CS224W Project: Graph-based node feature prediction system using structural features and incomplete node metadata

Yang Fang
yangfang@stanford.edu
Kaylie Zhu
kayliez@stanford.edu
Kylan Sakata
kylan@stanford.edu

December 11, 2019

Abstract

A myriad of recent machine learning endeavours are devoted towards missing node feature prediction using large-scale graphical datasets. In this paper, we experiment with the GNN, GraphSage, and GCN models using using node2vec embeddings and incomplete node and edge features from the Flickr dataset. We use ablation studies to explore the effectiveness of both portions of the input by running a series of experiments on both input forms on its own as well as combined. We also implement, train and tune each of the three models and compare their performance on image missing tag prediction. We find that GCN is best able to tackle the acute class imbalance deep-seated in the dataset and demonstrates most efficacious graphical structure learning in our research.

1 Introduction

With the explosive growth of social media platforms and e-commerce, massive graphical datasets have become increasingly prevalent and valuable, inspiring a plethora of algorithm research endeavours into node property prediction on graphical data in the field of machine learning. Many early efforts in the domain were hindered by the challenge of encoding graph structures to be easily exploited by machine learning models, and traditional machine learning approaches were reliant on user-defined heuristics to extract features encoding graphical structural information such as degree statistics. In recent years, however, we have witnessed a surge of deep learning techniques and nonlinear dimensionality reduction approaches, including matrix factorisation-based methods, random-walk based algorithms and graph convolutional networks [Hamilton et al., 2017], many of which have inspired us to build a graph-based image node feature prediction system using structural features and node metadata as follows. We implement and perform various experiments comparing GNN, GraphSage and GCN models on the Flickr dataset, before executing ablation studies to demonstrate the efficacies of using Node2Vec embeddings, incomplete node feature sets, and a concatenation of both in performing missing label prediction.

2 Task Definition

This project aims to leverage the graphical structure of the Flickr dataset to improve feature prediction. Specifically, given a target image, we would like to utilize the graphical relationships as well as node and edge features to perform a multi-class classification in order to predict any missing features or labels.

One natural extension and application of this task that was experimented with is in improving image recommendation, especially in the partial or full absence of node and/or edge features. For example, given partially or fully unlabelled edges or nodes (whether due to incomplete data, privacy requirements, etc.), a ”pre-recommendation” step could generate predictions for missing features, with its results being pipelined to traditional recommendation systems or networks for improved recommendation.

3 Related Work

Many modern machine learning models extract and leverage image features and metadata in order to accurately perform prediction and classification tasks, but most choose to independently process each image, despite the fact that the very co-existence of images within a collection (e.g. same author, same location) provides some inherently relational data that can be used to augment these models. In ”Image Labeling on a Network: Using Social-Network Metadata for Image Classification,”[Cheng et al., 2018] McAuley and Leskovec explore how this relational data can be modeled and harnessed to perform tasks such as image classification and labeling [McAuley and Leskovec, 2012]. They argue that by extension, a graphical model best captures the relationships between images. Using an image and metadata dataset collected from Flickr, McAuley and Leskovec found that the graphical model, compared against ”standard” machine learning approaches, generally produced significant performance improvements in prediction tasks. Interestingly, McAuley and Leskovec only take the approach of applying a graphical model to prediction tasks in an attempt to out-perform other machine learning models, and do not discuss the fact that there are certain tasks that are difficult or impossible to formulate as anything but a graph problem. For example, a graphical model can be leveraged to generate node groupings for a graph of fully or partially anonymized nodes and edges, where an anonymized edge indicates some unknown relationship between two nodes. This could lead to useful applications such as image recommendations in the event that there are legal regulations against retaining or accessing the actual metadata of images.

More recently, Sun et al [Sun et al., 2015] explores how to generate efficient and similar recommendations on a user-item basis using a social network to refine recommendations and give a better approach to interpreting the recommendations generated from close friends. They begin by using a biclustering algorithm, which is based on CTWC (Coupled Two-Way Clustering), which aims to group friends together to best help with their recommendations. Next, they interpolate collaborative filtering recommendations along with measures generated via the social network. If two users are in the same cluster, they use cosine similarity between the assigned tags to items of both users and otherwise use an interpolated cosine similarity formula. With this approach, Sun et al [Sun et al., 2015] were able to achieve improvements of by 25.88%, 9.26%, 19.95% and 39.63% relative to P@1, P@3, P@5 and R@5. This was tested on the Del.icio.us dataset [Zubiaga et al., 2013], which contains 437,593 <USER, URL, Tags> entries, 64,305 tags, 69,225 items (URLs) and 1867 users. This algorithm is highly scalable, as the overall runtime from clustering to running gradient ascent is $O(m\log n)$, where m is the number of stable user clusters and n is the number of stable item clusters.

