Experimental Evaluation of Train and Test Split Strategies in Link Prediction

Gerrit Jan de Bruin[1], Cor J. Veenman[1,3], H. Jaap van den Herik[2], and Frank W. Takes[1]

[1] Leiden Institute of Advanced Computer Science (LIACS), Leiden University
g.j.de.bruin@liacs.leidenuniv.nl
[2] Leiden Centre of Data Science (LCDS), Leiden University
[3] Data Science Department, TNO

Abstract. In link prediction, the goal is to predict which links will appear in the future of an evolving network. To estimate the performance of these models in a supervised machine learning model, disjoint and independent train and test sets are needed. However, objects in a real-world network are inherently related to each other. Therefore, it is far from trivial to separate candidate links into these disjoint sets.
Here we characterize and empirically investigate the two dominant approaches from the literature for creating separate train and test sets in link prediction, referred to as random and temporal splits. Comparing the performance of these two approaches on several large temporal network datasets, we find evidence that random splits may result in too optimistic results, whereas a temporal split may give a more fair and realistic indication of performance. Results appear robust to the selection of temporal intervals. These findings will be of interest to researchers that employ link prediction or other machine learning tasks in networks.

Keywords: link prediction, performance estimation, machine learning

1 Introduction

Machine learning has emerged as a powerful instrument to analyze all kinds of datasets. Here, we focus on supervised learning, of which the use on non-relational (i.e., tabular) data is rather straightforward. However, supervised machine learning on network data is more challenging due to problems of obtaining an independent train and test set [1]. A common type of machine learning in networks is link prediction, where the goal is to predict whether a link will form in some future state of an evolving network. In this work, we focus specifically on the generalization of link prediction methods. Link prediction is defined as the problem where given the current state of a network, new edges between pair of nodes are inferred for the near future [2]. This method has many applications in different kinds of real-world scenarios, such as spam mail detection, friend recommendations in online social networks and identifying related references in a publication. In recent years, there has been an increasing interest in link prediction and hence several review papers on this topic exist [3,4,5].

A crucial first step in machine learning in networks is feature engineering, where network topology data is converted into features with potentially useful information for a predictive model. The main established approaches for feature engineering in link prediction are based on similarity, probabilistic and maximum likelihood, and dimensionality reduction [3]. We will focus on the similarity-based approach. In this approach, pairs of nodes (candidates for links formed in the future) are assigned scores according to their similarity. We will exclusively use topological properties to assess similarity, such that we can apply the feature engineering also to networks where no additional information is available about the nodes. The similarity-based approach provides at least three benefits. First, similarity-based features provide more accurate results compared to embedding techniques [6]. Second, the similarity-based approach provides easily explainable features compared to other approaches. Third, most features can be obtained at relatively low computational costs for the larger networks used in this study.

This brings us to the main problem addressed in this paper. For proper validation in any machine learning task, instances belonging to the training set, on which the model is trained, should be disjoint and independent of features belonging to the validation and test set. However, because many dependencies exists between nodes in a network, this is inherently difficult to achieve. This possibly results in too optimistic performance measurements or, equivalently, overestimating the so-called generalization performance of the model [7]. According to Ghasemian *et al.* it is yet unclear how common machine learning steps, such as cross-validation and model selection methods, extend from non-relational to network data [8].

Assessment of the performance in supervised machine learning is important for at least two reasons. The first is model selection. Different models can be constructed for a certain task, ranging from completely different classifiers to identical models with different (hyper)parameters. The performance of such models for train data is not informative, as one wants the model with the best generalization performance on an independent test set. An independent validation set allows the detection of overfitting. The second reason for assessing model performance is to estimate the prediction error on new, unseen data. This should be assessed using the test data, not used in any part of training the model and neither used in choosing the right hyper-parameters or selecting a model [7]. In the current research we investigate to what extent differences in collecting the train and test set influence the generalizability score of the classifier.

The contributions of this work are as follows. First, we investigate the two most common ways in which pairs of nodes can be split in disjoint and independent train and test sets in link prediction. Second is an in-depth comparison of these two approaches, on a number of evolving real-world networks. We contribute to a better understanding of performance estimation in link prediction.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. We continue with definitions and our approach towards reporting generalization performance in Section 3. Section 4 features information about the datasets used. Then, Section 5 is concerned with the experimental setup, results and discussion. Conclusions and future work are provided in Section 6.

