Implementing Correct and Smooth Model for Transductive Node Classification Tasks

Anonymous ACL submission

Abstract

As part of our Graph Mining assignment, we implement the 'Correct and Smooth' procedure as introduced in the paper [1]'COMBINING LABEL PROPAGATION AND SIMPLE MODELS OUT-PERFORMS GRAPH NEURAL NETWORK' by Huang et al. for transductive node classification tasks. For experiments, we use the Cora, Citeseer, US County datasets (also used in the original paper) which have node features as well as edges between nodes. Our implementation is seen to perform better than the original paper's implementation for the Cora and Citeseer datasets and perform almost similar for the US county dataset. Through our results we also demonstrate that base model performance improves when we introduce graph awareness using label propagation.

1 Introduction

Graph neural networks (GNNs) have emerged as a popular solution for learning over graphs. However, recent research by Qian Huang ete al. suggests that a combination of label propagation and simple models can outperform GNNs in certain scenarios.

The key insight from Huang's work is the proposal of a correct and smooth pipeline that harnesses the complementary strengths of label propagation and simple models. The correct and smooth pipeline involves initially applying simple base predictor models in a graph-agnostic manner using node features as data and then apply label propagation to exploit the inherent structure of the graph. This methodology addresses some of the limitations observed in traditional GNNs by requiring lesser number of features and lower computation time, offering a promising alternative for graph-based learning tasks.

2 Methodology

As per the proposed methodology, we adopt a three-step approach. In the first step, we employ a robust base predictor algorithm to generate initial predictions on both the training and test datasets. This base predictor serves as the starting point for subsequent refinements. The second step involves spreading residual errors identified in the training dataset to correct and update predictions. This error correction process aims to capture nuanced patterns within the training data, contributing to improved model accuracy. At this point we also perform autoscaling to adjust the mass of the residual and to increase the impace of the error propagation step. In the final step, we implement a smoothing mechanism to optimize predictions on the test dataset.

2.1 Base Predictor

To establish a foundational model for our predictive task, we employ two distinct base predictors: Logistic Regression and a Multi-Layer Perceptron (MLP) neural network. These models serve as the initial building blocks for our methodology, capturing different aspects of the underlying data patterns.

2.1.1 Logistic Regression

We first train a Logistic Regression model. This provides a baseline for our predictive task. We utilize the scikit-learn implementation with a maximum of 1000 iterations.

2.1.2 Multi-Layer Perceptron (MLP)

Next, we explore the capabilities of a more complex model, the Multi-Layer Perceptron (MLP) neural network. The training of the MLP involves hyperparameter tuning and early stopping for efficient convergence. We perform a grid search over a predefined set of hyperparameters, including hidden layer sizes, regularization strength (α), and the initial learning rate. The grid search is conducted using a two-fold cross-validation scheme, and the scoring metric employed is the negative log loss.

The best-performing MLP model is selected based on the results of the grid search. Notably,

early stopping is enabled with a validation fraction of 0.1, allowing a portion of the training data to serve as a validation set for monitoring convergence.

2.2 Error Propagation

In this subsection, we delve into the process of error propagation, a crucial step in refining our predictions. The error propagation methodology involves several key functions and steps:

2.2.1 Normalized Adjacency Matrix Calculation

We begin by calculating the normalized adjacency matrix, denoted as S, using the input adjacency matrix A. The matrix S is computed by normalizing A based on node degrees.

$$S = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

where D is the diagonal degree matrix.

2.2.2 Residuals Computation

Next, we calculate the residuals, denoted as E, representing the difference between the true labels Y and the current predictions Z.

$$E = Y - Z$$

2.2.3 Label Spreading

The label spreading function incorporates the graph structure to propagate and refine the residuals. The process iteratively updates a matrix W using the normalized adjacency matrix S and a user-defined parameter α. The iteration continues until convergence or a maximum number of iterations is reached.

$$W_{\text{new}} = (1 - \alpha) \cdot W + \alpha \cdot S \cdot W$$

2.2.4 Correcting Predictions

To correct the predictions, we add the label-spreading-processed residuals (F_{hat}) back to the original predictions (Z).

$$Z_{\text{corrected}} = Z + F_{\text{hat}}$$

2.2.5 Best Alpha Selection

The process involves determining the best value for the parameter α. We iterate over a range of α values, computing the accuracy on a validation set for each. The α yielding the highest validation accuracy is selected as the best value.

