Data Mining and Predicting on Academic Networks Based on Graph Neural Networks

Ruiwen Zhou, Rui Ye, and Zhiyu Zhang[*]
Department of Electronic Engineering
Shanghai Jiao Tong University
{skyriver,yr991129,zhiyu-zhang}@sjtu.edu.cn

Abstract

In this paper, we tackle the node classification and link prediction problem on an academic network consisted of citation information of publications on top AI conferences. We first build a unified homogeneous graph involving author cooperation and paper citation relationships. For node classification task, we propose a simple but effective self-designed feature and use SGC to avoid the risk of information leakage. For link prediction task, we utilize SEAL framework with label smoothing and hierarchical ASAP graph classifier to alleviate overfitting. Furthermore, we utilize ensemble learning for better performance. We also conduct extensive experiments and verify the robustness and effectiveness of our solution, which ranked 2nd on the leaderboards of both tasks.

1 Introduction

In this project, we work on an academic network which is composed of 42,614 authors and corresponding 24,251 papers from 10 top conferences (2016 - 2019) in the field of Artificial Intelligence and Data Mining as well as citation information of their publications. This project includes two tasks, which are node classification and link prediction.

In the task of node classification, our ultimate objective is to assign each author node with multiple labels. That is, if the author has published papers in conference A, he/she will be labelled as A; if he/she published papers in several different conferences, he/she will have multiple labels.

In the task of link prediction, the provided author and paper information is from 2016 to 2019, and some authors may have new papers published in 2020. Consequently, there will be new cooperation and citation relationships between authors, and new edges will be connected between corresponding author nodes in the academic network. The ultimate objective is to predict each pair of author nodes in the test set based on the information provided. If there will be an edge between the two given nodes in 2020, the label is set to 1, otherwise 0.

Both homogeneous and heterogeneous network are applicable, and we choose to form a homogeneous network where each node can represent either an author or a paper. And each edge represents the citation relation between papers, the connection between a paper and its every author or co-authorship between the two connected authors. The node classification and link prediction tasks are both based on this academic network.

In this work, we tackle both the problem of node classification and link prediction. Our main contribution can be summaried as the followings:

[*]Group 12. Our codes are available at: `https://github.com/SkyRiver-2000/EE226-Final-Project`

1. Node classification: With properly designed feature and suitable graph convolutional networks (SGC & GCN), our classification model achieves 0.52216 evaluated by mean F1-Score on public dataset, ranked 2nd.
2. Link prediction: Introducing ASAP and soft labels to the SEAL framework, our prediction model achieves 0.79756 evaluated by AUC on testing dataset, ranked 2nd. (Further improved as discussed in Section 7.1, achieving a current best record of 0.81069)
3. Ensemble learning: Utilizing the technique of bagging, the performance is further improved.

2 Related Works

2.1 Representation Learning on Graph Data

Representation learning means learning representations of the data that make it easier to extract useful information when building classifiers or other predictors[1]. It has long been studied as the learned representations, or features, could have a significant influence on the resultant performance of downstream machine learning algorithms.

Assuming that connected nodes are more likely to be similar, Perozzi et al. [2] proposes to generate "node sentences" on graphs using random walks and extends the Word2Vec method [3] to node embedding. As DeepWalk does not provide much space for parameter tuning, Grover and Leskovec [4] improves its performance by utilizing biased random walks and several other techniques.

From another perspective, Tang et al. [5] defines the first-order and second-order proximity and optimize network embeddings accordingly, and Wang et al. [6] further incorporates node attributes, which also achieves competitive performance with random-walk-based methods.

2.2 Graph Neural Networks and Its Applications

Much effort has been put into extending neural networks (especially convolution operator) to graph data, but during a long period no results reach a good compromise between expressive power and computation efficiency. Kipf and Welling [7] presents a graph convolutional layer which scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. Velickovic et al. [8] introduces attention mechanisms into GNNs to capture the importance of different nodes while Wu et al. [9] tries to simplify GCN without negative impacts on its performance.

