_t, h_t\}$, where p_e represents the agent$'$s initial patient representation, e_t represents the current representation at time t, and h_t represents the historical trajectory of the agent. The authors use a fully-connected layer to learn a representation of each state:

$$x_t = \sigma(\sigma(x_{t-1})W_t + b_t),$$

Figure 3: An overview of the RioGNN model



where W_t and b_t constitute the parameters of a linear layer.

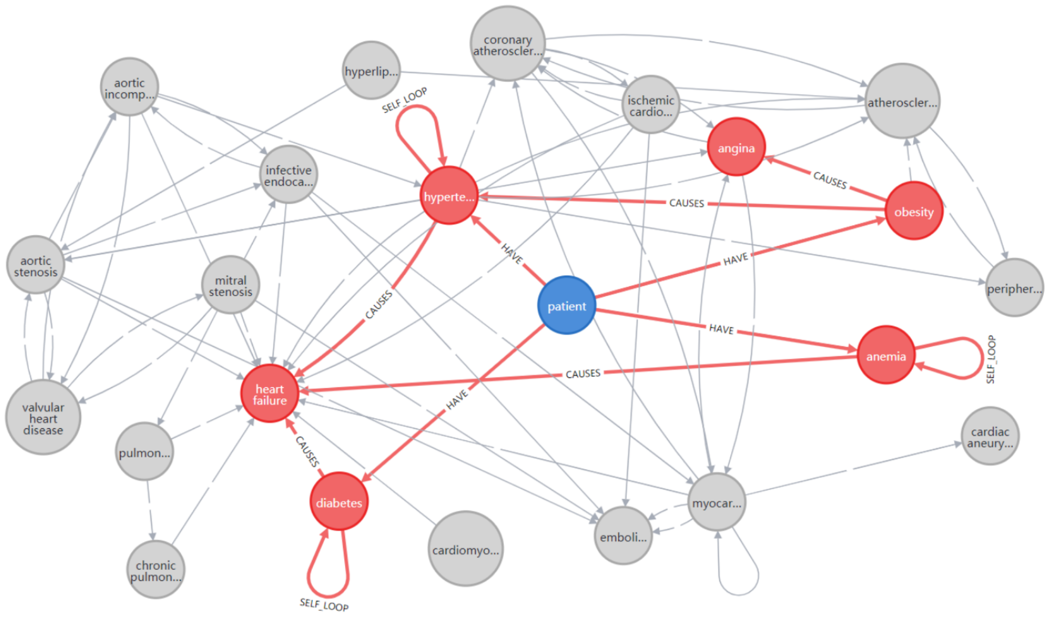
The model then constructs an agent with an initial state $s_0 = \{p_e, p_e, h_0\}$ (such that it is initialized at a patient node). The action space $\mathcal{A}$ contains actions $a_t = (r_{t+1}, e_{t+1})$ (walking to a new node). Over time, the agent learns the optimal policy π_t:

$$\pi_t = \text{softmax}\left((x_t W_p + b_p) \bigodot \mathcal{A}_t\right),$$

where $\bigodot$ represents the Hadamard product.

Once trained, the model is used for inference on new patient entities, such that implementing π_t will generate a walk that likely contains disease nodes, representing the model's diagnosis of the patient.

Figure 4: An illustration of the knowledge graph



3 Experiments

In this section, I present my findings on applying these models to data and evaluating their performance on a standard classification task.

3.1 Data

I use the eICU Collaborative Research Database, a database that stores deidentified, high-granularity information about over 200,000 ICU stays from electronically-recorded health systems across the US. The dataset includes, among other things, diagnoses, lab results, treatment plans and vital sign measurements. This dataset builds on the success of the MIMIC-III database and expands it to include information from many hospitals (as opposed to just Beth Israel in Boston, MA). In total, the database contains 139,367 unique patients admitted to one of 335 units at 208 hospitals in the US. Each patient, each hospital visit and each care unit stay are associated with a unique integer. The schema of the database is denormalized, meaning that each table (corresponding to one piece of information - for instance, lab events) can be accessed independently. The models in this literature review use information from four key tables: patients, admissions, diagnoses, and treatment. Admissions contains information about each time a patient was admitted to an ICU unit, Diagnoses contains every diagnosis event for each patient in the patients table, and Treatments contains every treatment prescribed to a patient in the table.