Now, when it comes to scalability of graphical models on data with tremendous scale, Ying et al. [Ying et al., 2018] presents a large-scale deep recommendation engine with a data-efficient Graph Convolutional Network (GCN) algorithm PinSage that they developed and deployed at Pinterest. The novel model they introduce consists of efficient random walks and graph convolutions to generate node embeddings that incorporate both graph structure and node feature information. Their work is particularly remarkable due to the model's robustness, convergence and in particular high scalability and performance on a 7.5 billion example dataset with 3 billion nodes and 18 billion edges, culminating in the largest deep graph embedding application to date. By using on-the-fly convolutions, producer-consumer minibatch constructions, highly efficient MapReduce inference, convolution construction via random walks, importance pooling of neighboring node features and curriculum harder-and-harder examples training scheme [Ying et al., 2018], the paper achieves groundbreaking results for the magnitude of its scale. The model, whilst built on top of GraphSAGE, is modified to avoid operating on the entire graph Laplacian. Moreover, the limitation that the whole graph must be stored in GPU memory is overcome by using low-latency random walks to sample graph neighbourhoods in a producer-consumer architecture. The authors use quantitative performance comparisons with content-based deep learning baselines with hit-rate and Mean Reciprocal Rank (MRR) as well as qualitative comparison using user studies and production A/B tests [Ying et al., 2018]. The paper also cleverly circumvents the prohibitive nature of node2vec [Grover and Leskovec, 2016] and Deepwalk [Perozzi et al., 2014] as they cannot include node feature information and directly learn embeddings of nodes therefore entails linearity between number of model parameters and graph size.

One weakness spotted in the paper is that its means of sampling is prone to high variance in certain datasets, which is overcome in later research including FastGCN [Chen et al., 2018a] and Chen et al. [Chen et al., 2018b] where different sampling strategies are proposed to reduce variance and improve performance. More specifically, FastGCN [Chen et al., 2018a] samples nodes in each convlutional layer by interpreting nodes as i.i.d. samples and graph convolutions as integral transforms under probability measures. It also demonstrates that variance can be further reduced by sampling nodes via their normalised degrees. Chen et al., [Chen et al., 2018b] on the other hand, uses historical activations of nodes as a control variate, and allows for arbitrarily samll sample sizes [Zhang et al., 2018]. While our dataset is foreseeably much smaller than the billion-example Pinterest dataset and may not benefit as much from the magnitude of scalability, we are inspired by many of the introduced strategies to incorporate node features and metadata as well as improve computation efficiency, which is crucial given our resources, time and memory limitations.

4 Dataset

In this project, we explore the same Flickr image datasets [1] that McAuley and Leskovec use in their paper, "Image Labeling on a Network: Using Social-Network Metadata for Image Classification" [McAuley and Leskovec, 2012]. It is important to note, however, that each of these four datasets were collected and labelled using different methods, and thus differ, quite significantly in some cases, in certain characteristics - for the purposes of our project, we shuffle all photos before partitioning into training, validation, and test sets to best ensure a consistent distribution.

During pre-processing, we directly parse the XML of each dataset to obtain the metadata for each photo, and extract a subset of feature values for each photo. The currently extracted feature values include only metadata from potentially relational categories such as photo location, owner, and so on, but do not include feature values that likely pertain less to the actual photo itself, such as license and image format. This feature selection or feature engineering is slightly subjective, but is needed to limit our project scope and to account for limited computational resources. Next, we aggregate the set of extracted features for each photo and then filter the extracted features for each photo to only contain the top 2,000 most common feature values - this is needed to limit computational scope since, for example, there are over 450,000 unique values for just tags alone.