The models mentioned above focus on learning node-level representations, and GNNs have also been studied for edge-level or graph-level learning tasks. The simplest way to utilize a GNN for a link-level learning task is to replace the encoder in an auto-encoder structure, which forms the Graph Auto-Encoder (GAE) [10]. Zhang et al. [11] presents a global sort pooling layer which builds an end-to-end framework for graph classification. On this basis, they propose SEAL framework [12], which predicts the existance of links by doing classification on an h-hop subgraph around the link and is one of the most powerful method for link prediction until today.

As the global sort pooling layer has shortcomings like high probability to overfit, Lee et al. [13], Ranjan et al. [14] studies different pooling layers while Gao and Ji [15] borrows the idea of classic U-Net structure [16] from computer vision, and they all improve the SOTA performance of graph classification to some extent.

3 Academic Network Modelling

In our project, we construct the academic network in a homogeneous form. That is, the authors and papers are regarded as nodes in the same type in the graph network, though treated in different manners in the process of training and inferring. There are three types of link. Type 1 link is called citation link, which represents the relation of reference. Type 2 link is called writing link, which bridges between paper and its author. Type 3 link is coauthor link. Figure 1 is a simple illustration of our academic network.

The decision of such network structure is based on many different attempts. First, we only hold the author or paper nodes, which leads to obviously poorer performance. Such result is reasonable since

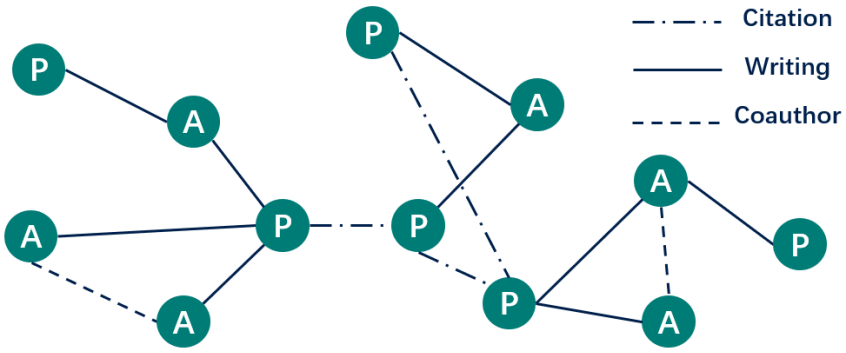


Figure 1: A simple illustration of our constructed homogeneous academic network. Author and paper are in the same form while there are three types of link, which can be distinguish by the line format.

that node can help deliver information from one node to another and this attempt eliminates such information delivering. We also try different combinations of link types and find that including all the three types of link results in the best performance. Note that for the task of node classification, the weight of different types of link can vary to further improve the performance.

4 Solution to Node Classification

The ultimate objective is to predict authors' labels, which involves multi-label learning if treated directly. Multi-lable learning is much more trickier compared with one-label learning. To avoid this issue, we decompose this task into two steps: paper prediction and combination. Based on the provided labels on several papers, we train and valid a graph convolutional network for classification. That is, we train and infer on the paper nodes to obtain papers' label (0 to 9), then combine the labels of papers that belong to the same author.

The achieved satisfactory performance (Ranked 2nd) mainly owe to two factors: feature design and the choice of graph neural network (GNN). The ensemble learning technique also makes some small contributions.

4.1 Feature Design

Generally, initial features are required to train a GNN. However, in this task, no initial features are given. Thus, how to construct the initial features is an issue that should be concerned about. There are three kinds of initialization method.

First, there are three relatively simple ways: assigning all ones, random initialization and one-hot initialization based on node's ID. However, these initialization ways do not introduce useful information to GNN, which lead to relatively poor performance.

Second, graph embedding technique (e.g. node2vec, graphsage) can also be adopted. The obtained embeddings are then assigned as the initial features of the nodes. With node2vec embeddings, the performance is largely improved compared with the previously stated three ways.

