# Network analysis methods for smart inspection in the transport domain

Bruin, G.J. de

# 3

# Performance of split strategies in link prediction

In Chapter 2, we have explored supervised machine learning towards the link prediction task. To train, validate (Definition 3) and test (Definition 4) supervised machine learning models, we need disjoint and independent splits of data. However, nodes in a real-world network are inherently related to each other. Therefore, separating candidate links into these disjoint sets is impossible. This challenge leads to Research question 2, which reads as follows.

> **Research question 2:** *How can we obtain accurate estimates of the performance of link prediction models by using adequate splits into the train, validation, and test set?*

In this chapter, we will evaluate two approaches to split data in link prediction: the *random* split and the *temporal* split. We will compare their performances on six large network datasets.

The current chapter corresponds to the following publication:
G. J. de Bruin, C. J. Veenman, H. J. van den Herik, and F. W. Takes. „Experimental evaluation of train and test split strategies in link prediction." In: *Proceedings of the 9th International Conference on Complex Networks and Their Applications*. Studies in Computational Intelligence 994. Springer, 2021, pages 79–91. DOI: 10.1007/978-3-030-65351-4_7

## 3.1 Machine learning methods on networks

Machine learning has emerged as a powerful instrument for analyzing all kinds of datasets. Here, we focus on supervised learning, which is well established when using non-relational (i.e., tabular) data. However, supervised machine learning on network data is challenging because obtaining an independent train, validation, and test set is nontrivial [74]. A common type of machine learning in networks is link prediction, where the goal is to predict whether a link will be formed in some future state of an evolving network (see Section 1.4 and Definition 10). In recent years, there has been an increasing interest in link prediction; hence, several review papers on this topic exist, e.g., [4, 101, 114].

A crucial first step in machine learning in networks is engineering the features. Here we assume that the network topology data can be converted into features with potentially helpful information for a predictive model. Established approaches for feature engineering in link prediction are based on (1) similarity, (2) probabilistic and maximum likelihood, and (3) dimensionality reduction [101]. Following our approach in Chapter 2, we will focus on the similarity-based approach. In this approach, pairs of nodes (candidates for links to be formed in the future) are assigned scores according to their similarity. We will exclusively use topological features to assess similarity, so we can apply the feature engineering 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 [63]. Second, the similarity-based approach provides easily explainable features compared to other techniques [114]. Third, most features are obtained at relatively low computational costs for the more extensive networks used in this study [114].

The similarity-based approach brings us to the main problem addressed in this chapter. For proper validation and testing in any machine learning task, instances belonging to the train set (on which the model will be trained) should be *disjoint* and *independent* of features belonging to the validation and test set. Since many dependencies usually exist between nodes in a network, it is a challenging task to achieve. We should seriously take into consideration that obtaining a *dependent* validation and test set possibly results in too-optimistic performance measurements (or, equivalently, overestimating the so-called generalization performance of the model [76]). According to Ghasemian *et al.*, it still needs to be determined how common machine learning steps, such as cross-validation and model selection methods, extend from non-relational to network data [62].

Assessment of the performance in supervised machine learning is essential for at least two reasons. The first reason is the selection of an appropriate model. It is possible to construct completely different models for a particular task, ranging from entirely different classifiers to identical models with other (hyper)parameters. Of course, we prefer to select a model with the best generalization performance on a dataset independent of the

train set. Here we aim at an independent validation set so that we can assess the extent to which overfitting takes place (Definition 11). The second reason for assessing model performance is to estimate the prediction error on new, unseen data. The performance should be assessed (1) by using the test data that is not used in any part of training the model, or (2) in choosing the right hyper-parameters or selecting a model [76]. Our research evaluates the differences in collecting independent datasets to examine a classifier's generalizability score. In our procedure, we perform a split only once, so two datasets are obtained (for instance, train and test data). Here we remark that our research methodology can easily be extended to obtain an independent third set (for instance, for validation purposes).

This work contributes to making an *in-depth* comparison of two approaches to splitting network data into two disjoint sets. Here, we aim to contribute to a better evaluation of performance estimation in link prediction and will answer Research question 2.

