2020-Fall Data Mining
Course Project Final Report

AutoML for Graph Representation Learning

IIIS, Si Jiang

1 Introduction

Graph is a powerful data struture which can be used to model countless problems and phenomenon in real-world, ranging from traffic scheduling, recommendation system to knowledge graph. With state-of-the-art machine learning toolbox, people try to learn low-dimensional representation and extract type information for each node in the graph, which develops into a new domain called graph representation learning. With more and more powerful graph neural networks (GNN)[1] have been proposed, many challenge tasks such as node classification and connectivity prediction get very good results.

Automated machine learning (AutoML) is the process of automating the process of applying machine learning to real-world problems. AutoML covers the complete pipeline from the raw dataset to the deployable machine learning model and is proposed as an artificial intelligence-based solution to the ever-growing challenge of applying machine learning. In pratical, AutoML is a promising approach to lower the manpower costs of machine learning applications and has achieved encouraging successes in hyper-parameter tuning, model selection, neural architecture search, and feature engineering.[2]

In this course project, we combine these two problems together: using AutoML for Graph Representation Learning. With given total availabel time budget and graph-structured data which contains nodes with features and edges information, we provide the solution to graph representation learning problems autonomously.

2 Solution

My final solution to the AutoML for Graph Representation Learning problem is mainly divided into three parts: Feature Processing (implemented in **featureProcessing.py**), Model Selection (implemented in **modelSelect.py**) and Model Training (implemented in **model.py**'s train_predict part). The structure of the solution is shown in the Fig.1:

2.1 Feature Processing

As described in the introduction part, AutoML task needs to be finished within the given time budget. To save time for later model selection and model training part, it is important to reduce the number of features and only remain the most important features. We use lightGBM, which is a gradient boosting framework that uses tree based learning algorithms, to do the feature selection.[3, 4]. As a supervised learning model, input with the whole feature table and labels, lightGBM can figure out the importance of each feature. This process takes little time (about 5 seconds) and we select the most important 80 features.

To double check the correctness of feature selection process, we also use PCA algorithm. We set the variance percentage threshold to be 0.3 for a componet to be keeping, which averagely drops more than 80% of features. Then we combine the features selected by lightGBM and PCA for later training process.

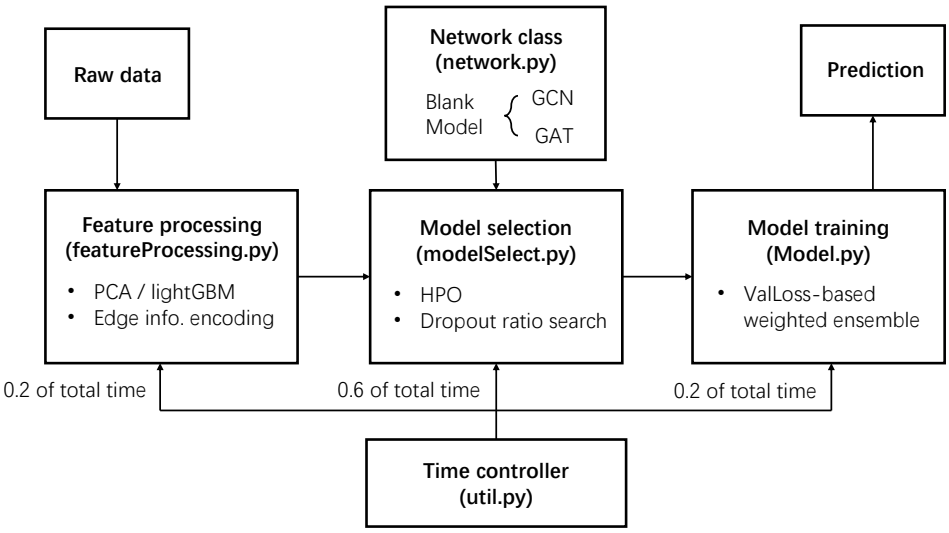

Fig. 1. The sturcture of the model. Feature processing selects important features from the raw feature table and directly encodes edge information into input. Model selection which compares different networks' performance on the validation set and selects the best network with certain parameters to do further training. Model training trains the selected network with full data set and do weighted ensemble. A time controller is used to balance time used in all three parts(0.2, 0.6, 0.2 of total time for each part), which guarantees the model return the prediction of test nodes within the time budget.

It should be noted that there are some cases that the raw data has few features (or even no feature, all information contain in edge data). In these cases, we do the direct edge information encoding as feature generation, which encodes the neighbors' label information as a feature. A concrete example is shown in Fig.2 to explain how this encoding works. Although networks used later consists of GCN convolution and GAT convolution which uses the egde index and weight as input, this direct encoding directly gives the model a strong priori knowledge for label prediction. However, this priori knowledge is not always true. To check the correctness of this priori knowledge, we compute the cross-entropy loss between the encoding and the true label. If the loss is smaller than a threshold, we add this encoding into the feature table.