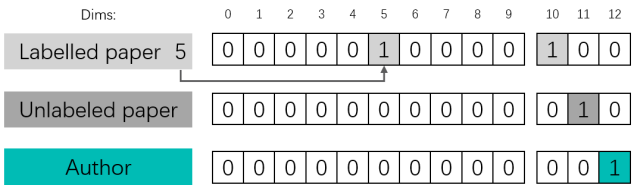


Figure 2: Applied feature design. Nodes are classified into three types, which are labelled paper, unlabeled paper and author, and are treated separately. Among the 13 elements (starting from 0), the element 10, 11 and 12 denote whether the node is labelled paper, unlabeled paper and author, respectively.

The third method is mainly based on one-hot paper label and proven to be the most effective way. The only information about paper provided are label and pub-year. However, in this setting, introducing pub-year information may even bring adverse effects. Thus, we only utilize the label information and encode it into a one-hot vector. Nodes are classified into three types (labelled paper, unlabeled paper and author) and represented by a 13-element feature. The first 10 elements are the encoded one-hot vector for labelled paper while for the unlabeled paper and author they are ten zeros. The last three elements denotes the type of the node. The applied feature design is illustrated in Figure 2.

4.2 Choice of GNN

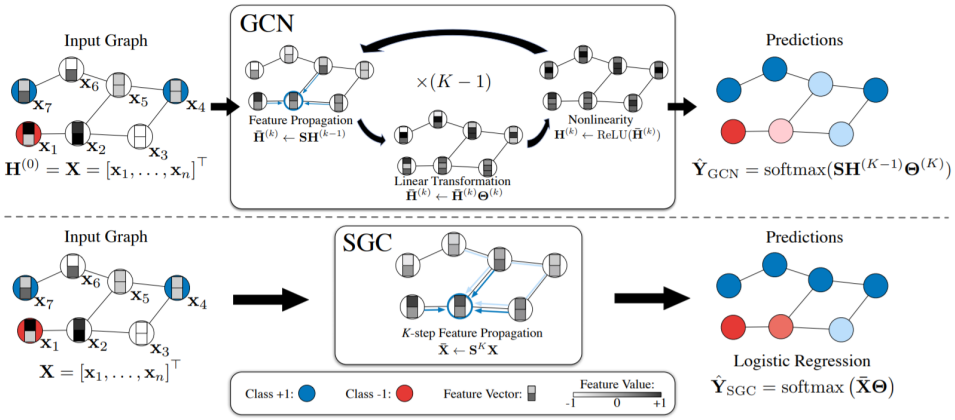


Figure 3: Schematic layout of a GCN and a SGC. The top one is GCN, which transforms the feature vectors repeatedly throughout K layers and then applies a linear classifier on the final representation. The bottom one is SGC, which reduces the entire procedure to a simple feature propagation step followed by standard logistic regression. [9]

In this work, we adopt two types of GNN, GCN and SGC, with their schematic layouts illustrated in Figure 3. There are many types of GNN that can be utilized in this task. As we conduct the experiments, we find that SGC is ahead of the other GNNs apparently. As for the ensembling process, it turns out that the weighted aggregation of GNN and SGC will lead to the best performance.

4.3 Implementation details

In the task of node classification, there are 24251 papers in the dateset, where 4844 are given with a label (0 to 9). We split the 4844 papers into a training set and validation set. The training set has 4601 samples and the validation set has 243 samples. For the process of training, only the nodes in the training set are treated as labelled papers while the nodes in the validation set are treated as unlabelled papers. As we train the GNN, we validate the model on the validation set and save the model that performs the best in validation set. The papers without a label are then predicted with a label in the inference process. Finally, we combine all the label of papers that belong to one same author.

5 Solution to Link Prediction

Our work on link prediction mainly revolves around the SEAL framework. However, the default setting of SEAL framework has a severe problem of overfitting, which leads to poor generalization. To avoid this tricky issue, we come up with two ideas. One is transforming the hard labels into soft labels and the other one is replacing the DGCNN in SEAL by Hierarchical ASAP Pooling Net. With several more adjustments, we obtain a satisfactory result on this task.