Finally, we build our project dataset by randomly removing each feature value of each node with some small probability (0.2 in this case). Then, the node features are embedded into feature vectors - i.e. an element is 1 if the node has the feature, or 0 otherwise; for some models, we also concatenate the corresponding generated Node2Vec embeddings to the feature embeddings for each node (discussed in section 6.1). Our final dataset consists of examples and labels of the partial-feature node embeddings and the true full-feature node embeddings, respectively. During model training and evaluation, the dataset examples are partitioned into train, validation, and test sets using a 70 / 20 / 10 split.

5 Baseline

As a reminder, the task that we are currently exploring is that, for a given photo, we would like to predict any missing feature values (i.e. photo labels). For our baseline algorithm, we take the approach of using a Naive Bayes classifier with Laplacian smoothing (specifically, add-one smoothing).

At a high level, using the photos in the training set, our implementation builds a dictionary of the total count of each co-existing pair of feature values (pairs of feature values that occur on the same photo) as well as the total count of each feature value. During evaluation (i.e. classification), for each test photo, we use these dictionaries as well as the test photo's remaining feature values to compute the conditional probability of each of the 2,000 possible features as a log likelihood, and then return the top n most likely features as the predictions (see below for the discussion on how to determine n). Any ties are broken by a secondary sort on the feature values (strings) themselves.

[1] http://snap.stanford.edu/data/web-flickr.html

We use exact match as a simple, binary correctness indicator - in other words, a classification is correct if and only if the combined set of the predicted labels as well as the input image's labels exactly matches the image's true set of feature values. Since this loss function definition dramatically increases or, in this case, maximizes the difficulty of the prediction task, we make a simplification and provide the exact number of missing labels as an input to the classifier (n in the paragraph above). This simplification reduces the problem space to a feasible dimension, but still leaves us with a non-trivial, meaningful task in trying to predict the most likely labels out of 2,000 possibilities. Formally, we define the baseline's classification loss to be:

$$L(i_{\text{test}}) = \sum_{f \in P(\text{test},n)} (f \notin \text{labels}(i_{\text{true}}))$$

Where i_{test} is the test image, n is the number of missing labels, $P(\text{test},n)$ is the predicted missing labels, and $(f \in i_{\text{true}}.\text{labels})$ is 1 if f is a true label or 0 otherwise. Note that we do not need to examine all of the photo's true labels since we are provided the exact number of missing labels for each test image. As discussed above, a test photo classification is correct if and only if $L(i_{\text{test}}) = 0$.

6 Methods

6.1 Methodology overview

As a first step, to capture graphical structure, we generated structural node embeddings using the node2vec algorithm (see section 6.2) for each node / photo, and then concatenated each node embedding along with its corresponding photo's extracted feature vector. This concatenation serves as an input to each of the Graphical Neural Networks (GNNs) that we build, train, and evaluate below; the three GNN models that were selected for this experiment were a simple 4-layer neural network, the GraphSage GNN model, and a Graphical Convolutional Network (GCN) model. Each of these models takes as input a photo feature vector with missing labels and/or a form of graphical structure information (e.g. node2vec embeddings, adjacency list) and learns to output a prediction of the missing features or, specifically, a prediction of the ground-truth full feature vector.

6.2 Feature extraction: node2vec

To train a set of embeddings on the graph, we use SNAP's *node2vec* program, retrieved from ttps://github.com/snap-stanford/snap/. We created 128-dim embeddings, the results of which are visualized in Figures 1 and 2 below. One problem we encountered was that the node IDs in the SNAP dataset correspond directly to Flickr's image IDs. Since many of these numbers contain 10+ digits, we were experiencing integer overflow with the SNAP library. To accomodate this, we assigned new IDs to each node, starting from 0 and maintaining the mappings to and from the IDs to these *indices*.

We've greatly improved our node2vec embeddings since the milestone, as the dot product of a pair of embeddings has been a good indicator of similar images. This time, we trained the new set of embeddings with dimension $d = 128$, walks per node $r = 10$, and walk length $l = 40$. After training a new set of embeddings on the Flickr graph, we tested them by taking random images and finding the most similar image by computing the dot product across all other nodes and sorting them.

