Towards Open-World Feature Extrapolation: An Inductive Graph Learning Approach

Qitian Wu, Chenxiao Yang, Junchi Yan

Abstract

We target open-world feature extrapolation problem where the feature space of input data goes through expansion and a model trained on partially observed features needs to handle new features in test data without further retraining.We propose a new learning paradigm with graph representation and learning. Our framework contains two modules: 1) a backbone network (e.g., feedforward neural nets) as a lower model takes features as input and outputs predicted labels; 2) a graph neural network as an upper model learns to extrapolate embeddings for new features via message passing over a featuredata graph built from observed data.Our experiments over several classification datasets and large-scale advertisement click prediction datasets demonstrate that our model can produce effective embeddings for unseen features and significantly outperforms baseline methods that adopt KNN and local aggregation.

Keywords： open-world, graph, feature extrapolation.

1 Introduction

Learning a mapping from observation x (a vector of attribute features) to label y is a fundamental and pervasive problem in ML community, with extensive applications spanning from classification/regression tasks for tabular data to advertisement click prediction[1], item recommendation[2], question answering[3] , or AI more broadly.Existing approaches focus on a fixed input feature space shared by training and test data. Nevertheless, practical ML systems interact with a dynamic open-world where features are incrementally collected. For example, in recommender/advertisement systems, there often occur new user profile features unseen before for current prediction tasks.

A challenge stems from the fact that off-the-shelf neural network models cannot deal with new features without re-training on new data. As shown in Figure1, a neural network builds a mapping from input features to a hidden representation through a weight matrix in the first layer. Given new features as input, the network needs to be augmented with new weight parameters and cannot map the new features to a desirable position in latent space if without re-training.

Our proposed framework contains two modules: a backbone network, which can be a feedforward neural network, and a graph neural network. The backbone network first maps input features to embeddings, and then, treats the observed data matrix as a feature-data bipartite graph that explicitly defines the proximity and locality structure between features and instances. Then, using the graph neural network for neural information transfer over neighboring features and instances in the latent space (abstraction), the embeddings of new features are computed inductively based on the embeddings of existing features, and the newly obtained embeddings capturing the semantic and feature-level relationships will be used to obtain hidden representations of new data with unseen features and to make final decisions.

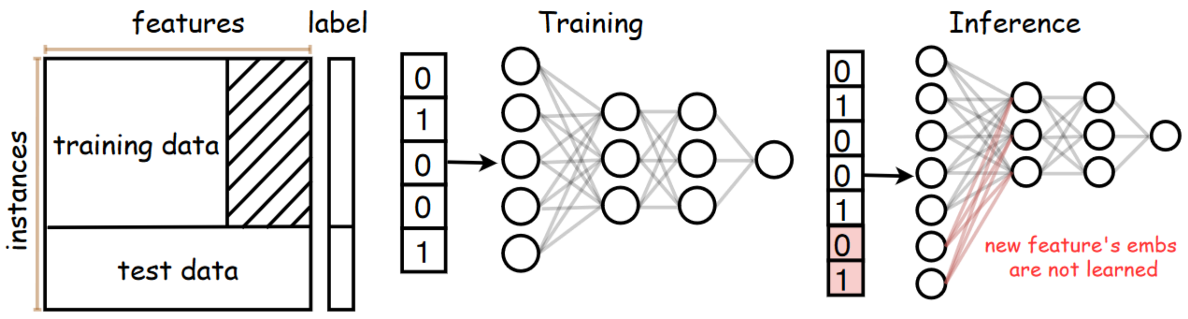


Figure 1: Limitation of neural network models for open-world feature extrapolation

To endow the model with ability for extrapolation to new features, we propose two training algorithms, one by self-supervised learning and one by inductive learning. The proposed model and its learning algorithm can easily scale to large-scale dataset (with millions of features and instances) via minibatch training.

2 Related works

Our introduced problem setting, open-world feature extrapolation (OFE), can be treated as an instantiation of out-of-distribution generalization or domain shift problem, focusing on distribution shift led by feature space expansion. To the best of our knowledge, this is a first-of-its-kind effort. Our work can be linked to other learning paradigm.