5.1 General View of SEAL

Before SEAL was proposed, mainstream methods on link prediction problem focus on inferring the existance of a link from its two endpoints, either by node embedding or node attributes, while SEAL explores the problem from the perspective of subgraph classification. As shown in Figure 4, SEAL

extracts an h-hop local enclosing subgraph around the target link (without the target link itself) as the target for its graph classifier, and proposes a useful node relabelling method called DRNL, which works well without node attributes and fits the condition of our task perfectly.

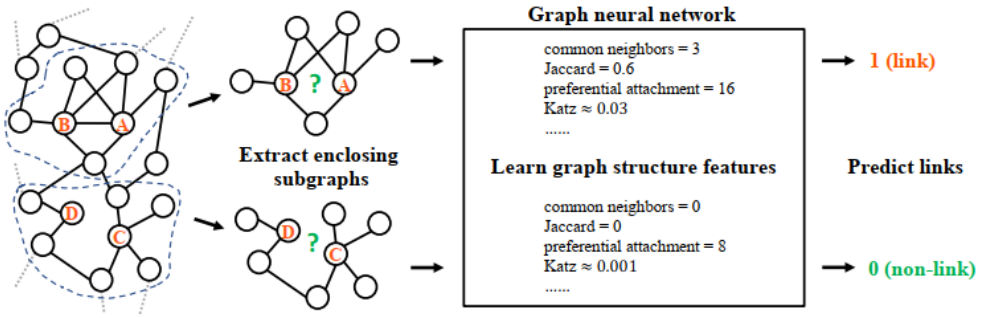


Figure 4: The SEAL framework. For each target link, SEAL extracts a local enclosing subgraph around it, and uses a GNN to learn general graph structure features for link prediction. [12]

5.2 The Reason for Our Choice

In general, we can come up with four types of solution for link prediction tasks:

1. Build a classifier with hand-crafted heuristics (e.g. Katz Index, Adamic-Adar, etc.).
2. Build a classifier with unsupervised node embeddings (e.g. node2vec embeddings).
3. Use graph generative models with a reconstruction loss (e.g. graph auto-encoders).
4. Treat the task as a subgraph classification problem (e.g. SEAL framework).

We motivate the use of SEAL framework with the following three reasons:

1. As stated in [12], GNN in SEAL can approximate most heuristics with exponentially decayed error, so it has higher expressive power than simple heuristics.
2. Unsupervised node embeddings only consider the similarity of nodes from a certain perspective, while SEAL uses h-hop subgraphs and incorporates more information.
3. Graph auto-encoder and its variants are powerful tools when nodes are attributed. However, this is not our case, and we will emperically prove SEAL is better for us in Section 6.3.

5.3 Improvements on SEAL

Using the default settings of SEAL (e.g. binary link labels and one DGCNN as classifier), we train the classifier for 3 epochs with 1e-4 learning rate and no weight decay, and we obtain some prediction for the existance probability of links distributed as shown in Figure 5.

From Figure 5, we find that most of link probabilities predicted by DGCNN falls in a narrow range around 0.0 or 1.0, indicating our model is too confident on a large proportion of samples, or overfits to the training set. Since our target is to improve the AUC score, this phenomenon is malignant as most wrong prediction would cause a huge loss on our target. Therefore, we propose the following three methods to alleviate the overfitting problem.

Soft label for links Label smoothing is a widely-used trick against overfitting. However, we are facing a binary classification problem, and simply using soft labels with fixed probability provides little help. Therefore, we design a confidence score for each existing cooperation link between authors, which helps more in our situation.

To be specific, for each pair of authors (a_i, a_j) that cooperates $n_{ij} > 0$ times, we use the following score as a "soft label" for the author cooperation link.

$$s(a_i, a_j) = \sigma(\beta n_{ij}) = \frac{1}{1 + \exp(-\beta n_{ij})} \tag{1}$$

