PGCN: Disease gene prioritization by disease and gene embedding through graph convolutional neural networks

1 Introduction

Exusting disease-gene prioritization methods:

- Filter methods: simply filter out irrelevant genes based on prior knowledge.
- Text Mining: Score the candidate genes using literature data (co-occurrence of gene and disease names).
- Similarity Profiling & Data Fusion: Similar genes should belong to similar diseases.
- Network-based methods: Represent diseases and genes as nodes in a heterogeneous network, while similarities are represented as edges.
- Matrix Completion: represent disease-gene association with an incomplete matrix and try to complete it.

Limitations:

- Similarity-based methods fail to handle unknown diseases with no known associated genes.
- Network-based methods are biased by network topology and cnnot integrate multiple data sources effectively.
- Matrix completion methods assumes the relationship is linear, which is not true in biological systems.

2. Takeaway

- **Embedding Model**: Learn the potential representation of nodes from initial raw representations(information encoded from different sources), considering the graph's topological structure and the nodes' neighborhood. Combine these into an embedding vector.
- **Decoding Model**: Predict whether a disease and a gene are associated based on their embedding vectors. (edge prediction)
- These two models are trained **end-to-end**, which means we combine the two models into a same pipeline, using the same loss function to optimize both models simultaneously, and to

regularize each other.

3. Prerequisites

3.1 GCNs

Graph-based Semi-Supervised Learning: node classification problem in a graph where labels are only available for a small subset of nodes.

Model Representation:

$$f(X, A)$$

- $X \in \mathbb{R}^{N \times F}$ is the feature matrix, where N is the number of nodes and F **is the number of features per node.**
- $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix of the graph.

Propagation Rule:

$$H^{(\ell+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(\ell)}W^{(\ell)}\right)$$

- $\tilde{A} = A + I$ representing the adjacency matrix with added self loops.
- $\tilde{D}$ is the degree matrix of $\tilde{A}$, i.e.

$$\tilde{D}_{ii} = \sum_{j} \tilde{A}_{ij}$$

- $W^{(\ell)}$ is the **trainable weight matrix**(parameter) at the ℓ-th layer.
- $\sigma(\cdot)$ is an activation function.
- $H^{(\ell)}$ is the matrix of activations at the ℓ-th layer, and $H^{(0)} = X$.
- Final output:

$$Z = H^{(L)}$$

where L is the total number of layers.

This is called graph convolutional layer because the expression comes from a much-simplified version of spectral graph convolutions. The mathematical details are elaborated here.

BackPropagation:
Suppose we have a loss function $\mathcal{L}$ for the GCN, we first compute:

$$\frac{\partial \mathcal{L}}{\partial Z}$$

where Z is the output of the GCN model. Then we can use the chain rule to compute the gradients of all parameters $W^{(\ell)}$ in each layer:

$$\frac{\partial \mathcal{L}}{\partial W^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial Z} \cdot \frac{\partial Z}{\partial W^{(\ell)}}$$

$$\ell = L-1, L-2, \ldots, 0$$

Use this to update the parameters using gradient descent or other optimizers.

Solving the graph-based semi-supervised learning problem using GCNs:

- Input: feature matrix X and adjacency matrix A.
- Output: label predictions for all nodes.
- **Loss function**: cross-entropy loss over all labeled nodes.

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

where $\mathcal{Y}_L$ is the set of labeled nodes, Y is the real label, and Z is the output of the GCN model:

$$Z = f(X, A)$$

Recall F is the number of features per node.

3.2 TF-IDF

- **TF(Term Frequency)**:

$$TF = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}}$$

- **IDF(Inverse Document Frequency)**:

$$IDF = \log \frac{\text{Total number of documents}}{\text{Number of documents with term t in it}}$$

- **TF-IDF**:

$$TF - IDF = TF \times IDF$$

3.4 Ontology Similarities

3.4.1 Resnik pairwise similarity

- Calculate occurrence frequency of a term c in a corpus: $p(c)$
- Calculate information content of term c:

$$IC(c) = -\log p(c)$$

- For two terms c_1 and c_2, find their common ancestors in the ontology graph, and select the one with the highest information content:

$$\text{sim}_{\text{Resnik}}(c_1, c_2) = \max_{c \in \text{Anc}(c_1, c_2)} IC(c)$$

  where $Anc(c_1, c_2)$ is the set of common ancestors of c_1 and c_2.
- For two diseases d_1 and d_2 each connected with a set of terms $s(d_1)$ and $s(d_2)$ respectively, calculate the pairwise similarity as:

$$\text{sim}_{\text{Resnik}}(d_1, d_2) = \max_{c_1 \in s(d_1), c_2 \in s(d_2)} \text{sim}_{\text{Resnik}}(c_1, c_2)$$

3.4.2 Ontology Similarities

See **section 4.2.1.2** in `RBM_DBN.pdf` for details. Note that below we're using **disease ontology** instead of gene ontology, so the similarity calculation step should be referred from:

> where the functional similarities of two genes g_1, g_2 ...

in the original text, and all g should be replaced with d (disease).

3.4.2 Best Match Average(BMA) similarity

- **Representation**: Ontology databases are presented as directed acyclic graphs (DAGs) in which the terms form nodes and the two kinds of semantic relations (‘is-a’ and ‘part-of ’) form edges. ‘is-a’ is a simple class-subclass relation, where A is-a B means that A is a subclass of B. ‘part-of ’ is a partial ownership relation; C part-of D means that whenever C is present, it is always a part of D, but C need not always be present.