Domain Adaption adapts a model trained on source domain to target with different distribution[4]. Our problem OFE is different from DA in two aspects: 1) the label space/distribution of training and test data is the same for OFE, while DA often mostly considers different label distributions for source and target domains; 2) OFE focus on combining new features that are related to the current task, while DA considers different tasks from different domains.

Open-set Learning is another line of researches that relate with us. Differently, open-set recognition mostly focus on expansion of label sets[5]. To our best knowledge, we are the first to study feature set expansion, formulate it as OFE and further solve it via graph learning.

Zero-shot Learning. Our problem setting is also linked with few-shot/zero-shot learning. In NLP domains, some studies focus on dealing with rare entities exposed in limited times or new entities unseen by training[6].

3 Method

3.1 Proposed Model

Our model contains three parts: 1) feature representation that builds a bipartite feature-data graph from input data; 2) a backbone network which is essentially a neural network model that predicts the labels when fed with input data; 3) a GNN model that inductively compute features' embeddings based on their proximity and local structures to achieve feature extrapolation.

Feature Representation with Graphs. We stack the feature vectors of all the training data as a matrix $\mathbf{X}_{tr} = [\mathbf{x}_i]_{i \in I_{tr}} \in \{0,1\}^{N \times D}$ where $N = |I_{tr}|$ Then we treat each feature and instance as nodes and construct a bipartite graph between them. Formally, we define a node set $F_{tr} \bigcup U_{tr}$ where $F_{tr} = \{f_j\}_{j=1}^{D}$ with f_j the j-th feature and $I_{tr} = \{o_i\}_{i=1}^{N}$ with o_i the i-th instance in training set. The binary matrix X_{tr} constitutes an adjacency matrix where the non-zero entries indicate edges connecting two nodes in F_{tr} and I_{tr}, respectively. The induced feature-data bipartite graph will play an important role in our extrapolation approach. The representation is flexible for variable-size feature set, enabling our model to handle test data $\bar{\mathbf{x}_{i'}} \in \{0,1\}^{\bar{D}}$ which gives $\mathbf{X}_{te} = [\bar{\mathbf{x}_{i'}}]_{i' \in I_{te}}$. 0 **Backbo 0 ne Networks.** We next consider a prediction model $h_\theta(\cdot)$ as a backbone network that maps data features $\mathbf{x}_i$ to predicted label $\hat{y_i}$. Without loss of generality, a default choice for h_θ is a feedforward neural network. The first layer serves as an embedding layer which shrinks $\mathbf{x}_i$ into a H-dimensional hidden vector $\mathbf{z}_i = \mathbf{x}_i \mathbf{W}$ where $\mathbf{W} \in \mathbb{R}^{D \times H}$ denotes a weight matrix. The subsequent network (called *classifier*) is often a stack of neural layers that predicts label $\hat{y_i} = FFN(\mathbf{z}_i; \varnothing)$ .We use the notation $\hat{y_i} = h(\mathbf{x}_i; \varnothing, \mathbf{W})$ to highlight two sets of parameters and $\theta = [\varnothing, \mathbf{W}]$.

Notice that the matrix multiplication in the embedding layer is equivalent to a two-step procedure: 1) a lookup of feature embeddings and 2) a permutation-invariant aggregation. More specifically, we consider $\mathbf{W}$ as a stack of weight vectors $\mathbf{W} = [\mathbf{w}_j]_{j=1}^{D}$ where $\mathbf{w}_j \in \mathbb{R}^{1 \times H}$ corresponds to the embedding of feature f_j. The non-zero entries in $\mathbf{x}_i$ will index the corresponding rows of $\mathbf{W}$ and induce a set of embeddings $\{\mathbf{z}_i^m\}_{m=1}^{d}$ where $\mathbf{z}_i^m$ is the embedding given by $\mathbf{x}_i^m$ (i.e., one-hot vector of m-th raw feature). Then the hidden vector of i-th instance can be obtained by aggregation, i.e., $\mathbf{z}_i = \sum_{m=1}^{d} \mathbf{z}_i^m$ which is permutation-invariant w.r.t. the order of feature embeddings in $\{\mathbf{z}_i^m\}_{m=1}^{d}$. A more intuitive illustration is presented in Figure 2).