The remainder of this chapter is structured as follows. Related work is discussed in Section 3.2. Our research methodology (a formal description of the link prediction problem) is presented in Section 3.3. Relevant properties of the six temporal networks are presented in Section 3.4. Section 3.5 features information about the experimental setup. Then, Section 3.6 is concerned with the precise description of the experimental setup, the results, and a discussion of these results. Conclusions and future work are provided in Section 3.7.

## 3.2 Related work on validation of link prediction models

Only a relatively small body of literature is directly concerned with splitting a network dataset into disjoint and independent sets to evaluate the performance for machine learning purposes. We start our exploration with literature on performance estimation *in general* before we focus on prediction tasks in networks.

One of the causes of too-optimistic performance estimation is what is often described as "test set re-use" [167]. A well-known example is the $p$-hacking problem [91]. In short, $p$-hacking is the application of many different models to the same data in search of a statistically significant result with a sufficiently high $p$-value. This misuse can increase the probability that applied research findings are false. More specifically to data-driven research, too-optimistic performance estimations are suspected in Kaggle competitions [167]. In these online competitions, participants get the same dataset and compete for the best classifier performance on some predictive tasks without access to the test data. However, Kaggle allows users to repeatedly probe test data to obtain a continuously better performance of a submitted model. It is argued that this would lead to too-optimistic results [53]. However, the optimistic results were experimentally only observed to a limited extent [167].

Returning to machine learning on networks, Ghasemian *et al.* [62] investigated under- and overfitting networks. They examined (1) the performance of missing link prediction and (2) the so-called link description task as a diagnostic to evaluate the general tendency of such algorithms to under- and overfit. Hence, it is remarked that the authors defined the link prediction task differently since they do not necessarily include temporal information about the edges (see also Section 1.4). Hence, they removed a fraction of edges from a network and employed a machine learner to find the removed links from all pairs of nodes that are not connected anymore. The link description problem is different, as explained below. A network is sampled, but now the machine learner's task is finding the remaining edges of the sampled network from all pairs of nodes. The previously mentioned authors explain that (1) no algorithm can excel at both the link prediction and link description task and (2) that these tasks force an algorithmic trade-off, like the bias-variance trade-off in non-relational data [76]. In our work, we want to bring the notion of overfitting from non-relational data to relational data. While Ghasemian *et al.* focus on overfitting caused by the bias-variance trade-off [62], we investigate the too-optimistic estimation of generalization performance caused by test set reuse in networks.

## 3.3 Chapter research methodology

This section will start with a formal description of the link prediction problem in Subsection 3.3.1. In Subsection 3.3.2, we explain how we split the data into disjoint and separate sets for the link prediction classifier. Subsection 3.3.3 continues with the description of two types of features. In Subsection 3.3.4 we provide information about the classifier. Finally, in Subsection 3.3.5, we explain the performance metrics used.

### 3.3.1 Link prediction

The link prediction task is similarly defined as in Chapter 2. The temporal, potentially undirected, network (see Definition 9) $G = (V, E)$ consists of a set of nodes $V$ and edges $(u, v, t_i) \in E$ connecting nodes $u, v \in V$ with time $t_i \geq t_a$. Time $t'$ 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. Now assume that 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 $t_a < t_b < t_c$. The evolution of a temporal network is shown in Figure 3.1a.

The task for the supervised binary link prediction classifier (explained in Subsection 3.3.4) 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 3.1b). The network $G_{[t_a,t_b]}$ needs to be sufficiently "mature" so that the underlying static topology is well captured [112]. Hence we call the period of time $[t_a, t_b]$ the *maturing interval*. Subsequently, we call the time period $[t_b, t_c]$ the *probing interval*. For every pair of nodes $x_i \in X_{[t_a,t_b]}$, we probe whether the couple is present in the probing interval (indicated by $y_i = 1$) or not (denoted $y_i = 0$). The entire procedure is summarized in Figure 3.1. In Figure 3.1b, the instances considered in the classifier are shown. Positive instances ($y_i = 1$) are shown in solid green lines, while negatives ($y_i = 0$) are shown in red dashed lines.