2.2.6 Final Correction

Finally, the best α is utilized to perform the error propagation on the entire dataset, resulting in the corrected predictions ($Z_{\text{corrected}}$). This step enhances the model's predictive capabilities by leveraging graph structure and label spreading.

The outputs of this process include the corrected predictions ($Z_{\text{corrected}}$), label-spreading-processed residuals (F_{hat}), and raw residuals (F), providing insights into the refined predictions and the impact of error propagation.

2.3 Autoscaling

2.3.1 Autoscale Function

The `autoscale` function is responsible for adjusting the scale of the residuals (E_{hat}) based on a scaling factor derived from the original residuals (E) and the number of labeled nodes (L). The scaling process is designed to ensure that the corrected predictions take into account the magnitude of the residuals while avoiding division by zero.

$$\text{E\_auto}[L:] \times = \left(\frac{\sigma}{\text{norm\_E\_hat}} \right)$$

where σ is the mean L_1-norm of the original residuals.

2.3.2 Scaled Fixed Diffusion

The `scaled_fixed_diffusion` function applies a diffusion process to adjust the scale of residuals. This process involves iteratively updating the residuals based on the graph structure (A) and the diagonal degree matrix (D). The diffusion process aims to achieve a scaled version of the residuals while preserving the overall structure.

$$\text{E\_fixed}[: \text{val\_end}] = E[: \text{val\_end}]$$

$$\text{E\_fixed}[\text{val\_end} :] = D^{-1} \cdot A \cdot \text{E\_fixed}[\text{val\_end} :]$$

2.3.3 Scaling Method Application

Both 'autoscale' and 'fixed diffusion' scaling types have been implemented. The final corrected predictions are obtained by adding the scaled residuals to the original corrected predictions.

2.4 Final Label Propagation (Smoothing)

The final step in our methodology involves refining the corrected predictions ($Z_{\text{corrected}}$) through label propagation, smoothing the predictions over the entire graph. This step is crucial for enhancing the model's ability to generalize and make more accurate predictions. The key components of this process are outlined below:

2.4.1 Label Propagation Function

The `propagate_labels` function performs label propagation on the corrected predictions (H). It utilizes the normalized adjacency matrix (S) and a user-defined parameter α to iteratively update the predictions. The process continues until convergence or a maximum number of iterations is reached.

$$H_{\text{new}} = (1 - \alpha) \cdot H + \alpha \cdot S \cdot H$$

2.4.2 Smoothing Process

The final smoothing process involves applying the label propagation function to the corrected predictions. The labels are propagated over the entire graph, refining the predictions.

$$H = \text{propagate\_labels}(Z_{\text{corrected}}, S, 0.1, \text{max\_iter})$$

Additionally, training labels are retained, ensuring that the training set remains unchanged during the label propagation.

$$H[:\text{validation\_end}] = Y[:\text{validation\_end}]$$

This step contributes to the overall robustness and generalization of the model, providing a more refined set of predictions (H) for the entire dataset.

3 Experiments

3.1 Dataset

3.1.1 Citeseer

The Citeseer dataset comprises 3312 scientific publications categorized into six classes: Agents, AI, DB, IR, ML, HCI. The citation network is formed by 4732 links connecting these publications. Each publication is represented by a 0/1-valued word vector indicating the presence or absence of corresponding words from a dictionary of 3703 unique words.