GNN for Feature Extrapolation. We proceed to propose a graph neural networks (GNN) model for embedding learning with the feature-data graph. Our key insight is that the bipartite graph explicitly embodies features' co-occurrence in observed instances, which reflects the proximity among features. Once we conduct message passing for feature embeddings over the graph structures, the embeddings of similar features can be leveraged to compute and update each feature' s embedding. The model can learn to extrapolate for new features' embeddings using those of existing features with locality structures in a data-driven manner. The message passing over the defined graph representation is inductive w.r.t. variable-sized feature nodes and instance nodes, which enables the model to tackle new feature space with distinct feature sizes and supports.

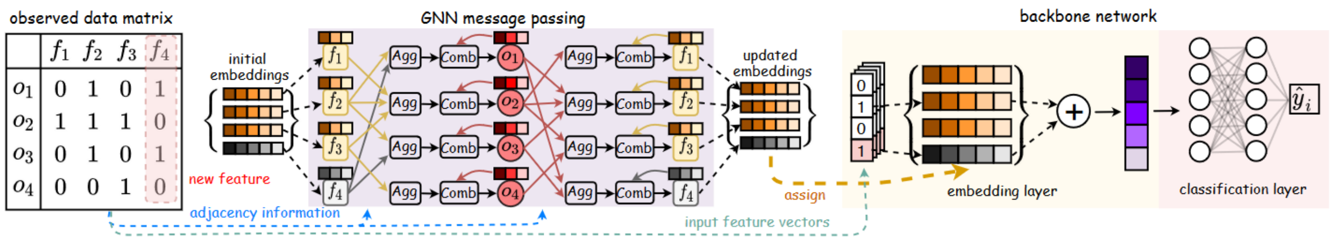
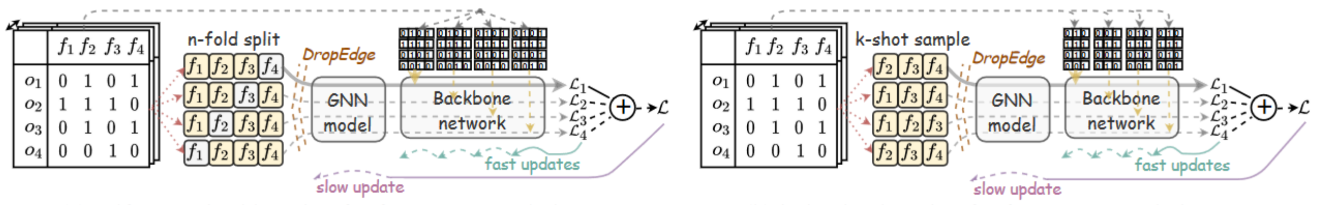


Figure 2: Feedforward of our model with input $\{\mathbf{x}_i\}$: we build a feature-data graph between feature nodes $\{f_j\}$ and instance nodes $\{o_i\}$. A GNN is used to inductively compute features' embeddings via message passing and a backbone network uses the updated embeddings to predict labels $\{y_i\}$.

3.2 Model Learning

We put forward two useful strategies. 1) Proxy training data: we only use partial features from training set as observed ones for each update. 2) Asynchronous updates: we decouple the training of backbone network and GNN network and using different updating speeds for them in a nested manner (see Figure 3). Based on these, we proceed to propose two specific training approaches.



(a) self-supervised learning for feature extrapolation (b) inductive learning for feature extrapolation

Figure 3: Illustration of two proposed training approaches: (a) self-supervised learning with n-fold splitting ($n = 4$), and (b) inductive learning with k-shot sampling $(k = 3)$. The two methods differ in configuration of proxy data, and both consider asynchronous updating rule: the backbone network is updated with n-fold proxy data before the GNN network is updated once with an accumulated loss.

4 Implementation details

4.1 Comparing with released source codes

Since the key code of the paper is already open source, we refactored the code of the model. It mainly includes, the preparation of dataset, the construction of bipartite graph and GNN, the construction of feedforward neural network and classifier. Compared with the open source code, our code is more brief and more readable.