4. Method

4.1 Problem Formalization

- Nodes: a disease or a gene
- Edges: a specific kind of interaction.
- Supplemented Information
- Goal: Predict potential associations between diseases and genes.

4.2 Data Preprocessing Methods

- Hetrogeneous network: gene network, disease-similarity network, disease-gene association network. Combine them into a heterogeneous network with two types of nodes (diseases and genes) and three types of edges (disease-disease similarity, gene-gene interaction, disease-gene association).
- Construction of feature matrix X:
  If we denote the number of diseases as N_d and the number of genes as N_g, then the feature matrix X is of size $(N_d + N_g) \times F$, where F is the number of features per node.
  - Diseases:
    - Disease Ontology(DO). Use Gene Ontology Similarities to get a similarity matrix of size $N_d \times N_d$
    - Clinical Features from OMIM. First delete most frequent and rarest terms. Use **TF-IDF** to represent each disease as a vector, with a feature length F_1. Now we get another matrix of size $N_d \times F_1$.
  - Genes:
    - Microarray measurement from BioGPS and CLUE of gene expression level: Apply **PCA**(principal Component Analysis) to reduce dimension to 100. Get a matrix of size $N_g \times 100$.
    - Gene-Phenotype association matrix (row: gene, col: phenotype) (Note maybe this is unnecessary since we need to use matrix completion. See `matrix_completion_gene_disease.pdf` for details).
  - Concatenate the matrices for diseases and genes respectively to form the feature matrix X_d and X_g (note they may have different feature lengths).

4.3 GCN Construction

4.3.1 Node Embeddings

Propagation Rule:

$$\mathbf{h}_{i,k} = \sum_{\ell} \sum_{j \in \mathcal{N}_i^{\ell}} c_{i,j} W_{\ell}^{k} \mathbf{z}_{j,k} + W_{t_i,s}^{k} \mathbf{z}_{i,k}$$

$$\mathbf{z}_{i,k+1} = ReLU(\mathbf{h}_{i,k})$$

- $\mathbf{z}_{i,k}$: hidden representation of node i at layer k, where $\mathbf{z}_{i,0} = X[i]$, and **the final output embedding of node i is $\mathbf{z}_{i,N}$ after N layers.**
- $\mathbf{h}_{i,k}$: aggregated feature vector for node i at layer k.
- ℓ: edge type (disease-disease, gene-gene, disease-gene)
- N_i^{ℓ}: neighbor set of node i under edge type ℓ.
- $c_{i,j}$: normalization constant,

$$c_{i,j} = \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}$$

Learnable Parameters:

- W_{ℓ}^{k}: trainable weight matrix for edge type ℓ at layer k.
- $W_{t_i,s}^{k}$: trainable weight matrix for node type t_i (disease or gene) at layer k, preserving the node information itself.

4.3.2 Edge Decoding (prediction)

Edge Decoder:

$$P(d_i, g_j) = \sigma(\mathbf{z}_{d_i}^{T} \mathbf{W}_d \mathbf{z}_{g_j})$$

- σ: Sigmoid function.

Learnable Parameters:

- $\mathbf{W}_d$: trainable weight matrix modeling the interaction between disease and gene embeddings.

> Note that the **three learnable parameters** W_{ℓ}^{k}, $W_{t_i,s}^{k}$ and $\mathbf{W}_d$ are all trained together in an end-to-end manner, because they're shared over all nodes and edges, which prevents

> overfitting.

4.4 Hyperparameters

For the subgraph with sole gene-disease connections $G = (V, \mathcal{E})$:

- Loss function: cross Entropy:

$$\mathcal{L}(d_i, g_j) = -\log P(d_i, g_j) - \mathbb{E}_{g_n \sim \mathcal{D}(g_j)} \log(1 - P(d_i, g_n))$$

$$L = \sum_{(d_i, g_j) \in \mathcal{E}} \mathcal{L}(d_i, g_j)$$

  - $\mathcal{E}_{g_n \sim \mathcal{D}(g_j)}$: negative sampling, where g_n is a negative gene sampled from a distribution $\mathcal{D}(g_j)$. Here we want the model to output a low association probability for the negative sample.
  - $\mathcal{D}(g_i)$: uniform distribution over all genes known to be associated with disease d_i.
- Layer number : 2
- hidden layer dimension: 64
- Embedding dimension: 32
- Optimizer: Adam
- LR: 0.001
- Dropout rate: 0.1
- Weight Decay: 0.0005
- Parameter Initializer: Xavier
- batch size: 512
- 10-fold cross validation

> Note: the original paper trained the model in **10 hours**. You should consider using GPU acceleration or downsize the dataset.

4.5 Evaluation Criteria

Evaluation Metrics Explained

- **AUROC (Area Under the Receiver Operating Characteristic Curve)**: Measures the ability of the model to distinguish between positive and negative classes across all thresholds. Higher AUROC indicates better discrimination.

- **AUPRC (Area Under the Precision-Recall Curve)**: Evaluates the trade-off between precision and recall for different thresholds, especially informative for imbalanced datasets.
- **BEDROC (Boltzmann-Enhanced Discrimination of ROC)**: Focuses on early recognition by giving higher weight to the top-ranked predictions, useful when early retrieval is important.
- **AP@K (Average Precision at K)**: Computes the average precision of the top K ranked predictions, reflecting how many of the top K are true positives.
- **R@K (Recall at K)**: Measures the proportion of true positives found in the top K predictions, indicating how many relevant items are retrieved among the top K.