COMP90051 Statistical Machine Learning - Project Report

Team: VivaLaVida-Lockdown

Matthias Bachfischer *(1133751)* Parikshit Diwan *(1110497)* Liam Simon *(1128453)*

1 Introduction

In this report, we present five classification systems for the task of predicting links between users in online social networks: A traditional SVM classifier, two ensemble classifiers, a graph autoencoder (FastGAE) that makes use of a convolutional graph neural network (CGN) as well as a classifier built using the PyTorch BigGraph framework [1]. The goal of this paper is to compare the performance of these systems with respect to the problem domain of link prediction and evaluate the approach on the COMP90051 Kaggle competition.
All experiments reported in this paper were performed on a VM instance within the Google Cloud Platform running Debian 9 with 16 CPU cores and 60 GB of RAM. For training the deep-learning based classifiers we further leveraged a Tesla K80 GPU.

2 Data

Before training, we first download the training set from the COMP90051 Kaggle competition website. The training dataset is provided as a directed graph that contains an extract from the Twitter social network consisting of 4,867,136 nodes and 23,936,602 edges representing the relationship between different users on the platform.
The dataset was provided in the format of an edge list with each row denoting a node and its neighbours. Prior to performing any further processing steps, we first transform the dataset into a list of individual links with a (source,target) relationship as shown below:

Original training data:	1 2 3 4 5
Preprocessed training data:	(1,2), (1,3), (1,4), (1,5)

3 Data Processing

The problem of link prediction is well-studied within the research community and a variety of methods exist to solve this task. The following section provides an overview of the data sampling and feature extraction methods that were used for building the classifiers that are described in this report.
Disclaimer: Please note that a different data processing strategy was used for training the deep-learning classifiers FastGAE & BigGraph - these is discussed further below in section 4).

3.1 Data Sampling

In order to approach the problem of link prediction as a binary classification problem (in which the output of the classifier is the probability for a link between two nodes to exists), we use three different sampling strategies to extract samples from the edges that exist within the network *(positive edges)* and edges that do not exist within the network *(negative edges)*.

Random sampling

We first perform random sampling on the training set by randomly picking two nodes that have an edge in between them *(positive class)* as well as randomly picking two nodes from the training set that don't have an edge in between them *(negative class)*. The result of this processing step is a set of 100,000 edges on which we train our models (50,000 edges per class).

Importance sampling

The importance sampling method is nearly identical to the random sampling except that we use the measure of degree centrality in order to limit the selection of nodes to the 1 million most important nodes in the graph and randomly sample from this subset of nodes. By using this sampling strategy, we expect a performance improvement by reducing the number of bad samples (i.e. nodes with low number of neighbours).

Sampling on full dataset

The third strategy that we apply is to use the entire graph for learning and extracting features from the graph. This sampling strategy is only feasible with highly-optimized deep-learning based classifiers and is heavily intertwined with the respective model architecture. Details are provided further below.

3.2 Feature Engineering

For our experiments, we use the sampled dataset to calculate five metrics that indicate missing links between two node pairs (u, v) via a similarity score. The metrics that were used in this process are shown in table 1:

Common Neighbours (CN)	**Resource Allocation** (RA)	**Jacards' Coefficient** (JA)	**Adamic-Adar Score** (AA)	**Preferential Attachment** (PA)
$\lvert N(u) \cap N(v) \rvert$	$\sum_{t \in N(u) \cap N(v)} \frac{1}{\lvert N(w) \rvert}$	$\frac{\lvert N(u) \cup N(v) \rvert}{\lvert N(u) \cap N(v) \rvert}$	$\sum_{t \in N(u) \cap N(v)} \frac{1}{\log \lvert N(t) \rvert}$	$\lvert N(u) \rvert . \lvert N(v) \rvert$

Table 1: Overview of extracted features ($N(u)$ denotes the neighbours of node u)

After performing various experiments with our SVM and ensemble models using different combinations of features, we found that the preferential attachment score (PA) leads to overfitting. We subsequently removed this metric from the training data. Hence, the following feature set (in combination with standardization procedures) was used for training the SVM andthe ensemble classifiers: *CN, RA, JC, AA*.

4 System Description

To establish a baseline for the evaluation of our results, we use a SVM classifier as well as two ensemble methods that we train on the featurized dataset. Since it is not computationally feasible to train these types of classifiers on the entire graph network, we expect that we will be able to improve our performance by using more advanced, deep-learning based classifiers. This motivates us to also evaluate study the performance of the FastGAE model as well as the PyTorch BigGraph model on the link prediction task. A subset of the configuration parameters that we use to train the models during the Kaggle competition are shown in table 2.

4.1 Support Vector Machine (SVM)

To find the optimal combination of hyperparameters for our SVM (baseline) model, we utilize the Grid search procedure and arrive at the optimal parameter combination of $kernel = linear$ with regularization $C = 10$. We find that the model performs reasonably well on the given task but most likely suffers from overfitting because of the relatively small training set size on which it is trained on and which does not yet represent the sparsity of the original graph.

4.2 Ensemble Methods

We implement two ensemble methods and test their performance on the featurzed dataset by using the cross-validation mechanism provided by sklearn's 'RepeatedStratifiedKFold'.

4.2.1 AdaBoost

AdaBoost is an ensemble method that combines weak estimators to build a stronger classifier. By using the SVM classifier as the estimator and training the AdaBoost model on this configuration, we end up with a lower AUC score compared to the pure SVM model. We assume that this is again related to the limited sample size that in turn leads to overfitting during the boosting process.