## 3.3.2 Splitting strategies

After describing the general procedure of link prediction, we need a strategy to separate the pairs of nodes into different disjoint and independent sets for the classifier. Below, we will explain two dominant methods to split the data [3, 112]. Applying a temporal split is more complicated than the random split due to the various parameters involved. However, a temporal split prevents, to a greater extent, the reuse of the node and edge set information from the test set in training.

### 3.3.2A Random split

In the random split, the train and test sets are obtained by randomly splitting instances from a single probing phase. The validation set was omitted in our research (see Section 3.1). The random split method is, e.g., used in [112]. The entire procedure consists of three steps (see also Figure 3.2a).

1. We obtain all pairs of nodes disconnected during the maturing phase, $X_{[t_a,t_b]}$.
2. We determine for each of these pairs of nodes whether they connect (the value of $y_i$) in the probing phase $E_{[t_b,t_c]}$, as shown in Equation 3.1.

$$y_i = \begin{cases} 1 & \text{if } x_i \in E_{[t_b,t_c]} \\ 0 & \text{if } x_i \notin E_{[t_b,t_c]} \end{cases} \text{ for } x_i \in X_{[t_a,t_b]} \tag{3.1}$$

3. The pairs of nodes $X_{[t_a,t_b]}$ are separated into two disjoint sets $X^{\text{train}}_{[t_a,t_b]}$ and $X^{\text{test}}_{[t_a,t_b]}$ such that $X^{\text{train}}_{[t_a,t_b]} \cup X^{\text{test}}_{[t_a,t_b]} = X_{[t_a,t_b]}$ and $X^{\text{train}}_{[t_a,t_b]} \cap X^{\text{test}}_{[t_a,t_b]} = \emptyset$.

### 3.3.2B Temporal split

In the temporal split two consecutive probing phases obtain a train and a test set from two different time intervals. The temporal split method is for example used in [3]. As the name states, it takes the temporal aspect into account. More specifically, in the temporal split method two disjoint datasets are obtained by applying the probing phase on two

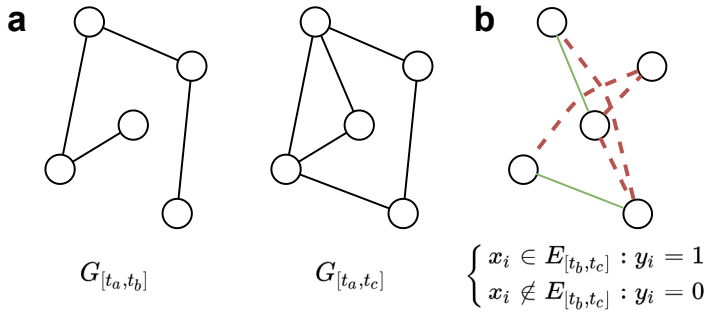
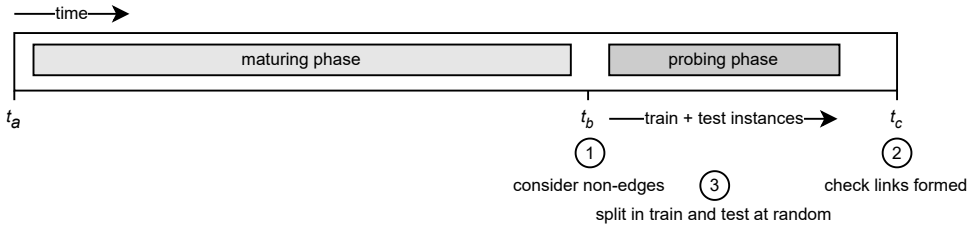
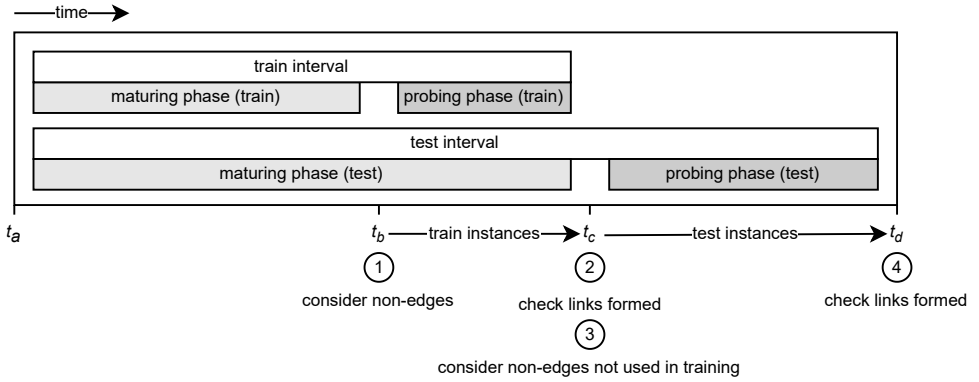