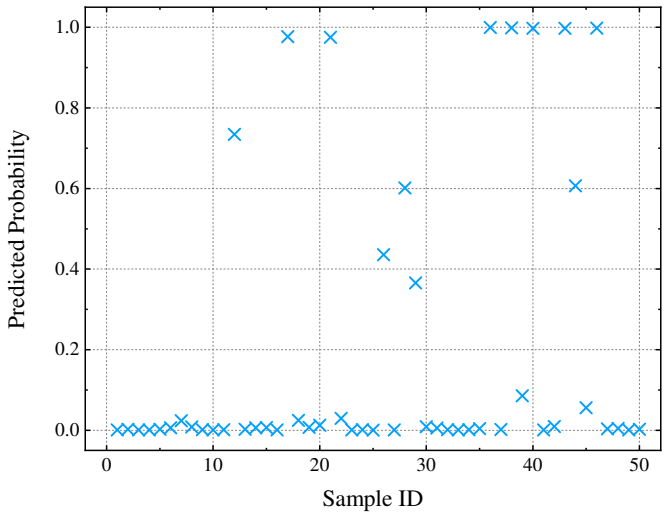


Figure 5: The distribution of prediction for 50 link samples randomly drawn from the training and validation set. This prediction achieves an AUC score of 0.76214 on the whole testing set.

As indicated by Equation (1), the more cooperation a pair of authors have, the higher confidence in the existance of their links, which regularizes the classifier and benefits the performance on the testing set even more due to its different data distribution. The "temperature" β could be adaptively adjusted, and we set it to 0.5 according to the result of grid search on its value.

Hierarchical ASAP *Adaptive Structure Aware Pooling* (ASAP), proposed by Ranjan et al. [14], extends the idea of DiffPool [17] and SAG Pooling [13]. As shown in figure 6, each ASA pooling layer generates local clusters with self-attention node aggregation, which serves as graph nodes for its successive layer. This hierarchical structure can both extract important information step by step and scale to large graphs without loss of efficiency and effectiveness.

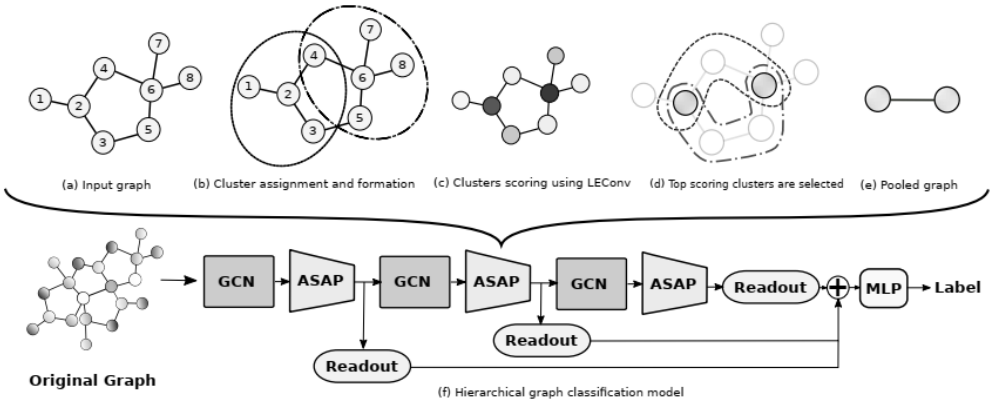


Figure 6: The hierarchical ASAP framework [14]. For a target graph, hierarchical ASAP extracts local information step-by-step with an ASA pooling operator and a readout layer [18]. Note that we follow the original design and substitute GCN convolution with LE convolution.

Bagging graph classifiers As a common trick for variance reduction and performance improvement in machine learning, ensembling several "weak" classifiers into a strong one usually outperforms single strong classifiers. Due to the extreme overfitting in our situation, boosting, which is often seen as a simple, safe and effective choice for ensembling, does not satisfy our requirement, so we decide to implement the naive bagging idea on our subgraph classifier (e.g. hierarchical ASAP network).

5.4 Final Configuration

With the same network modelling as described in Section 3, we ensemble 10 hierarchical ASAP graph classifiers and utilize the soft label for links mentioned in Section 5.3. The positive training links includes not only author cooperation links but also paper citation links and author-paper links as a regularization, and we sample negative training edges excluding positive edges and authors with citation relationship.

To further alleviate overfitting, we set the dropout rate for each fully connected layer to 0.3, and train our model using Adam optimizer with 0.01 learning rate and 1e-3 weight decay for only one epoch.