2 Related Work

There is a relatively small body of literature that is directly concerned with splitting a network dataset into train and test set to evaluate the performance for machine learning purposes. Hence, we start our exploration of the literature with work on performance estimation in general, before focusing specifically on prediction tasks in networks.

One of the causes of too optimistic performance estimation is what is often described as "test set re-use" [9]. A well-known example is the p-hacking problem [10]. In short, p-hacking is the application of many different models to the same data in search for a statistically significant result with a high enough p-value. This misuse can result in increasing probability that applied research findings are false. More specific to data-driven research, too optimistic performance estimation is suspected in Kaggle competitions. In these online competitions participants all get the same dataset and compete for the best classifier performance on some predictive task, without having access to the test data. However, Kaggle allows users to repeatedly probe test data to obtain the performance of a submitted model. This is argued to lead to too optimistic results [11], which was experimentally only observed to a limited extent [9].

Returning more specifically to the topic of machine learning on networks, Ghasemian et al. [8] investigated under- and overfitting in networks. They did so in an attempt to estimate the performance of various community detection algorithms. The performance on link prediction and so-called link description task are used as a diagnostic to evaluate the general tendency of such algorithms to under- and overfit. The authors define the link prediction task a little differently, since they do not necessarily have temporal information about the edges. Hence they remove a fraction of edges from a network and employ a machine learner to find these removed links from all pairs of nodes that are not connected anymore. The link description problem is different. Again a network is sampled, but now the task for the machine learner is to find the remaining edges of the sampled network from all pairs of nodes. The authors explain that no algorithm can excel at both the link prediction and link description task and that these two tasks force an algorithmic tradeoff, like the bias-variance tradeoff in non-relational data [7]. In this work, we likewise want to bring the notion of overfitting from non-relational data to relational data. While [8] focusses on overfitting caused by the bias-variance tradeoff, we investigate too optimistic estimation of generalization performance caused by test set reuse in networks.

3 Approach

This section will start with a formal description of the link prediction problem. In Section 3.2 we explain how we split the data into a train and test for the link prediction classifier. Section 3.3 continues with the features used. Section 3.4 provides information about the used classifier. Finally, in Section 3.5 we explain the performance metrics used.

3.1 Link prediction problem

We base our procedure of supervised link prediction in evolving networks upon definitions used by Liben-Nowell et al. [2], Lichtenwalter et al. [12], and Kumar et al. [3]. To ensure uniformity, we use sometimes slightly different terminology than aforementioned works.

The temporal, potentially undirected, network $G = (V, E)$ consists of a set of nodes V and edges $(u, v, t) \in E$ connecting nodes $u, v \in V$ with time $t \geq t_0$. Time t_0 indicates the time of the first edge occurring in G. Parallel edges with different timestamps can exist. Since the network is temporal, we can construct snapshots of network G for a given time interval. We denote such a snapshot with $G_{[t_a,t_b]} = (V_{[t_a,t_b]}, E_{[t_a,t_b]})$ with $E_{[t_a,t_b]}$ being a set consisting only of edges occurring between t_a and t_b (with $t_a < t_b$) and $V_{[t_a,t_b]}$ the nodes taking part in these edges. We make two such snapshots, $G_{[t_a,t_b]}$ and $G_{[t_b,t_c]}$ from two time intervals $[t_a, t_b]$ and $[t_b, t_c]$ with $t_a < t_b < t_c$. This procedure is shown in Figure 1a.

The task for the supervised binary link prediction classifier (see Section 3.4) is to predict from $G_{[t_a,t_b]}$ whether a pair of nodes will connect in $G_{[t_b,t_c]}$. Hence, the input for the classifier is all pairs of nodes $X_{[t_a,t_b]} = \left(V_{[t_a,t_b]} \times V_{[t_a,t_b]}\right) \setminus E_{[t_a,t_b]}$, see also Figure 1b. The network $G_{[t_a,t_b]}$ needs to be "mature" enough that the underlying static topology is well captured [12] and hence we call $[t_a, t_b]$ the *maturing interval*. Subsequently, we define the *probing interval* $[t_b, t_c]$. For every pair of nodes $x_i \in X_{[t_a,t_b]}$, we probe whether the pair is present in the probing interval (indicated with $y_i = 1$) or not (indicated with $y_i = 0$). The entire procedure is summarized in Figure 1.