4.2 Experimental environment setup

Implementation. We specify FATE in the following ways. 1) Backbone: a 3-layer feedforward NN. 2) GNN: a 4-layer GCN. 3) Training: self-supervised learning with n-fold splits. Several baselines are considered for comparison and their architectures are all specified as a 3-layer feedforward NN.

5 Results and analysis

Figure 4 shows the experimental results of the original paper on six data sets with observation proportions ranging from 30% to 80% on. Figure 5 shows the experimental results of our replicated FATE model with the same setup.

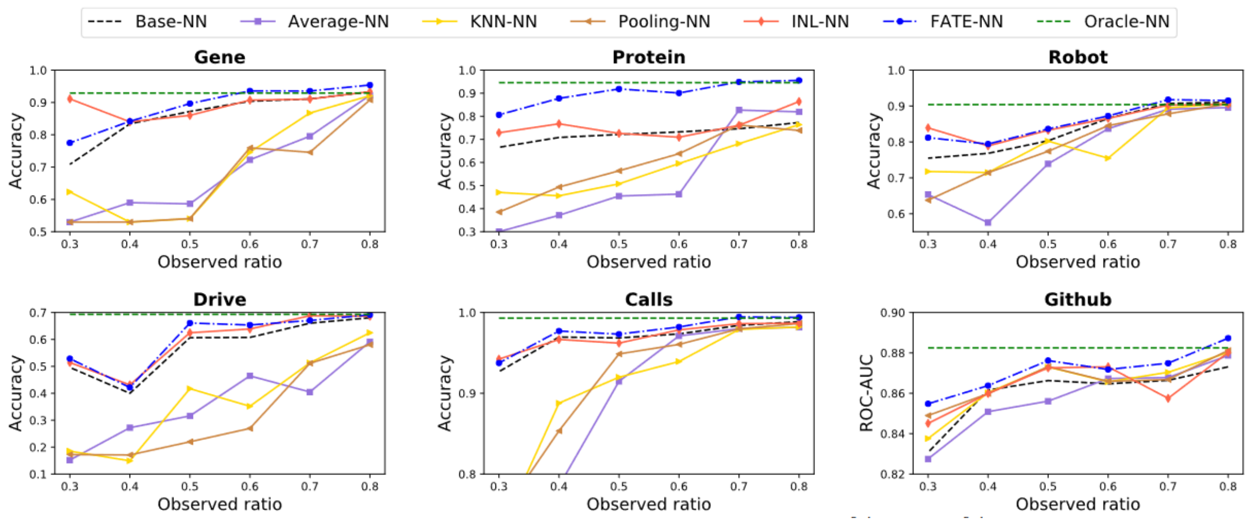


Figure 4: **Original paper result:**Accuracy/ROC-AUC results for UCI datasets with 30% 80% observed features for training. We run each experiment five times with different random seeds and report averaged scores.