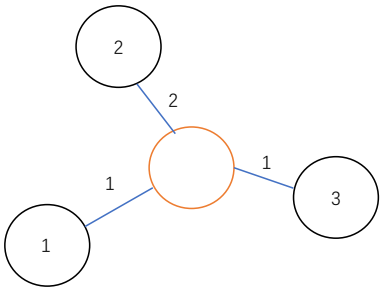

Fig. 2. A example for our direct edge information encoding. Suppose there are totally 3 labels, we need to do encoding for the central red node. The neighbors' labels and edge weights are marked in the figure. Then the edge information encoding for the red node is $Softmax([1, 2, 1]) = [0.212, 0.576, 0.212]$.

2.2 Network Class

network.py contains all network class we use. In particular, we use two kinds of graph networks: Graph Convolutional Network(GCN)[5] and Graph Attention Network(GAT)[6]. In our implementation, GCN and GAT are both inherited a abstract network class: BlankModel, which sets network type-irrelevant parameters and implements customized training and prediction process. For example, the training process can be configured by whether using validation set, whether halting the training when breakTime is used. To save the time for training and reduce the number of hyper-parameter in later searching, the max training epoch is set to be 1000 and we use early termination strategy: the training terminates early when validation loss estimated by EMA does not reduce any more.

The detail of BlankModel, GAT and GCN is shown in Fig.3. Since later we use HPO to search the best hyper-parameters, the parameters is saved in a args dictionary and pass to each network's init method. To save the memory, one can call resetMetaData method with new hyper-parameters, this method reset and hyper-parameters and randomly reset inner parameters of one existing network instead of creating a new one in the memory.

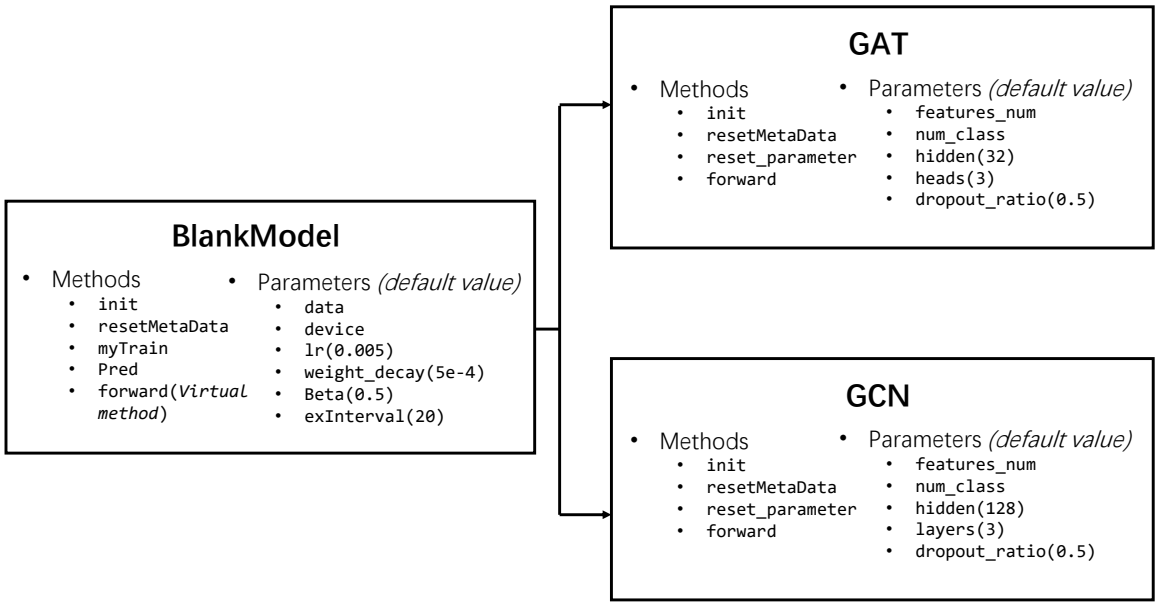

Fig. 3. The detail of BlankModel, GAT and GCN class. Both GAT and GCN classes are inherited from BlankModel class and BlankModel class is inherited from **torch.nn.Module** class. BlankModel contains the hyper-parameters for training. Beta is the EMA coefficient and exInterval is the check interval used in early termination. In GAT and GCN class, the network structure is defined in forward method, in particular, it shows important that we need to add batch-normalization layers. The hidden is the number of hidden neurons of each layer, layers in GCN means the number of convolution layers and heads in GAT means the number of heads in the first layer. The numbers in the parentheses are the value by default.

2.3 Model Selection

At the start of the model selection, the time controller first calls myModel class to train a GAT (since GAT is more time-consuming than GCN) with default parameter set, estimate the time for one training process. Depending on the remaining time, we either use hyper-parameter optimization (HPO) to find the

best model and parameters or use predetermined hyper-parameters, which make the networks have strong expressivity but easy to be overfitted, and do parameter searching only on dropout ratio.