3.2 Splitting strategies

Now that we described the general procedure of link prediction, we need a strategy to separate the pairs of nodes into a train and test set for the classifier. The classifier is learned on the train set and the performance is determined on the test set. We will now explain two dominant ways encountered in literature to split the dataset. While the procedure of applying a temporal split is more complicated than the random split due to the various parameters, it prevents to a greater extent the reuse of node and edge set information from the test set in training.

Random split This procedure for example used in [12], consists of three steps, which are also shown in Figure 2a. The first step is to obtain all pairs of nodes that are not connected during the maturing phase, $X_{[t_0,t_b]}$. Second, we determine for each of these pairs of nodes whether they connect (the value of y_i) in the

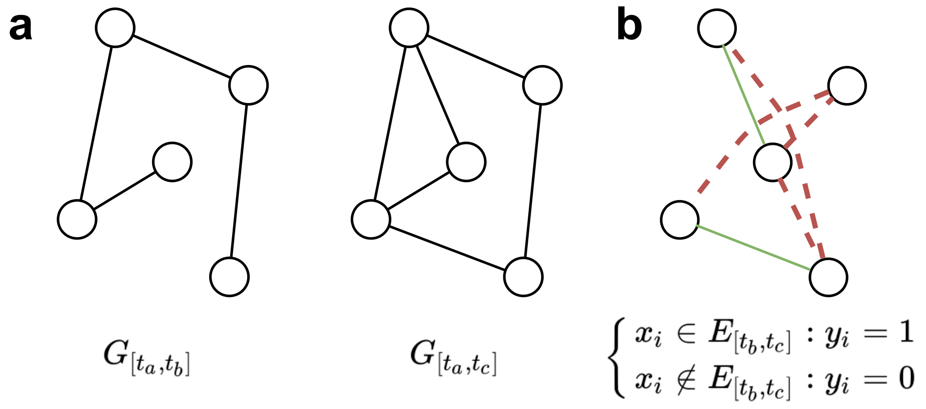
a
$G_{[t_a,t_b]}$
$G_{[t_a,t_c]}$
b
$\begin{cases} x_i \in E_{[t_b,t_c]} : y_i = 1 \\ x_i \notin E_{[t_b,t_c]} : y_i = 0 \end{cases}$

Fig. 1. (a) The evolution of a temporal network divided into different snapshots. (b) Instances considered in the classifier. Positive instances ($y_i = 1$) are shown in green solid lines, while negatives ($y_i = 0$) are shown in red dashed lines.

probing phase $E_{[t_b,t_c]}$, as shown in Equation 1.

$$y_i = \begin{cases} 1 & \text{if } x_i \in E_{[t_b,t_c]} \\ 0 & \text{if } x_i \notin E_{[t_b,t_c]} \end{cases} \text{ for } x_i \in X_{[t_0,t_b]} \tag{1}$$

Third, these pairs of nodes $X_{[t_0,t_b]}$ are separated into into two disjoint sets $X^{\text{train}}_{[t_0,t_b]}$ and $X^{\text{test}}_{[t_0,t_b]}$ such that $X^{\text{train}}_{[t_0,t_b]} \cup X^{\text{test}}_{[t_0,t_b]} = X_{[t_0,t_b]}$ and $X^{\text{train}}_{[t_0,t_b]} \cap X^{\text{test}}_{[t_0,t_b]} = \emptyset$. We will refer to this split procedure as the *random split*, as the train and test set are taken at random from the instances in $X_{[t_0,t_b]}$.