6 Experiments

6.1 Experiment Setup

For this project, we use python 3.7.0 with pytorch 1.8.1, torch-geometric [19] 1.7.0, and lightgbm 3.1.1 as development environment (dependencies and basic packages like numpy, pandas, scikit-learn, etc. are omitted for simplicity, and correct conda or pip installation is just fine).

For the node classification problem, we adopt four classic aggregation operators as our baseline: GCN [7], GAT [8], SAGE [20], and RGCN [21]. Note that we only specify extra edge types to form an heterogeneous graph for RGCN. For the link prediction problem, we report the best (as we tuned) result of several common methods for comparison.

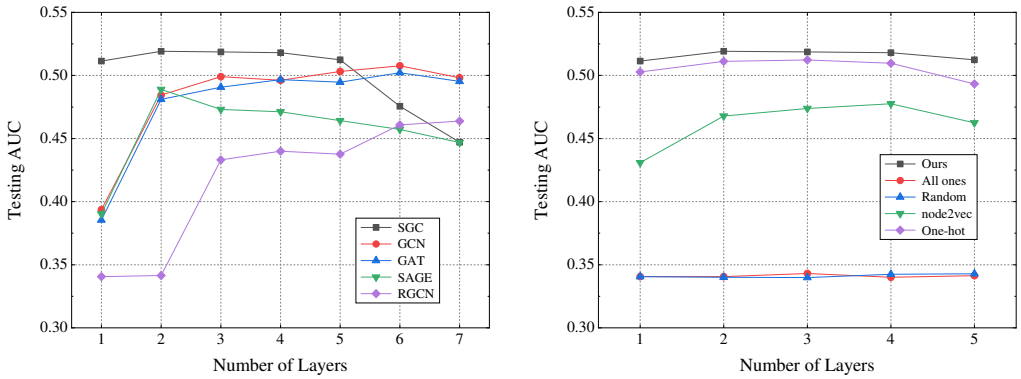
For both problems, we adopt the score on Kaggle leaderboard as our evaluation metric, namely mean F1-score for node classification task and AUC score for link prediction task.

6.2 Node Classification

In this section, we conduct several experiments to explore the impacts of different aggregating operators, initial features and edge weights on the task of node classification.

6.2.1 The Impact of Aggregating Operator

With all the edge weights set to 1 and our designed initial features, we explore the impact brought by different aggregating operators. As shown in Figure 7a, we plot layers-AUC curves for different aggregating operators. According to the curves, SGC proves its superiority with layers' number less than 6 and achieves the overall best result when the layers' number is 2 to 4. Apart from SGC, we find that GCN wins the second place when the number of layers is 6. This facts enlighten us to utilize SGC and GCN in the process of ensemble learning.



(a) Aggregating operator (b) Initial features

Figure 7: The influence of several factors on model performance

6.2.2 The Impact of Initial Features

With all the edge weights set to 1 and SGC as the aggregating operator, we compare the effectiveness of different initial features. As shown in Figure 7b, we also plot layers-AUC curves for different initial features. Note that the one-hot denotes one-hot 10-element features based on paper's label and ours is our designed 13-element features (append 3 indicative elements to the one-hot vector). Apparently, our designed features overwhelmingly takes the lead for all numbers of layers. These experiments prove the effectiveness of the disigned one-hot feature and the three indicative elements by contrast.

6.2.3 The Impact of Edge Weights

With our designed initial features and SGC as the aggregating operator, we search for the most suitable edge weights. As shown in Table 1, we test the AUCs for different edge weight ratios (x:y:z denotes the weight of citation edge: writing edge: coauthor edge). According to the listed results, when the weight ratio is 2:1:1, we achieve the highest testing AUC (double the weight of citation edge).

Table 1: The influence of edge weight ratio on model performance

Weight ratio	1:1:1	**2:1:1**	1:2:1	1:1:2	1:2:2	2:2:1	2:1:2
Testing AUC	0.51861	**0.52168**	0.51343	0.51728	0.51169	0.51918	0.51115

6.3 Link Prediction

Here we emperically study the performance of several available methods for the link prediction task. For each method listed below in Table 2, we only report the best result (AUC) it achieves on the whole testing set of Kaggle.