Figure 3.1: Procedure to obtain instances for the binary link prediction.




(a) Random split.




(b) Temporal split.

Figure 3.2: Two different strategies exist to obtain disjoint and independent sets.

consecutive snapshots called the training interval $[t_b, t_c]$ and test interval $[t_c, t_d]$. The four steps of this process are shown schematically in Figure 3.2b.

The train set is constructed in the first two steps as follows.

1. We consider every node pair that is not connected in the maturing phase of the train interval $X_{[t_a,t_b]}$.
2. For each node pair, we determine whether it will connect in the probing phase of the train interval, like Equation 3.1.

The test set is constructed similarly to the train set in Step 3 and 4.

3. We consider every node pair that is not connected in the maturing phase of the test interval $X_{[t_a,t_c]}$ and not used in the probing phase of the train interval.
4. We determine whether each pair of nodes connects in the probing phase of the test interval, as shown in Equation 3.2.

$$y_i = \begin{cases} 1 & \text{if } x_i \in E_{[t_c,t_d]} \\ 0 & \text{if } x_i \notin E_{[t_c,t_d]} \end{cases} \text{ for } x_i \in X_{[t_a,t_c]} \text{ and with } t_c < t_d \qquad 3.2$$

### 3.3.3 Features

As input for a classifier, we need a feature representation for every pair of nodes $x_i \in X$. As discussed in Section 3.1, 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 and not on contextual information [112, 125]. Hence, the features do not need any node information. Nodes with similar scores and thus a high similarity are more likely to connect. The score is either neighbor-based (similarity in local properties of the two nodes) or path-based (quasi-local or based on global properties of the two nodes) [40, 101]. We use the so-called High-Performance Link Prediction (HPLP) feature set defined in [112], as these are known to obtain good performance while limiting the number of features. The features can be separated into two types of features; the neighbor-based (Subsection 3.3.3A) and path-based (Subsection 3.3.3B) features. The features differ from those used in Chapter 2, where we used features that could be temporally extended to take past interactions into account (see Section 2.1).

In directed networks, we differentiate between (1) the neighbors connecting to node $u$, indicated by $N_{\text{in}}(u)$, and (2) the neighbors to which node $u$ connects, $N_{\text{out}}(u)$. Likewise, we differentiate also between the in-degree and out-degree of node $u$, $|E_{\text{in}}(u)|$ and $|E_{\text{out}}(u)|$, respectively.

#### 3.3.3A Neighbor-based features

Neighbor-based features take only the direct neighbors of the two nodes under consideration into account. Below we provide definitions of three concepts useful in subsequent feature definitions.