Temporal split A different procedure is for example used by Hasan et al. [13], which we will call *temporal split*. In this procedure, the train and test set are made by applying the probing phase on two different, consecutive snapshots called the training interval $[t_0, t_c]$ and test interval $[t_0, t_d]$. The four steps of this process are shown schematically in Figure 2b. The training set is constructed in the first two steps as follows. First, we consider all node pairs that are not connected in the maturing phase of the train interval $X_{[t_0,t_b]}$. Second, for each of these node pairs we determine whether it will connect in the probing phase of the train interval, like Equation 1. In steps three and four the test set is constructed in a similar way as the train set. In step three, we consider all pairs of nodes $X_{[t_0,t_c]}$. Finally, in step four we determine for each of these pairs of nodes in whether they connect in the probing phase of the test interval, as shown in Equation 2.

$$y_i = \begin{cases} 1 & \text{if } x_i \in E_{[t_c,t_d]} \\ 0 & \text{if } x_i \notin E_{[t_c,t_d]} \end{cases} \text{ for } x_i \in X_{[t_0,t_c]} \text{ and with } t_c < t_d \tag{2}$$

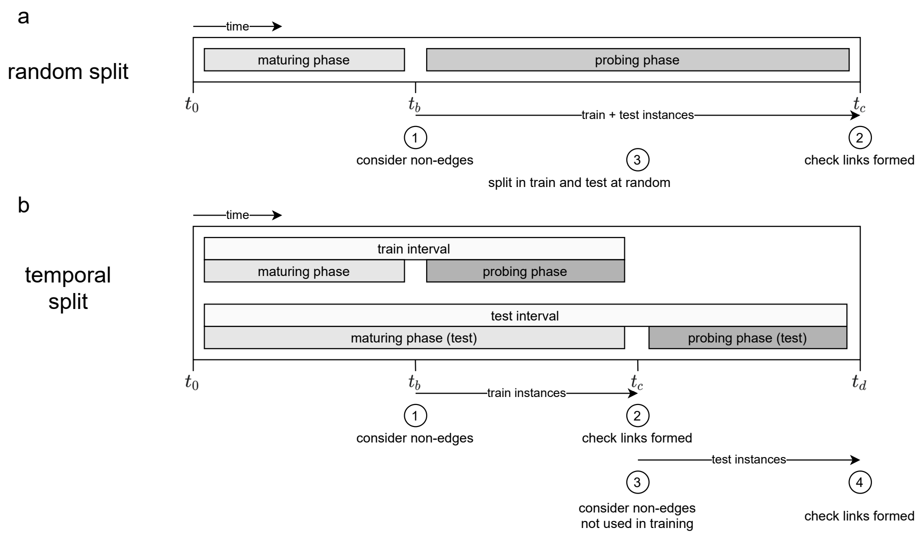

Fig. 2. Different strategies to obtain train and test set for classifier f, discussed in Section 3.2. a) Train and test set are obtained by randomly splitting instances from a single probing phase. b) Train and test set are obtained by two consecutive probing phases obtained from two different time intervals.

3.3 Features

As input for a classifier one needs a feature representation for all pairs of nodes $x_i \in X$. As discussed in the introduction, in this work we use the well-established similarity-based approach, where the feature for each pair of nodes $x_i = (u, v)$ consists of a particular score for each feature $S_{\text{feature}}(u, v)$. These scores are based solely on topological properties intrinsic to the network itself and not on any contextual information [12,14]. Hence, features used can be employed in any network, without requirements on node information available. Nodes with similar scores and hence a high similarity are then more likely to connect. The score is either neighbor-based (similarity in local properties of the two nodes) or path-based (quasi-local or global properties of the two nodes) [3,15]. We use the so-called HPLP feature set defined in [12], as these are known to obtain good performance while keeping the number of features limited.

In the definitions that follow, the set $\Gamma(u)$ denotes the neighbors of node u and $\deg(u)$ the degree of node u. Parallel edges can exist in the network, and hence the number of neighbors of node u, $|\Gamma(u)|$, is not necessarily equal to the degree of node u, $\deg(u)$. In directed networks, we differentiate between the neighbors connecting to node u, indicated by $\Gamma_{\text{in}}(u)$, and the neighbors node u connects to, $\Gamma_{\text{out}}(u)$. Likewise, we differentiate also between the indegree and outdegree of node u, $\deg_{\text{in}}(u)$ and $\deg_{\text{out}}(u)$, respectively.