Again, to visualize the node embeddings, we computed similarity scores for each pair of nodes, (n_1, n_2), as $v_1^T v_2$ where v_i is the embedding for node n_i. Since the underlying structure of the graph relates images by tag and label similarity, we hypothesized that images with similar node embeddings would be of similar content, style, and so on. As an illustration, we fetched the following query image from Flickr [de Wit, 2007] (See figure 1):

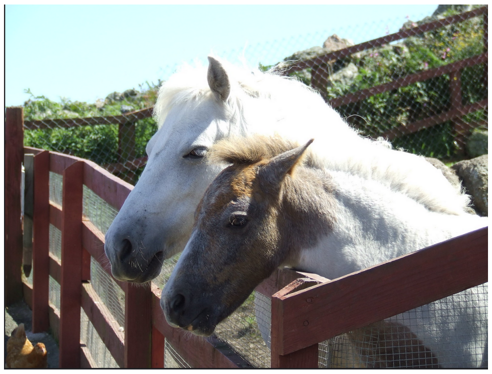

Figure 1: Original query image

The top 3 closest images returned by our initial embeddings are:

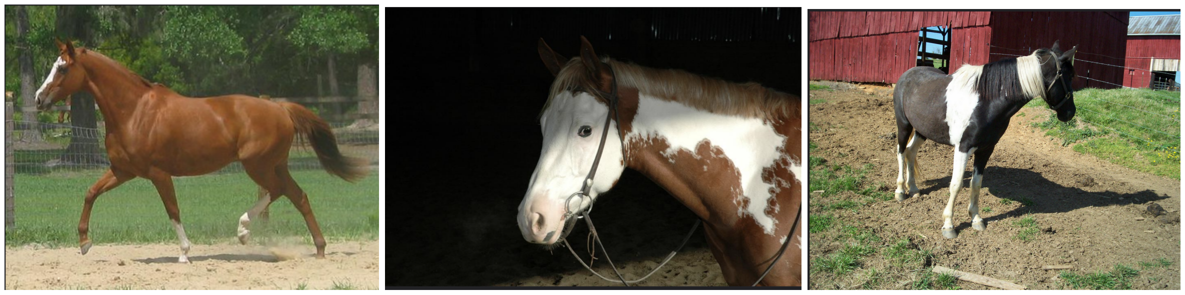

Figure 2: Retrieved top 3 images

Qualitatively, we see that this set of embeddings performs much better than our previous; all of our candidate images are all very similar to the query photo. The original picture, containing only two horses was chosen since it is relatively easy to tell if an image is a good recommendation or not: if it has a horse, and perhaps if the horse is the main focus of the image. Moreover, all 10 of the top 10 nearest images are also good recommendations of horses, showing a strong improvement from the milestone (not shown to save space). We tested this with other animals–pigeon and cat–and found similar, good results from these. We also tested this on a French building (inanimate object) and found that this returned good results as well in the form of other European buildings, but was not able to be as precise with the top images as the animal images, sometimes choosing images from Europe, just not of buildings.

6.3 Model architecture

6.3.1 GNN

We first created a 4-layer neural network to predict missing features. The layers are as follows:

$$Dense(128) \to Dropout(0.2) \to Dense(32) \to Dropout(0.2) \to Dense(8) \to Dense(2000) \to sigmoid$$

After each Dense layer except the last, we use a hidden LeakyReLU($\alpha = 0.3$) activation function. We chose to use two hidden dropout layers to regularize the weights, using a dropout probability of 0.2 and used an Adam optimizer with the default hyperparameter values. This model's output layer is a 2000-node, densely connected layer with a sigmoid activation for multi-label classification.

Since we are doing multi-label classification, we used a binary cross entropy loss function with a weighted variation. Since our feature data is very sparse (around 0.1% 1s, 99.9% 0s), we adjusted the loss function to penalize picking

0's for every prediction. In the new loss function, there is a large penalty for picking 0 when the ground truth label is a 1. To offset this, we also used a higher threshold for the sigmoid output to balance the new preference for picking 1s (0.95). We settled on a decaying learning rate, which is initialized at $\eta = 0.08$ and also used early stopping on our model to prevent overfitting the train set, stopping after about 50 epochs.

6.3.2 GraphSage

Next, we explored using a GraphSage GNN model to perform missing label prediction. First, we further pre-processed the dataset of examples and labels to construct an edge list, where an edge exists between two examples if and only if the two examples share at least one feature and are within the same partition (i.e. train, val, or test). Using this adjacency list as well as the training examples as input and the corresponding ground-truth feature vectors as labels, we built a GraphSage GNN model consisted of two message passing (i.e. aggregation and convolutional) layers using mean aggregations and convolutions, as illustrated in Figure 3 below. The model was trained with an ADAM optimizer, a learning rate of 0.1, a dropout of 0.5, and using the standard PyTorch binary cross-entropy loss with logits as an objective function.