3.1.2 Cora

The Cora dataset is centered around a citation network in the domain of scientific research papers. It includes 2708 machine learning papers classified into seven classes: Case_Based, Genetic_Algorithms, Neural\Networks, Probabilistic_Methods, Reinforcement_Learning, Rule_Learning, and Theory. The dataset contains 5429 edges representing citations, with each paper citing or being cited by at least one other paper. The vocabulary comprises 1433 unique words.

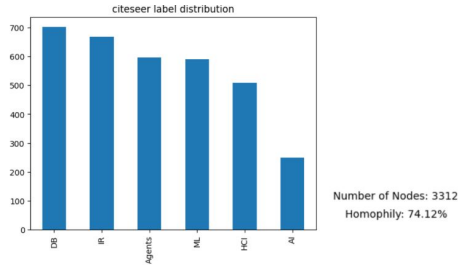

Figure 1: Citeseer Label Distribution with edge homophily

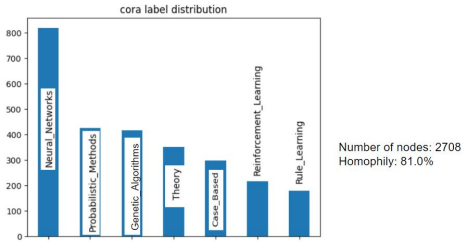

Figure 2: Cora Label Distribution with edge homophily

3.1.3 US County

The US County dataset represents a network where nodes correspond to US counties, and edges connect bordering counties. Each node encompasses demographic features and presidential election outcome results (number of democratic and republican votes) from 2012 and 2016. The dataset consists of 3234 nodes, 12717 edges, and each node is characterized by 6 demographic features.

3.2 Implementation Differences from Original Paper

Error propagation step: The paper's implementation spreads the errors on the train data alone to all other data points. To leverage the information on the validation data, we decided to choose the smoothing parameter alpha from a range of values by testing performance on the validation data with each value. Once we found the optimal value of alpha, we run the propagation step again using

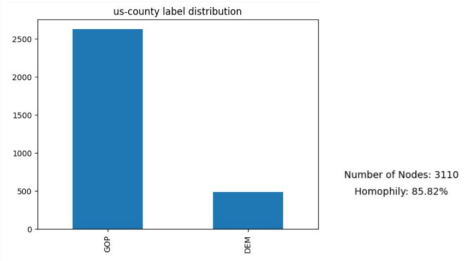

Figure 3: US County Label Distribution with edge homophily

the best alpha value but this time, spreading errors from validation as well as train.

Final label propagation step: Instead of retaining true labels on the train data alone, we retained true labels on train as well as validation data during the iterative propagation. However, choosing of hyperparameter alpha could not be done here using validation data as the model had already acquired validation information during the error spreading.

The same validation data was used for training and hyperparameter tuning of the MLP base predictor

3.3 Overall Performance

The performance of ouor model and its comparison with the implementation in the original paper is giveen in Table 1 and Table 2.

3.4 References

References

[1] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. 2020. Combining label propagation and simple models out-performs graph neural networks.

Acknowledgements

We extend our thanks to Dr. Vinti Agarwal for giving us the opportunity to work on this project and for her guidance and mentorship throughout this endeavor. Her expertise and support have been instrumental in shaping the direction of this work.

Method	**citeseer**				**us-co**	
	base	linear	mlp	mlp_se	base	linear
base	71.19	73.00	71.95	72.25	84.20	85.98
error_propagation	71.34	76.02	72.25	73.15	85.34	86.69
autoscale	72.85	74.21	73.00	75.26	86.43	86.95
Label Propagation autoscale	**77.07**	75.41	76.47	76.77	86.82	86.50
fixed_diffusion	72.55	76.17	72.10	73.15	86.56	87.27
Label Propagation fixed_diffusion	75.87	76.17	76.77	75.87	86.62	87.07

Table 1: Results from our experiments

	Original Paper	**Our implementation**
Citeseer	72.56%	**77.07%**
Cora	79.56%	**87.08%**
US County	**89.62%**	89.30%

Table 2: Comparison of results