Neighbor-based features Neighbor-based features take only the direct neighbors of the two nodes under consideration into account.

Number of neighbors (NN) This feature is determined differently for undirected and directed networks. For directed networks, we use both the number of neighbors connecting to nodes u and v and the numbers of nodes connected by u and v. Hence, we get four features: $S_{\text{NN-in-}u}(u,v) = |\Gamma_{\text{in}}(u)|$, $S_{\text{NN-in-}v}(u,v) = |\Gamma_{\text{in}}(v)|$, $S_{\text{NN-out-}u}(u,v) = |\Gamma_{\text{out}}(u)|$, and $S_{\text{NN-out-}v}(u,v) = |\Gamma_{\text{out}}(v)|$. In the undirected case, the same score for pairs of nodes (u,v) and (v,u) is desired and there is no difference between the number of nodes connecting from or to node u. Hence we report both the maximum and minimum for a given pair of nodes, i.e., $S_{\text{NN-min}}(u,v) = \min(|\Gamma(u)|, |\Gamma(v)|)$ and $S_{\text{NN-max}}(u,v) = \max(|\Gamma(u)|, |\Gamma(v)|)$.

Degree (D) The degree feature is defined similarly as the number of neighbors, except that the number of edges is considered instead of the number of nodes connected. For directed networks, we obtain again four features, viz. $S_{\text{D-in-}u}(u,v) = \deg_{\text{in}}(u)$, $S_{\text{D-in-}v}(u,v) = \deg_{\text{in}}(v)$, $S_{\text{D-out-}u}(u,v) = \deg_{\text{out}}(u)$, and $S_{\text{D-out-}v}(u,v) = \deg_{\text{out}}(v)$. For undirected networks, we obtain the maximum and minimum degree of nodes u and v, $S_{\text{D-min}}(u,v) = \min(\deg(u), \deg(v))$ and $S_{\text{D-max}}(u,v) = \max(\deg(u), \deg(v))$.

Common Neighbors (CN) The number of common neighbors for a given pair of nodes is calculated by $S_{\text{CN}}(u,v) = |\Gamma(u) \cap \Gamma(v)|$. For directed networks, the score is calculated by considering the nodes that are connected from nodes u and v, i.e. $S_{\text{CN}}(u,v) = |\Gamma_{\text{out}}(u) \cap \Gamma_{\text{out}}(v)|$.

Path-based features Path-based features take into account the paths between the two nodes under consideration. Since many paths can exist, these features are computational more expensive than the neighbor-based features.

Shortest Paths (SP) This measure $S_{\text{SP}}(u,v)$ indicates the number of shortest paths that run between nodes u and v.

PropFlow (PF) The PropFlow measure, $S_{\text{PF}}(u,v)$, corresponds to the probability that a restricted random walk starting from node u ends at node v within l steps [12]. We use the commonly used value of $l = 5$. We collapse the network with multiple edges (occurring at different timestamps) to a weighted network where the weight is equal to the number of parallel edges running between two nodes. Higher weights result in a higher transition probability for the random walk. This method is known to potentially obtain different scores for pairs of nodes (u,v) than for (v,u), even in the undirected case [16]. Hence, we use the mean of the scores obtained for the pairs of nodes (u,v) and (v,u) in the undirected case.

3.4 Classifier

We used a tree-based gradient boost learner for our classifier, as these are known to perform well in generic classification tasks. The Python implementation of XGBoost was used [17]. This classifier has various hyperparameters. While extensive hyperparameter tuning is beyond the scope of this paper, we cross-validate two important hyperparameters, viz. maximum depth of tree and class weights.

3.5 Performance metric

Link prediction is associated with extreme class imbalance, lower bounded by the number of nodes in the network [12]. Ideally, performance metrics used to evaluate the classifier, should be robust against this class imbalance. The commonly encountered Receiver Operator Characteristic (ROC) lacks this robustness [18,16] and is hence not used. We are especially interested in correctly predicting positives without loosing precision, i.e., keep the number of false positives low, and without loosing recall, i.e., make sure we find all true positives. The Average Precision (AP) metric, which is equal to the weighted mean of precisions achieved at each threshold in the precision-recall curve, is well-suited in this case.