For this paper, I will be using the models to predict in-hospital mortality. To do so, I only use information from the first 24 hours of each care unit stay to predict eventual death in a care unit. A patient can be admitted multiple times within the dataset, so I use the discharge flag associated with each admission (which takes on values including death and improved condition) to classify each admission.

3.2 Results

In this section, I standardize the models discussed above and evaluate their performance on predicting in-house mortality.

To evaluate the performance of each model, I will use accuracy under the precision-recall curve. AUPRC is a useful metric in the context of medical data analysis because the long-term objective of such data analysis is to identify those patients most likely in need of extra attention. A model that correctly identifies all patients likely to suffer adverse health effects and also produces some false positives is a more useful model than one that produces no false positives but some false negatives. Here, precision is the proportion of true positives out of all identified positives, and recall is the proportion of true positives out of all true cases.

Below are results I have obtained in a standardized environment (Python 3.8, Tensorflow 2.0, Pytorch 1.15, NVIDIA RTX 2070 Super w/CUDA).

Table 1: Model evaluation on the eICU IHM task

Model	AUPRC
HORDE [1]	0.433
LSTM-GNN [2]	0.386
GCT [3]	0.581
VGNN [4]	0.603
GRAM [5]	0.575
RioGNN [7]	0.572*

**Provided their own MIMIC matrix, not standard dataset*

4 Next steps

In the upcoming term, I plan to develop a model that synthesizes the approaches of [6] and [7].

These models are cutting-edge, novel solutions to pressing problems in the field of graph machine learning. However, while innovative, they lack generality. My goal is to synthesize these models to create a more general framework for neighborhood selection. I believe that a model that can learn to selectively rebalance a graph dataset through pruning and edge interpolation would yield significant advancement in EHR diagnosis tasks. If successful, my new model will have broad applicability.

As an overview, I will develop a model that uses RL to selectively delete nodes and add edges to increase the predictive accuracy of the walking agent from [8]. To develop this model, I will need to work on a few key mathematical pieces. Most importantly, RioGNN's reinforcement learning approach succeeds because the MDP's reward function can inform how the network's predictive accuracy is liable to change when a node is deleted. The function thus serves as an efficient proxy for retraining the entire network with every action in the MDP. This function depends on a label-aware similarity measure:

$$\mathcal{S}(u, v) = 1 - \mathcal{D}(u, v),$$

where $\mathcal{D}(u, v)$ is the scaled embedding distance between two node representations. The reward function then takes the average similarity of nodes in a neighborhood to evaluate a certain action on the local graph. Conversely, GraphSMOTE uses a probabilistic model to predict the presence of an edge between two nodes:

$$E(u, v) = \text{softmax}(\sigma(h_v^1 \cdot S \cdot h_u^1)),$$

where σ is a nonlinear activation, S is the parameter matrix that encodes node relatedness, and h_i is the emedding of node i. Here, the training loss is $\mathcal{L}_{edge} = ||A - \widehat{A}||$, where $\widehat{A}$ is the predicted reconstructed adjacency matrix.

My hypothesis is that for diagnosis tasks on EHR data, the presence of an edge is less important than the preservation of the approximate geometry of the network. The success of [8] supports this hypothesis. To combine walk-reachability and graph alteration, I propose a similarity function based on p-step reachability, where $\mathcal{L}_{edge} = ||R_p - \widehat{R_p}||$, where $\widehat{R_p}$ is the p-step reachability matrix, and p is a hyperparameter to be tuned. This would reward the graph reconstructions with the most faithful path structures.