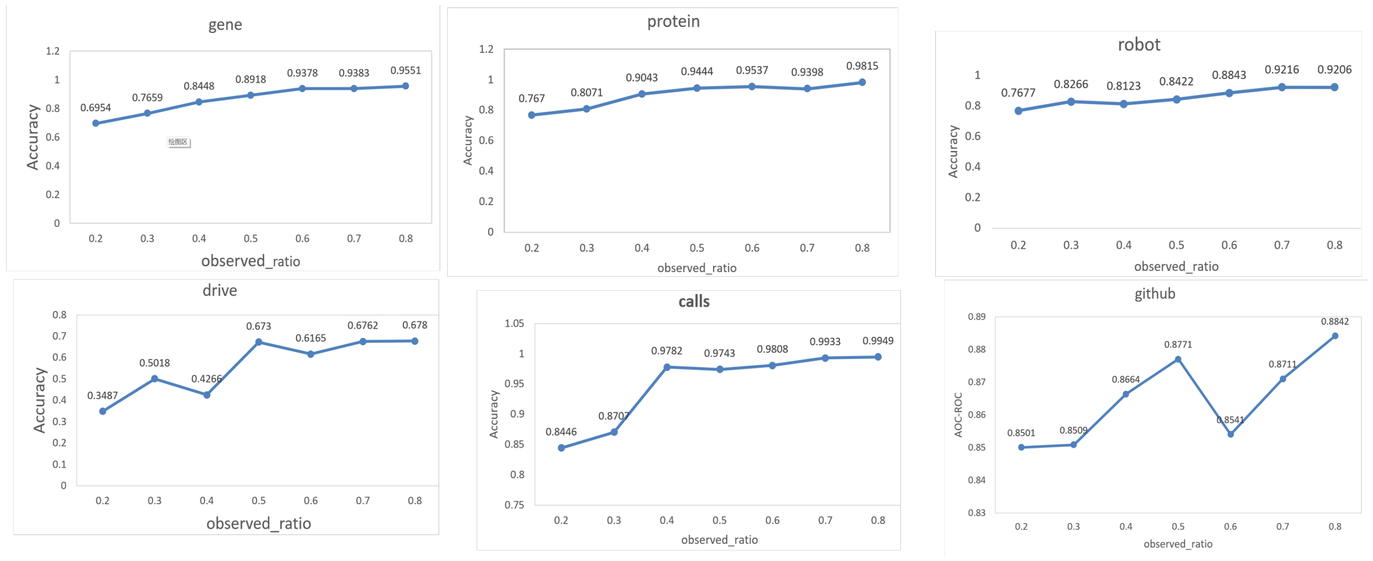


Figure 5: **Partial results of the replication:** Accuracy/ROC-AUC results for UCI datasets with 30% 80% observed features for training. We run each experiment five times with different random seeds and report averaged scores.

表 1: Comparison of experimental results

Dataset	Result	Accurate in different observed ratio					
		30%	40%	50%	60%	70%	80%
gene	paper	0.7659	0.8448	**0.8918**	**0.9378**	**0.9383**	0.9551
	my_result	**0.78**	**0.85**	0.88	0.93	0.93	**0.96**
protein	paper	0.8071	0.9043	0.9444	0.9537	0.9383	0.9815
	my_result	0.80	0.89	0.93	0.92	0.95	0.96
gene	paper	0.8266	0.8123	0.8422	0.8843	0.9216	0.9206
	my_result	0.81	0.79	0.93	0.83	0.87	0.92
gene	paper	0.5018	0.4266	0.673	0.6165	0.6762	0.678
	my_result	0.52	0.40	0.67	0.62	0.67	0.68
gene	paper	0.8707	0.9782	0.9743	0.9808	0.9933	0.9949
	my_result	0.93	0.97	0.96	0.97	0.99	0.99
gene	paper	0.8509	0.8664	0.8771	0.8541	0.8711	0.8842
	my_result	0.855	0.866	0.878	0.870	0.875	0.0.880

6 Conclusion and future work

We completed the replication of some experimental results of the original paper and obtained experimental results similar to those of the original paper. In the future, we will explore more applications of feature extrapolation in other fields.

References

[1] JUAN Y, ZHUANG Y, CHIN W S, et al. Field-aware factorization machines for CTR prediction[C]// Proceedings of the 10th ACM conference on recommender systems. 2016: 43-50.

[2] KOREN Y, BELL R, VOLINSKY C. Matrix factorization techniques for recommender systems[J]. Computer, 2009, 42(8): 30-37.

[3] XU M, JIN R, ZHOU Z H. Speedup matrix completion with side information: Application to multi-label learning[J]. Advances in neural information processing systems, 2013, 26.

[4] HIGUERA C, GARDINER K J, CIOS K J. Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome[J]. PloS one, 2015, 10(6): e0129126.

[5] CHEN W, CHANG M W, SCHLINGER E, et al. Open question answering over tables and text[J]. arXiv preprint arXiv:2010.10439, 2020.

[6] SCHICK T, SCHÜTZE H. Rare words: A major problem for contextualized embeddings and how to fix it by attentive mimicking[C]//Proceedings of the AAAI Conference on Artificial Intelligence: vol. 34: 05. 2020: 8766-8774.