4 Data

Since our research aims to split the network into different snapshots based on time, temporal networks are needed. In this work, we use six different temporal networks, spanning a broad range of different domains. Properties of these networks are shown in Table 1. The density, diameter Ø and mean distance $\bar{d}$ were calculated on the underlying static network, i.e., the network without parallel edges. Below, we briefly discuss the six datasets used in this work. Except from the Condmat network, all datasets were obtained from KONECT [19].

AU The Ask Ubuntu (AU) network is an online contact network. Interactions were gathered from the StackExchange site "Ask Ubuntu". The nodes are the users, and a direct edge is created when a user replies to a message of another user. These interactions can consist of an answer to a question of another user, comments on another user's question, and comment on another user's answer. Each edge is annotated with the time of interaction.

Condmat This scientific co-authorship dataset entails condensed matter physics collaborations from 1995 to 2000, obtained from https://github.com/rlichtenwalter/LPmade. A temporal undirected network is created by adding a node for each author in a publication and adding an edge between all authors of a publication [18]. For each edge, the date of the publication connecting these

Table 1. Statistics of networks used in this study. Edges and nodes in giant component (GC) are indicated between brackets. Mean distance between nodes is given in column $\bar{d}$ and the column Ø indicates the diameter of the networks.

dataset	directed	nodes (gc)	edges (gc)	density	$\bar{d}$	Ø
AU	✓	159,316 (96%)	964,437 (100%)	4.0×10^{-5}	3.9	13
Condmat	✗	17,218 (88%)	88,090 (100%)	3.7×10^{-4}	6.3	19
Digg	✓	30,398 (98%)	87,627 (100%)	1.9×10^{-4}	4.7	12
Enron	✓	87,273 (97%)	1,149,072 (100%)	7.9×10^{-5}	4.9	14
Slashdot	✓	51,083 (100%)	140,778 (100%)	9.0×10^{-5}	4.5	17
SO	✓	2,601,977 (99%)	63,497,050 (100%)	8.7×10^{-6}	3.9	11

authors is used. We observe that the number of authors per paper increases over time. This may cause varying performance in link prediction for different temporal snapshots. We deemed this outside the scope of the current research.

Digg The Digg network is a communication network and contains the reply network of the social news website Digg. Each node in the network is a person, and each edge connects the user replying to the receiver of the reply. Each reply is annotated with the time of that interaction.

Enron The Enron dataset is a communication network and contains over one million emails sent between employees of Enron between 1999 and 2003 [20]. For each email present in the dataset, sender and recipient are added as nodes and a directed edge from sender to recipient indicates the date of the email.

Slashdot Technology website Slashdot is a popular English website. The website allows commenting on each page, where users can start threaded discussion. The communication network is constructed from these threads where users are nodes and replies are edges, annotated with the time of the reply.

SO Like AU, the Stack Overflow (SO) network is collected from StackExchange and can be considered an online contact network. Nodes are users, and directed edges represent interactions, annotated with the time of the interaction.

5 Experiments

This section starts with the experimental setup used. We continue then in Sections 5.2 and 5.3 with the results and robustness checks.

5.1 Experimental setup

A few parameters need to be addressed to run the link prediction task. We highlight the following four in the next sections. First, the selection of node pairs using their distance in the network are considered. Second, the time intervals for the maturing and probe phase(s) need to be chosen for both the random and temporal split, as explained in Section 3.2. Third, the number of pairs of nodes used for training and testing are discussed. Fourth, the value of two hyper-parameters of the classifier are determined. Lastly, we explain how multiple snapshots from a network are constructed for robustness checks.

Distance selection The task of link prediction is computationally intensive for larger networks, since $\left|\left(V_{[t_0,t_b]} \times V_{[t_0,t_b]}\right) \setminus E_{[t_0,t_b]}\right|$ instances needs to be considered in the classifier. One way to reduce computational complexity, and reduce class imbalance as well, is to only consider pairs of nodes at a certain distance in the network [18]. We consider only pairs of nodes at a two-hop distance.

Time intervals The time intervals used for the maturing and probing phase in both the random and temporal split, can potentially have an effect on the obtained results. Hence, these values needs to be consistent for various networks.