Table 2: Testing AUC of several methods

Method	**Testing AUC**
node2vec + Logistic regression	0.51444
node2vec + GBDT (lightgbm)	0.62094
VGAE without node attributes	0.73210
VGAE with node2vec embeddings	0.75314
SEAL (1x DGCNN, hard labels)	0.76214
SEAL (1x DGCNN, soft labels)	0.78058
SEAL (10x DGCNN, soft labels)	0.78842
SEAL (1x ASAP, soft labels)	0.78934
SEAL (10x ASAP, soft labels)	**0.79756**

As we can see from Table 2, naive classifier built with node2vec performs poorly, and we guess it's due to some characteristics of our network design, which we will discuss in Section 7.1.

Due to the lack of node attributes, generative models (e.g. VGAE) cannot fully exploit its power to encode local information and thus gives a decent while not very satisfactory result. We also notice that node2vec embedding as initial node attributes does not provide much performance gain to VGAE as the GCN encoder might have done similar things.

Even with default settings, we find SEAL achieves a better result over all other methods (the blue line). We also test the tricks we introduce against overfitting, and the results show that both soft labels and hierarchical ASAP structure lead to better performance of SEAL. The red line indicates the solution we finally adopt for the link prediction task.

7 Discussions

7.1 The Influence of Node2vec Embeddings

Our reported result of node2vec with binary classifier does not consist with the result of some other groups, especially Group 15 who adopted this method and achieved an AUC higher than 0.8, showing a huge difference. We guess the main reason for this seemingly strange phenomenon is that:

As described in Section 3, we build a homogeneous network involving both author and paper nodes and excluding author citation links as they are so dense that they are harmful to training. As our links are "not complete" and homogeneous, the node2vec model for our mixed homogeneous graph (rather than node2vec model for author-only graph or metapath2vec [22] model for heterogeneous graph) could not absorb enough information for link prediction between authors.

We discuss with Haochen Zhao in Group 15, follow their setting and generate node2vec embeddings. Using lightgbm GBDT With slightly tuned parameters, we find this method (without voting) achieves an AUC score of 0.791, and shows significantly different prediction on some links with our solution.

We further ensemble their proposed method and ours, and raise our AUC score on the whole testing set to 0.81069, which is the highest as far as we know.

7.2 Step-by-step Training for Node Classification

As we explore the distribution of paper labels (shown in Table 3), we find that the papers with label 5 accounts for a large proportion. Thus, we search for another training strategy targeted in this scenario. Here, we propose a potentially feasible training method called step-by-step training. This method is composed of three steps:

1. Train a prior binary classification model. In this period, papers provided with label 5 is assigned as class 1 while other papers are all categorized as class 0.
2. Train a subsequent multiclass classification model. In this period, papers with label 5 are excluded to make a 9-class classification.
3. Results aggregation of the above two models. First, mark the papers as 5 if is classified as class 1 by the prior model. For the papers not marked, assign them with label from 0 to 4 and 6 to 9 according to the prediction values given by the subsequent model.

The schematic diagram of this step-by-step training process is shown in Figure 8.

Table 3: The number of samples belonging to each class

Class	0	1	2	3	4	5	6	7	8	9
#Samples	233	321	825	776	31	1325	233	311	506	283

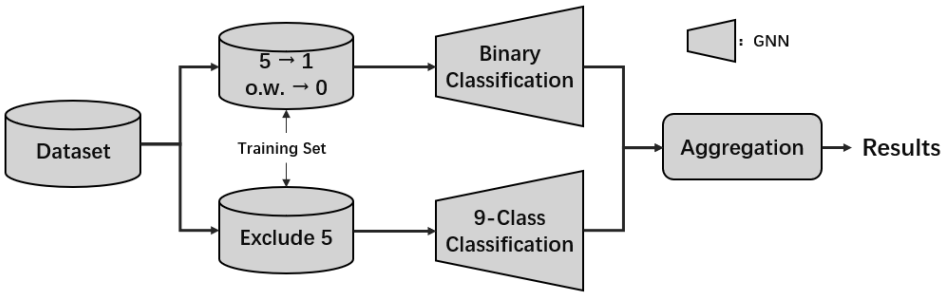


Figure 8: Step-by-step training strategy. The above flow transforms the dataset into a binary dataset and goes through a binary classification GNN. The below flow excludes papers with label 5 from the origin dataset and goes through a 9-class classification GNN. The final step is the aggregation of the results of the two flows.