I will then increase the MDP's action space to include edge interpolation, and construct a reward function that takes the average similarity $\mathcal{S}(u, v)$ between nodes in a p-step neighborhood. A successful action in the MDP would create a shorter path between a patient node and a diagnosis that he or she is likely to receive in the future.

I have collaborated with Prof. Gu this semester on a comprehensive review of applications of graph machine learning to EHR data, and as a next step, I would like to pursue the development of this model. She has agreed to continue as my research mentor. With my investment in background research, I believe I have created a strong foundation this project.

I propose to use HMC research funding to increase my computing resources. Right now, I am running my models in a standard environment (Py3.8, TF2.7, Pytorch 1.15, CUDA 11.4) on a local machine with 8GB of VRAM. To run models more efficiently and tune hyperparameters faster, I would like to purchase time on a GPU cloud.

Based on the success of the previous approaches I have presented, the clear direction for mathematically expanding these models, and the potential for significant positive impact, I believe that my project is both viable and valuable. Moreover, I believe that the goal of constructing functions to act as a proxy for the network's predictive accuracy has the potential to provide new insight on the mathematical foundations of machine learning, an incredibly important and fast-growing field.

Acknowledgments

I am incredibly grateful for Prof. Gu's for help in identifying literature and consulting on methods. I would also like to thank her and Nat for providing relevant instruction throughout the semester in Math189.

References

[1] Harmonized representation learning on dynamic EHR graphs https://www.sciencedirect.com/science/article/pii/S153204642030054X?viaCode: https://github.com/donalee/HORDE or https://github.com/Lishany/HORDE-pytorch

[2] Predicting patient outcomes with representation learning https://paperswithcode.com/paper/predicting-patient-outcomes-with-graph

[3] Learning the Graphical Structure of Electronic Health Records with Graph Convolutional Transformer https://paperswithcode.com/paper/graph-convolutional-transformer-learning-the

[4] Variationally Regularized Graph-based representation learning for electronic health records https://paperswithcode.com/paper/graph-neural-network-on-electronic-health

[5] GRAM: Graph-based attention model for healthcare representation learning https://paperswithcode.com/paper/gram-graph-based-attention-model-for

[6] GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks https://github.com/TianxiangZhao/GraphSmote

[7] Reinforced Neighborhood Selection Guided Multi-Relational Graph Neural Networks https://arxiv.org/pdf/2104.07886.pdf

[1] Lee, Donghua and Yu, Hwanjo. "Harmonized representation learning on dynamic EHR graph." Journal of Biomedical Informatics, June 2020. https://doi.org/10.1016/j.jbi.2020.103426

[2] Rocheteau, Emma et al. "Predicting patient outcomes with graph representation learning." Jan 2021. https://arxiv.org/pdf/2101.03940v1.pdf

[3] Choi, Edward et al. "Learning the Graphical Structure of Electronic Health Records with Graph Convolutional Transformer." Google Health, Jun 2019. https://arxiv.org/pdf/1906.04716v3.pdf

[4] Zhu, Weicheng and Razavian, Narges. "Variationally Regularized Graph-based representation learning for electronic health records." Dec 2019. https://arxiv.org/pdf/1912.03761v2.pdf

[5] Choi, Edward et al. "GRAM: Graph-based attention model for healthcare representation learning." Google Health, Nov 2016. https://arxiv.org/pdf/1611.07012v3.pdf

[6] Zhao, Tianxiang et al. "GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks." Mar 2021. https://arxiv.org/pdf/2103.08826.pdf

[7] Peng, Hao et al. "Reinforced Neighborhood Selection Guided Multi-Relational Graph Neural Networks." Oct 2021. https://arxiv.org/pdf/2104.07886.pdf

[8] Sun, Zhoujian et al. "Interpretable Disease Prediction based on Reinforcement Path Reasoning over Knowledge Graphs." Oct 2020. https://arxiv.org/pdf/2010.08300.pdf