Since a similar set-up is used as [12], timestamps of t_b, t_c and t_d were set in such a way that the proportion of edges in the maturing and probing phase remains roughly similar for the condmat network in [12]. This ratio $\left|E_{[t_0,t_b]}\right| : \left|E_{[t_b,t_c]}\right|$ is approximately equal to 5 : 1. To allow fair comparisons between the random and temporal split, the probing phase of the test interval should contain a similar number of edges as the probing phase of the training interval, i.e., $\left|E_{[t_0,t_b]}\right| \approx \left|E_{[t_c,t_d]}\right|$.

For computational reasons, we choose t_b, t_c and t_d for all networks such that the number of edges in each interval remains similar to the Condmat network. This means that $\left|E_{[t_0,t_b]}\right| \approx 50000$ and $\left|E_{[t_b,t_c]}\right| \approx \left|E_{[t_c,t_d]}\right| \approx 10000$.

Training and testing In case of random splitting,the instances $X_{[t_0,t_b]}$ should be split into two disjoint sets, as explained in Section 3.2. 75% of the instances are used for training and the remainder for testing, i.e. $\left|X^{\text{train}}_{[t_0,t_b]}\right| = 3\left|X^{\text{test}}_{[t_0,t_b]}\right|$.

Hyper-parameters Default parameters for the XGBoost were used, except for the following. The weights of the positive instances can be adjusted during training in such a way that the total weight of the positive and negatives samples are equal. In a 5-fold cross-validation setting applied on the training data, we determined for each network separately whether this improved performance on the train set. Furthermore, the maximum tree depth used for the base learners was also determined in the same 5-fold cross-validation.

Robustness checks We check whether our results are robust. Hence, the entire procedure is repeated ten times on Ask Ubuntu. We create ten non-overlapping snapshots by shifting intervals such that each next interval starts (t_a) at the end of the previous interval (t_c for random split, t_d for temporal split).

5.2 Results

The average precision score (AP) of the classifiers for the six networks with the random split and temporal split method is shown in Table 2. This metric shows large performance differences between the random and temporal split. The performance of the temporal split is for all networks lower than the random split. This may indicate that the random split provides an overly optimistic indication of the performance value. The difference between the random and temporal splits, varies widely between the networks, which may indicate that the extent to which the test set is reused, varies per network. Notably, the AP of the Ask Ubuntu network drops with 80%, demonstrating that the test set reuse could be severe.

5.3 Robustness checks

We check the robustness of the findings, by following the procedure outlined in Section 5.1. We find an AP of 0.025 ± 0.009 (mean$\pm$standard deviation) when

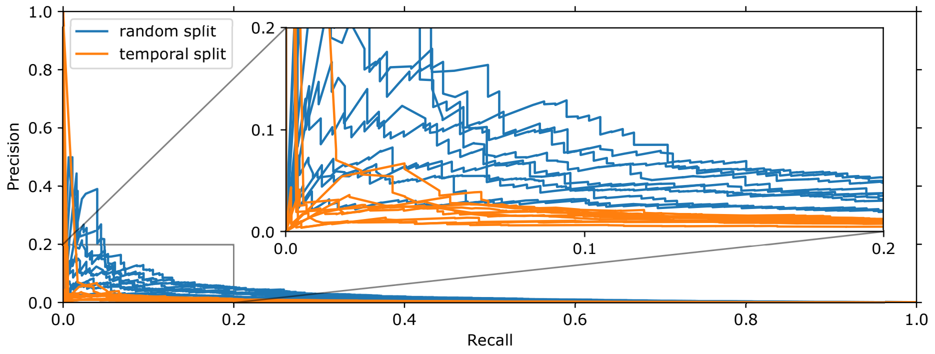

Fig. 3. Precision-recall curves for ten snapshots of the AskUbuntu network.

using the random split, while an AP of only 0.0061 ± 0.0016 is found for the temporal split. The different average precision curves are shown in Figure 5.3. The random split precision - recall curves clearly dominate their temporal counterparts at all snapshots.

6 Conclusion

The aim of the present research was to analyze different ways of obtaining the train and test set in link prediction. The results of this investigation on various large networks indicates that the random split consistently show better performance than the temporal split.