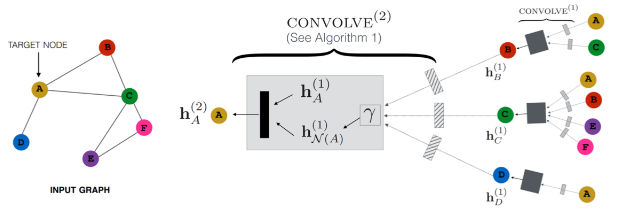

Figure 3: Architecture of a (two-layer) GraphSage GNN model.

6.3.3 GCN

We also explore using a Graph Convolutional Network [Kipf and Welling, 2016] (See Fig. 4) to predict the full feature labels for each node. GCN contains the word "convolutional" since it has filter parameters typically shared over all locations in the graph. We implement a GCN model that learns a function of features on the graph on our extracted data $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ which takes as input:

- a feature description x_i for every node i in the form of a $N \times D$ feature matrix X (N being number of nodes and D being number of input features)
- an adjacency matrix A that represents the graph structure

and produces a node-level output Z (an $N \times F$ feature matrix, F being the number of output features per node). For each neural network layer, we can write it as a non-linear function with a layer-wise propagation rule, for instance:

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)})$$

where $W^{(l)}$ is a weight matrix for the l-th neural network layer and $\sigma(\cdot)$ is a non-linear activation function, such as ReLU. We also add a tunable dropout layer (optimal at 0.1) in between the graph convolution layers where we set hidden dimension as 128 units. We use binary cross entropy with logits loss (weighted by class ratio, positive : negative = 9993:7), and Adam optimiser using 5e-4 weight decay and a starting learning rate of 0.01.

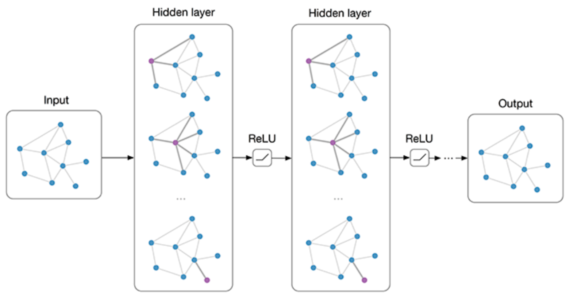

Figure 4: Multi-layer Graph Convolutional Network (GCN) with first-order filters.

7 Results and findings

7.1 Evaluation metrics

To evaluate a missing labels prediction, we will calculate the accuracy as the ratio of the total number of matching labels to the total number of all labels. Formally, let F_{pred} be the combined set of the input image's labels as well as the predicted labels, let F_{true} be the set of all of the image's true labels and let F_{all} be the set of all labels in the scope of consideration. Then, each missing label prediction accuracy can be calculated as:

$$acc = \frac{\sum_{f \in F_{\text{pred}}} f \in F_{\text{true}}}{|F_{\text{all}}|}$$

Since the dataset is rather imbalanced, we also use F1 Score, which is a weighted average of the precision and recall [Goutte and Gaussier, 2005].

$$2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Both metrics reach their best values at 1 and worst scores at 0.

For future extensions of closest neighbour recommendations, we can consider hit-rate as means of evaluating a closest image neighbour prediction [Ying et al., 2018]. For each positive pair of data instances (q,i) in the test set, we use q as a query instances before computing its top K nearest neighbors NN_q from a sample of test instances. Hit-rate is defined as the fraction of queries q where $i(i \in NN_q)$ was ranked among the top K of the test sample. And to determine the true closest neighbours, we can use Jaccard similarity to compute the differences between image metadata by [Johnson et al., 2015]:

$$d(x,x') = 1 - \frac{|t_x \cap t_{x'}|}{|t_x \cup t_{x'}|}$$

In the following subsections, we will present quantitative results of our final models and techniques, and relevant ablation studies, in comparison with baseline results on the same dataset and in a controlled environment.