7.3 Distribution difference and overfitting

Along the study into this project, we find that there could be a huge difference between the distribution of testing dataset and the provided training dataset. As we conduct our experiments with different methods, most could achieve 0.9+ or even 0.99+ AUC score on the validation set, but fail to exceed 0.8 AUC score on the Kaggle testing set, inspite of all efforts and tricks we tried against overfitting. This indicates that the distribution of two dataset might have a huge difference, since this 0.2 score gap cannot be explained with accidental factors or other common reasons.

8 Conclusions

In this paper, we work on the node classification and link prediction problem on an academic network consisted of citation information of publications on top AI conferences. Our main results are summarized as following:

1. We build a simple homogeneous network including both author and paper nodes, which has expressive power not less than a fancy heterogeneous graph.
2. We propose a novel feature design scheme from paper labels, which uses only 13 elements, and utilize SGC to avoid information leakage, achieving a relatively decent result in node classification task.
3. We solve the link prediction problem based on SEAL framework and introduce several tricks to alleviate overfitting and enhance the performance of our model.
4. We also conduct extensive experiments and analysis for a better insight into this task.

Due to the lack of node attributes and huge distribution difference between testing set and the provided training dataset, overfitting is a severe and hard-to-solve problem in this project, and this might be further studied in the future.

9 Acknowledgements

This project for EE226 course in SJTU was designed by Prof. Jiaxin Din and the teaching assistants. They also provided helpful suggestions and nsightful discussions. Thanks for their time and effort.

The project is formulated as two online Kaggle competitions, and thanks to other teams in our class for their great performance. Thanks to the APEX lab and MediaBrain lab for their hardware support.

References

[1] Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 35, no. 8, pp. 1798–1828, 2013.

[2] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2014, pp. 701–710.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *arXiv preprint, arXiv*, 2013.

[4] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016, pp. 855–864.

[5] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web (WWW)*, 2015, pp. 1067–1077.

[6] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016, pp. 1225–1234.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[8] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proceedings of the 6th International Conference on Learning Representations, (ICLR)*, 2018.

[9] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proceedings of the 36th International Conference on Machine Learning, (ICML)*, vol. 97, 2019, pp. 6861–6871.

[10] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *arXiv preprint, arXiv*, 2016.

[11] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018, pp. 4438–4445.

[12] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proceedings of the 31st Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 5171–5181.

[13] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, vol. 97, 2019, pp. 3734–3743.

[14] E. Ranjan, S. Sanyal, and P. P. Talukdar, "ASAP: adaptive structure aware pooling for learning hierarchical graph representations," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020, pp. 5470–5477.

[15] H. Gao and S. Ji, "Graph u-nets," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, vol. 97, 2019, pp. 2083–2092.

[16] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proceedings of the 18th Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 9351, 2015, pp. 234–241.

[17] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proceedings of the 31st Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 4805–4815.

[18] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proceedings of the 35th International Conference on Machine Learning, (ICML)*, vol. 80, 2018, pp. 5449–5458.

[19] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[20] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 30th Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 1024–1034.

[21] M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Proceedings of the 15th Extended Semantic Web International Conference (ESWC)*, vol. 10843, 2018, pp. 593–607.

[22] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2017, pp. 135–144.

Appendix

A Roles and Responsibilities

The main contribution of each member in our group is summerized as in Table 4:

Table 4: Contribution of group members

Member	Contribution
Ruiwen Zhou	Link prediction and its improvements.
Rui Ye	Node classification and data preprocessing.
Zhiyu Zhang	Node classification and ensemble learning scheme.