2.3.1 Hyper-parameter optimization

If there is enough time to do multiple rounds of training, we search the whole hyper-parameter's space by using hyper-parameter optimization, which is a Python library for serial and parallel optimization over awkward search spaces[7]. By setting the loss of validation set as the target function, we search the best parameters in the space shown in Table.1. The remaining time is calculated by the time controller and passed to HPO, which guarantees that HPO halts before timeout.

Parameter	networkType	hidden	headslayers	dropout ratio	lr
Space	0,1	16(8)×randint(1, 9)	randint(1, 4)	Uniform(0.2, 0.8)	Uniform(1e-4, 1e-2)

Table. 1. The searching space for HPO. networkType is choosen from $\{0, 1\}$, which means either using GAT or GCN. hidden is a integer randomly choosen from $[1, 9)$. If the network is GCN, then multiply hidden by 16 and headslayers means the number of convolution layers. If the network is GAT, then multiply hidden by 8 and headslayers means the number of heads in the first layer. Dropout rate is uniformly choosen from $U[0.2, 0.8]$ and learning rate is uniformly choosen from $U[0.0001, 0.01]$

2.3.2 Dropout ratio search

If the model selection part's time only remains for a few rounds of training, then the most urgent task is to balance networks' expressivity and generalization ability with limited number of searching. Hopefully there is one hyper-parameter controls both these two abilities: dropout ratio. We first set other hyper-parameters to make networks have strong expressivity, then do dropout ratio search from 0.2 to 0.8 (Searching from small to large is reasonable since only large-scale problem leads to dropout ratio search part instead of HPO, and large-scale problem needs more expressive network). Time controller records the time for first round of training with dropout ratio equals to 0.2, then it determines the search interval and repeat times. Also, time controller stops the search when model selection part times out, returns the best network and parameters found up to now.

2.4 Model training

The network with selected hyper-parameters are further trained with all training set. We use validation loss-based weighted ensemble: which trains the network for multiple times, each time multiply the prediction by exponential of nagative validation loss and average them as the final prediction. This method can do unbiased estimation and effectively reduce the variance of prediction. The number of ensemble is determined by the time controller, once timeout signal received, the model calculates the label with max confidence on average prediction and output. Also, to save the time, all training process in the ensemble uses early termination.

3 Experience

- **Feature Processing**: In the midterm report, the feature processing is not controlled by the timer. We always do fully direct edge encoding, which takes a large amount of time when the graph has many edges (e.g. test c and d). After midterm, we strictly restrict time taken in this part by sampling a small part of edges. Also, we find the prior knowledge given by direct edge encoding is not always true, then we add cross-entropy loss checking. Besides PCA, we also apply lightGBM to do feature selection.

- **Network**: The network classes are totally rewritten after midterm, which integrates customized training and prediction process and fits into HPO frame. We find that batch-normalization layers are very important. We also try other GNNs like GraphSage, but these GNNs have worse performance than GCN and GAT on all test cases. To save time we give up these GNNs and only use GCN and GAT.
- **Model selection**: We add HPO as a powerful search tool when the time is enough. HPO can do heuristic search on large space with multiple hyper-parameters. The time control in this part also become more strict: any unfinished training process with timeout signal received will immediately halts. Also, when the time is not enough, we restrict the search only on dropout ratio, which takes only a few rounds of training and can get relatively good performance.
- **Model training**: In the midterm report, we use Bagging in the final training part. This is not a good idea. Bagging takes part of dataset and enlarge it by repeated sampling. However, in graph representation learning, all node and edge data constract a unique graph and cannot be dropped or repeated sampled. Instead, we use weighted ensemble, which is unbiased and can dramatically reduce the variance of prediction.

4 Performance

I run my model on a server using one NVIDIA TITAN V GPU, with random seed set as 1234, the performance of the baseline and my model on each test set is shown in Table.2

	Test a	Test b	Test c	Test d	Test e	Test demo
Baseline	0.8390	0.7110	0.8690	0.9361	0.8468	0.8477
My model	0.8526	0.7118	0.9447	0.9447	0.8861	0.8917

Table. 2. The performance on each test set with random seed 1234. The whole model runs on a single NVIDIA TITAN V GPU with 12GB memory. Both test c,d have large graph with more than 1,000,000 edges. The model chooses GAT on test a, e, demo. On test c and d, the model has no enough time to do HPO and only does dropout ratio search. Only test e use the direct edge encoding. There is not much improvement on test b since both GAT and GCN have not good performance on b and b's label distribution is more complex (for example, the cross-entropy loss between the direct encoding and the true label is larger than 3, which is the largest among all test cases)

5 Group Members And Contribution

There is no other group member and all works are finished by myself.

References

[1] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21, 2020.

[2] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 847–855, New York, NY, USA, 2013. Association for Computing Machinery.

[3] jean barbier, Mohamad Dia, Nicolas Macris, Florent Krzakala, Thibault Lesieur, and Lenka Zdeborová. Mutual information for symmetric rank-one matrix estimation: A proof of the replica formula. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 424–432. Curran Associates, Inc., 2016.

[4] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 3146–3154. Curran Associates, Inc., 2017.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[7] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. pages I–115 to I–23., 2013.