7.2 Baseline results

Our baseline Naive Bayes classifier, as described in Section 5, was trained and evaluated on a dataset of 24,647 photos, with a feature value removal probability of 0.2. After filtering to only use the top 2,000 most common feature values, we counted 305,846 unique feature-value pairs, and over 1 million pairs in total. Evaluating on our test set of approximately 2500 photos, we achieved a test accuracy of approximately 35.8%. However, since the feature value removal probability was relatively small (0.2), many test images were missing 0 labels, and so there corresponding predictions were trivial in the sense that the input image was correctly returned as the fully labelled image. Accounting for this by filtering out the trivial test cases, we find that the non-trivial test accuracy in this case was around 8.7% on average.

While an accuracy of 8.7% may initially seem rather low, the image feature value prediction task is quite difficult, as there are 2,000 distinct possibilities and only a relatively small training set of less than 20,000 images. We expect the baseline classification accuracy to significantly increase with the introduction of the larger, full training set (for example, around 200,000 images). As is, we will still be able to derive some value from our baseline algorithm by using it as a qualifying metric for the performance of our project's model during development and iteration.

7.3 Model results

We tune and train all three models: GNN, GraphSage, GCN, and select the epoch with the lowest BCEwithLogits loss on the validation set, and report the accuracy and F1 scores on train, val and test test splits (See Table 1), and class-wise performances by GNN in Table 2 (whose training curve is plotted in Figure 5). We also do run ablation studies experiments using GCN using a) only node2vec embeddings; b) only incomplete features as inputs; and c) both node2vec embeddings and incomplete features as inputs (See Table 3 below).

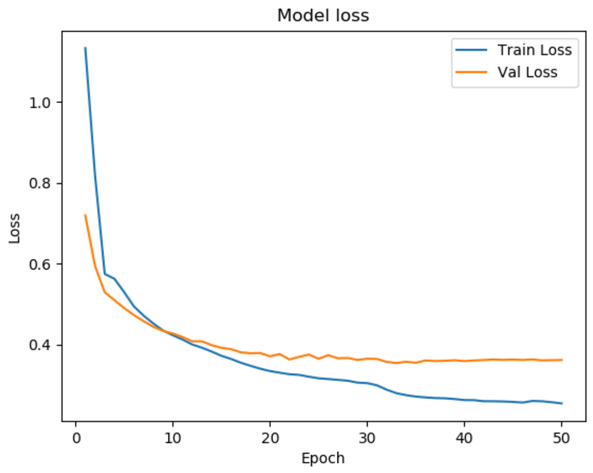

Figure 5: GNN Training Curve

Scores	GNN			GraphSage			GCN		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
Acc.	0.991	0.991	0.991	0.991	0.989	0.988	0.992	0.988	0.984
F1	0.991	0.991	0.991	0.990	0.989	0.988	0.921	0.887	0.813

Table 1: Results for using both node2vec embeddings and incomplete features as input

Scores	Class 0			Class 1		
	Train	Val	Test	Train	Val	Test
Acc.	0.996	0.996	0.997	0.505	0.425	0.176
Rec.	0.995	0.995	0.994	0.530	0.454	0.276

Table 2: Class-specific results for using GNN on both inputs

Scores	Only node2vec			Only incomplete			Both		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
Acc.	0.752	0.711	0.689	0.872	0.834	0.799	0.992	0.988	0.984
F1	0.703	0.664	0.613	0.842	0.808	0.777	0.921	0.887	0.813

Table 3: Results for using GCN on all 3 types of inputs (ablation studies)

8 Discussion and Analysis

Overall, training a neural network to perform multi-class classification predictions was difficult due to the sparsity of the data (each image tended to have only a very few features - on average, around 7 out of a possible 2000). As a result, our models tended to skew towards learning to make predictions of near-zero for all classes, which yielded extremely good accuracy. Out of the three models we built, the GCN was the most adept at actually learning graph structure and leveraging both feature and node embeddings to make predictions, rather than trivially predicting 0 for almost all classes; this can be evidenced by its relatively lower F1 scores, which indicate that it is making many (erroneous) non-zero classifications, whereas the high F1 scores for the other two models indicate a low number of non-zero predictions.