4.2.2 LightGBM

LightGBM [2] is an implementation of a Gradient Boosting algorithm that uses histograms to bin features into discrete sections. We train two LightGBM classifiers using the data obtained from the random and importance-based sampling strategy, set the $objective = binary$ and $boosting = gbdt$ parameters and evaluate their performance. Again we achieve a similiar AUC score as with the previous models which is not surprising since the classifiers that we have described so far all rely on the same data during the training process.

Subpar performance of the models described above leads us to the conclusion that in order to get state-of-the-art results on the link prediction problem. we need to find better features that represent more information about the nodes and edges of the full graph (instead of a randomly sampled subset) and that we need to train our models on the full dataset that was provided by the competition organizers.

4.3 Fast Graph AutoEncoder (FastGAE)

In order to surpass the performance of the previously studied classifiers that were trained on manually-crafted feature embeddings, we leverage the FastGAE framework [3] for the generation of those embeddings. FastGAE is a highly scalable implementation for Graph Autoencoder (AE) models [4] and can be used to perform link prediction on the given dataset. Even though the FastGAE framework does not yet support directed graphs and does therefore not fit perfectly for the given problem set, we still observe a significant increase in performance with respect to the AUC score compared to previous models. We also observe that by increasing the sample size of the nodes that are processed per iteration $n_{(S)}$, we are able to improve the performance of the model even further.

Models	Parameters	AUC score	Remarks
SVM	$C = 10$ $Kernel = Linear$	0.75333	
ADA Boost	$Estimator = SVC$	0.71968	Same parameters used for SVC as with SVM model
LightGBM	$sampling = random$, $objective = binary$ $boosting = gbdt$	0.76458	Random sampling
	$sampling = importance$, $objective = binary$ $boosting = gbdt$	0.80321	Sampling by importance
FastGAE	$\eta = 0.01$, $\alpha = 1$, $n_{(S)} = 1,000$ $n_{iter} = 200$, $d_{embedding} = 16$	0.83029	Small number of nodes sampled per iteration $n_{(S)}$ - allows faster training but reduces quality of embeddings
	$\eta = 0.01$, $\alpha = 2$, $n_{(S)} = 10,000$ $n_{iter} = 200$, $d_{embedding} = 16$	0.84481	Large number of nodes sampled per iteration $n_{(S)}$ - leads to slower training but improves quality of embeddings
PyTorch BigGraph	$\eta = 0.001$, $loss = logistic$ $n_{epochs} = 50$, $d_{embedding} = 1024$	0.88553	Large embedding dimension $d = 1024$ and $logistic$ loss
	$\eta = 0.001$, $loss = ranking$ $n_{epochs} = 50$, $d_{embeddings} = 1024$	**0.93463**	Large embedding dimension $d = 1024$ and $ranking$ loss

Table 2: Results for link prediction task on test set (AUC score obtained from Kaggle competition)

4.4 PyTorch BigGraph

The PyTorch BigGraph system [1] supports the automatic learning of graph embeddings from large-scale graphs. It therefore fits well with the given task as it allows training on the full dataset within a reasonable amount of time. To train the PyTorch BigGraph system, we first create a training and validation set and configure various hyperparameters for the BigGraph system as shown in table 2. We then train the model with a learning rate $\eta = 0.001$ for $n_{epochs} = 50$ and store the generated embeddings on disk. To perform link prediction with the learned embeddings, we calculate the dot product for the node-pairs in the test set and pass these scores through the sigmoid function to transform the score into the probability that an edge between a given node pair exists.

5 Model Evaluation

The driving factor for why we prefer the BigGraph model over the other methods presented in this report is that, besides achieving a better AUC score, it also works by repeatedly sampling the whole data set during training. The embeddings that are obtained from the BigGraph model hence encode a very comprehensive view of the graph and the relationship between the nodes within the graph. Due to computational constraints, it is usually not feasible to train any of the other classifiers (e.g. SVM) on the whole dataset as those models are not optimized to deal with extremely large graph data. PyTorch BigGraph allows for embeddings to be created more quickly and easily than traditional methods and hence performs better than any of the other models that we previously tested.

6 Conclusion

In this report, we have shown how various classification systems can be used to perform link prediction in graph networks. We first developed classifiers based on traditional SVM and ensemble techniques and used them as a baseline to evaluate the performance of our predictions. Subsequently, we further improved our results in the task by leveraging two deep-learning based graph networks: FastGAE and PyTorch BigGraph. By using these models, we were able to achieve a final score of 93.46 AUC in the Kaggle competition.
Future work could use the embeddings that we obtained from training the PyTorch BigGraph model as features during the training process of a traditional classifier (e.g. SVM or LightGBM). This approach of re-using the graph embeddings for down-stream tasks like link prediction is common in the research community and usually leads to an additional increase in performance.

References

[1] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, "PyTorch-BigGraph: A Large-scale Graph Embedding System," in *Proceedings of the 2nd SysML Conference*, 2019.

[2] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157.

[3] G. Salha, R. Hennequin, J.-B. Remy, M. Moussallam, and M. Vazirgiannis, *FastGAE: Fast, Scalable and Effective Graph Autoencoders with Stochastic Subgraph Decoding*, Feb. 2020.

[4] T. Kipf and M. Welling, "Variational Graph Auto-Encoders," *ArXiv*, vol. abs/1611.07308, 2016.