In future work, we plan to investigate new splitting strategies to separate train and test set. While the procedure of the temporal split prevents using the exact same temporal information of a given node, it still allows that the same node is both used in train and test set. More rigorous strategies should be devised to ensure to a further extent that the train and test set are truly disjoint and independent.

Table 2. Comparison of performance on the link prediction classification measured using average precision.

dataset	random split	temporal split
Askubuntu	0.023	0.0046
Condmat	0.012	0.0048
Digg	0.0043	0.0014
Enron	0.016	0.012
Slashdot	0.0076	0.0021
StackOverflow	0.0029	0.0013

References

1. W. L. Hamilton, R. Ying, and J. Leskovec, “Representation Learning on Graphs: Methods and Applications,” *arXiv preprint arXiv:1709.05584*, 2017.
2. D. Liben-Nowell and J. Kleinberg, “The link-prediction problem for social networks,” *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.
3. A. Kumar, S. S. Singh, K. Singh, and B. Biswas, “Link prediction techniques, applications, and performance: A survey,” *Physica A: Statistical Mechanics and its Applications*, vol. 553, p. 124289, 2020.
4. L. L. Linyuan and T. Zhou, “Link prediction in complex networks: A survey,” *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 6, 2011.
5. M. Al Hasan and M. J. Zaki, “A survey of link prediction in social networks,” in *Social Network Data Analytics*, pp. 243–275, Springer, 2011.
6. A. Ghasemian, H. Hosseinmardi, A. Galstyan, E. M. Airoldi, and A. Clauset, “Stacking models for nearly optimal link prediction in complex networks,” *Proceedings of the National Academy of Sciences*, p. 201914950, 2020.
7. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Science & Business Media, 2009.
8. A. Ghasemian, H. Hosseinmardi, and A. Clauset, “Evaluating Overfit and Underfit in Models of Network Community Structure,” *IEEE Transactions on Knowledge and Data Engineering*, 2019.
9. R. Roelofs, J. Miller, M. Hardt, S. Fridovich-keil, L. Schmidt, and B. Recht, “A Meta-Analysis of Overfitting in Machine Learning,” *NeurIPS*, p. 11, 2019.
10. J. P. Ioannidis, “Why most published research findings are false,” *Getting to Good: Research Integrity in the Biomedical Sciences*, vol. 2, no. 8, pp. 2–8, 2018.
11. C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth, “Preserving statistical validity in adaptive data analysis,” *Proceedings of the Annual ACM Symposium on Theory of Computing*, pp. 117–126, 2015.
12. R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla, “New perspectives and methods in link prediction,” in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 243–252, 2010.
13. M. A. Hasan, V. Chaoji, S. Salem, M. Zaki, and N. York, “Link Prediction using Supervised Learning,” in *SDM06: workshop on link analysis, counter-terrorism and security*, pp. 798–805, 2006.
14. E. C. Mutlu and T. A. Oghaz, “Review on graph feature learning and feature extraction techniques for link prediction,” *arXiv preprint arXiv:1901.03425*, 2019.
15. Z. Huang, X. Li, and H. Chen, “Link prediction approach to collaborative filtering,” *ACM/IEEE Joint Conference on Digital Libraries*, pp. 141–142, 2005.
16. Y. Yang, R. N. Lichtenwalter, and N. V. Chawla, “Evaluating link prediction methods,” *Knowledge and Information Systems*, vol. 45, no. 3, pp. 751–782, 2015.
17. T. Chen, T. He, M. Benesty, V. Khotilovich, and Y. Tang, “Xgboost: extreme gradient boosting,” *R package version 0.4-2*, pp. 1–4, 2015.
18. R. Lichtenwalter and N. V. Chawla, “Link prediction: Fair and effective evaluation,” *Proceedings of the 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2012*, pp. 376–383, 2012.
19. J. Kunegis, “Konect: the koblenz network collection,” in *Proceedings of the 22nd International Conference on World Wide Web*, pp. 1343–1350, 2013.
20. B. Klimt and Y. Yang, “The Enron corpus: A new dataset for email classification research,” in *European Conference on Machine Learning*, pp. 217–226, 2004.