Another likely issue we observed - especially for both the GNN and GraphSage models - was the distribution of the train/val/test sets. The fraction of 1s (features) in the test set turned out to be 0.46%, less than half of the already infrequent 0.99% of 1s in the train set and 0.95% in the val set. This could explain the severe drop-off in accuracy and precision per-class in Table 2; by having much less 1s in the test set, the GNN model tends to predict more than this, producing a poor accuracy on 1s while increasing the accuracy of 1s in the train set. Even with 1s being so sparse, the model was still able to achieve 0.28 recall, so we see the potential in that the simple GNN model is able to reasonably be cautious predicting 1s but see that the sparsity makes high-accuracy and high-recall for 1s extremely difficult.

Many of the challenges we encountered were necessitated or magnified by the scale and sparsity of the dataset. For example, the large size and sparseness of the dataset (there are 450,000 unique values for just tags alone) required us to perform a significant amount of data preprocessing and filtering, in order to limit computational burden and complexity. A challenge that arose from the sparsity of features - i.e. each image tends to have a low number of tags and labels - was how to represent image features; we decided to encode features as 2000-dimensional feature vectors, for the lack of a better method, which was sub-optimal, considering the extreme sparsity of the vectors.

9 Conclusions

Inspired by the recent surge in machine learning endeavours devoted towards node feature and linkage prediction in large-scale graphical datasets, we implement GNN, GraphSage and GCN on the Flickr dataset using node2vec embeddings and incomplete node and edge features to predict full image node features. We trained and tuned each of the three models to compare their performance on image complete feature prediction.

We then use ablation studies to further explore the effectiveness of both portions of the inputs (trained node2vec embeddings and incomplete set of node and edge features) by running a series of experiments on them both separately and combined. Through ablation studies (see Table 3 for ablation studies using GCN model), we were able to directly examine the impact of the raw partial-feature vectors and the node2vec embeddings in improving model performance. We find that GCN is best able to tackle the acute class imbalance deep-seated in the dataset and demonstrates most efficacious graphical structure learning on our dataset. On the GCN model, we saw that using the graphical structure (captured using node2vec embeddings) in addition to the features yielded a 12% increase in missing label prediction accuracy, when compared against using features only. Given this significant improvement in performance, we may conclude that graphical convolutional model structure yields itself to make the relatively most significant improvements in the task of missing feature or label prediction.

10 Contributions

All members of the team contributed equally towards model, paper writing and research.

References

[Chen et al., 2018a] Chen, J., Ma, T., and Xiao, C. (2018a). Fastgcn: Fast learning with graph convolutional networks via importance sampling. *CoRR*, abs/1801.10247.

[Chen et al., 2018b] Chen, J., Zhu, J., and Song, L. (2018b). Stochastic training of graph convolutional networks with variance reduction. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 942–950, Stockholmsmässan, Stockholm Sweden. PMLR.

[Cheng et al., 2018] Cheng, Y., Tu, Z., Meng, F., Zhai, J., and Liu, Y. (2018). Towards robust neural machine translation. *CoRR*, abs/1805.06130.

[de Wit, 2007] de Wit, F. (2007). Tour de france.

[Goutte and Gaussier, 2005] Goutte, C. and Gaussier, E. (2005). A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In Losada, D. E. and Fernández-Luna, J. M., editors, *Advances in Information Retrieval*, pages 345–359, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653.

[Hamilton et al., 2017] Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584.

[Johnson et al., 2015] Johnson, J., Ballan, L., and Li, F. (2015). Love thy neighbors: Image annotation by exploiting image metadata. *CoRR*, abs/1508.07647.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

[McAuley and Leskovec, 2012] McAuley, J. J. and Leskovec, J. (2012). Image labeling on a network: Using social-network metadata for image classification. *CoRR*, abs/1207.3809.

[Perozzi et al., 2014] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652.

[Sun et al., 2015] Sun, Z., Han, L., Huang, W., Wang, X., Zeng, X., Wang, M., and Yan, H. (2015). Recommender systems based on social networks. *Journal of Systems and Software*, 99:109 – 119.

[Ying et al., 2018] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. *CoRR*, abs/1806.01973.

[Zhang et al., 2018] Zhang, Z., Cui, P., and Zhu, W. (2018). Deep learning on graphs: A survey. *CoRR*, abs/1812.04202.

[Zubiaga et al., 2013] Zubiaga, A., Fresno, V., Martinez, R., and Garcia-Plaza, A. P. (2013). Harnessing folksonomies to produce a social classification of resources. *IEEE Transactions on Knowledge and Data Engineering*, 25(8):1801–1813.