- **Number of Neighbors (NN)** is determined differently for directed and undirected networks. For *directed* networks, we use (1) the number of neighbors connecting to nodes $u$ and $v$ and (2) the number of nodes connected by $u$ and $v$. Hence, we get four features: $S_{\text{NN-in-}u}(u,v) = |N_{\text{in}}(u)|$; $S_{\text{NN-in-}v}(u,v) = |N_{\text{in}}(v)|$; $S_{\text{NN-out-}u}(u,v) = |N_{\text{out}}(u)|$; and $S_{\text{NN-out-}v}(u,v) = |N_{\text{out}}(v)|$. For 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., $S_{\text{NN-min}}(u,v) = \min(|N(u)|, |N(v)|)$ and $S_{\text{NN-max}}(u,v) = \max(|N(u)|, |N(v)|)$.
- **Degree (D)** is defined similarly, except that the *number of edges* is considered. For *directed* networks, we obtain again four features, viz. $S_{\text{D-in-}u}(u,v) = |E_{\text{in}}(u)|$; $S_{\text{D-in-}v}(u,v) = |E_{\text{in}}(v)|$; $S_{\text{D-out-}u}(u,v) = |E_{\text{out}}(u)|$; and $S_{\text{D-out-}v}(u,v) = |E_{\text{out}}(v)|$. For *undirected* networks, we obtain the maximum and minimum degree of nodes $u$ and $v$; $S_{\text{D-min}}(u,v) = \min(|E(u)|, |E(v)|)$; and $S_{\text{D-max}}(u,v) = \max(|E(u)|, |E(v)|)$.
- The **Common Neighbors (CN)** for a given pair of nodes is calculated by $S_{\text{CN}}(u,v) = |N(u) \cap N(v)|$. For *directed* networks, the score is calculated by considering the nodes that are connected from nodes $u$ and $v$, i.e., $S_{\text{CN}}(u,v) = |N_{\text{out}}(u) \cap N_{\text{out}}(v)|$.

#### 3.3.3B Path-based features

Path-based features take into account the paths between the two nodes under consideration. Since many paths exist, these features are computationally more expensive than neighbor-based ones. Below we provide the features with their definitions.

- **Shortest Paths (SP)**, $S_{\text{SP}}(u,v)$, indicates the number of shortest paths that run between nodes $u$ and $v$.
- **PropFlow (PF)**, $S_{\text{PF}}(u,v)$, corresponds to the probability that a restricted random walk starting from node $u$ and ends at node $v$ within $\ell$ steps [112]. We use the commonly applied value of $\ell = 5$ [112]. We collapse the network with multiple edges (occurring at different timestamps) to a weighted network where the weight equals the number of parallel edges between two nodes. Higher weights result in a higher transition probability for the random walk. This method is known for potentially obtaining scores for pairs of nodes $(u,v)$ that are different from those obtained for the pair $(v,u)$. This observation even holds for pairs in the undirected case [209]. Hence, we use the mean of the scores obtained for the pairs of nodes $(u,v)$ and $(v,u)$ in the undirected case.

### 3.3.4 Tree-based gradient boost classifier

We used a tree-based gradient boost learner for our classifier, as these are known to perform well in classification tasks [76]. The Python implementation of XGBoost was used [41]. This classifier has various hyper-parameters. While extensive hyper-parameter

tuning is beyond the scope of this chapter, we cross-validate two important hyper-parameters, viz. maximum depth of tree and class weights.

### 3.3.5 Performance metric: Average Precision

Link prediction is associated with *extreme class imbalance*, lower bounded by the number of nodes in the network [112]. Ideally, performance metrics used to evaluate the classifier should be robust against this class imbalance. The commonly encountered AUC lacks this robustness [111, 209] and is therefore not used. We are particularly interested in correctly predicting positives without losing precision, i.e., keeping the number of false positives low, and without losing recall, i.e., making sure we find all true positives. The Average Precision (AP) metric equals the weighted mean of precisions achieved at each threshold in the precision-recall curve. It is well-suited for our case.

## 3.4 Properties of the six temporal networks

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 that are (1) spanning a broad range of different domains, (2) publicly available, and (3) sufficiently large. The properties of these networks are shown in Table 3.1. We mention for each network whether it is directed, the number of nodes, the number of edges, the density, the mean distance ($\bar{d}$), and the diameter (Ø). The density, mean distance, and diameter 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 for the Condmat network, all datasets were obtained from KONECT [104].

1. The *Ask Ubuntu* network is an online contact network [143]. The snapshot of the network that we used was obtained in 2017. Ask Ubuntu is a community-driven question-and-answer site dedicated towards Ubuntu; it is derived from StackExchange,

Table 3.1: Summary statistics of the six temporal networks. (Edges and nodes in the GC are indicated between brackets. The mean distance between nodes is given in column $\bar{d}$, and column Ø indicates the diameter of the networks.)

| # | dataset | directed | nodes | (GC) | edges | (GC) | density | $\bar{d}$ | Ø |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Ask Ubuntu | ✓ | 159,316 | (96%) | 964,437 | (100%) | $4.0 \times 10^{-5}$ | 3.9 | 13 |
| 2 | Condmat | ✗ | 17,218 | (88%) | 88,090 | (100%) | $3.7 \times 10^{-4}$ | 6.3 | 19 |
| 3 | Digg | ✓ | 30,398 | (98%) | 87,627 | (100%) | $1.9 \times 10^{-4}$ | 4.7 | 12 |
| 4 | Enron | ✓ | 87,273 | (97%) | 1,149,072 | (100%) | $7.9 \times 10^{-5}$ | 4.9 | 14 |
| 5 | Slashdot | ✓ | 51,083 | (100%) | 140,778 | (100%) | $9.0 \times 10^{-5}$ | 4.5 | 17 |
| 6 | Stack Overflow | ✓ | 2,601,977 | (99%) | 63,497,050 | (100%) | $8.7 \times 10^{-6}$ | 3.9 | 11 |

a network of question-and-answer websites on topics in diverse fields. The nodes are the users, and a direct edge is created when a user replies to another user's message. These interactions can consist of an answer to another user's question, comments on another user's answer, and comments on another user's comments. Each edge is annotated with the time of interaction.

2. The scientific co-authorship dataset *Condmat* entails condensed matter physics collaborations from 1995 to 2000[1]. The undirected temporal network is made by adding an edge between all authors of a publication [111]. For each edge, the date of the publication connecting these authors is used. We observe that the number of authors per paper increases over time. It may cause varying performance in link prediction for different temporal snapshots. We deemed this outside the current research scope.
3. The *Digg* network is a communication network and contains the reply network of the social news website Digg from November 2009 [79]. Each node in the network is a person, and each edge connects the user replying to the reply receiver. Each reply is annotated with the time of that interaction.
4. The *Enron* dataset is a communication network and contains over a million emails sent between employees of Enron between 1999 and 2003 [97]. A directed edge from the sender to the recipient is added for each email.
5. The *Slashdot* website is a English tech news website that allows users to place a comment on each page, and shows where users can start a threaded discussion [65, 159]. The period during which the data was crawled covered August 2005 to August 2006 [85]. The communication network is constructed from these threads where users are nodes, and replies are edges, annotated with the answer time.
6. Like Ask Ubuntu, the *Stack Overflow* network is collected from StackExchange and can be considered an online contact network [143]. Nodes are users; directed edges represent interactions annotated with the exchange time.

## 3.5 Experimental setup

In Section 3.3 we explained the research methodology of our experiment. However, a few parameters need to be addressed explicitly to run the link prediction task (cf. Subsection 3.3.1) used in the experiment. We discuss them in the sections below. First, we discuss the *selection of node pairs* using their *distance* in Subsection 3.5.1. Second, in Subsection 3.5.2, we discuss the *choice of the time intervals* for the maturing and probe phases for both the *random* and *temporal* split (Subsection 3.3.2). Third, we continue in Subsection 3.5.3 by discussing the *number of pairs of nodes* used for *training* and *testing*. Fourth, the *performance is improved* by optimizing *the class weight* and *the value of the maximum tree depth* in Subsection 3.5.4. The class weight and the maximum tree depth

[1]The data was obtained from https://github.com/rlichtenwalter/LPmade

are called hyper-parameters. Fifth, we explain in Subsection 3.5.5 how *multiple snapshots* from a network are *constructed* for robustness checks.

### 3.5.1 Distance selection

The task of link prediction is computationally intensive for larger networks because there are $\left|\left(V_{[t_a,t_b]} \times V_{[t_a,t_b]}\right) \setminus E_{[t_a,t_b]}\right|$ instances. A way to reduce the number of instances and to reduce class imbalance, is to only consider pairs of nodes at a limited distance of each other in the network [111]. For our distance selection, we consider only pairs of nodes at a distance of two in our network.

### 3.5.2 Time intervals

The choice of the time intervals used for the maturing and probing phase in both the random and temporal split can affect the obtained results. To allow *fair* comparisons between the random and temporal split, the probing phase of the test interval should contain a number of edges similar to the probing phase of the training interval, i.e., $\left|E_{[t_a,t_b]}\right| \approx \left|E_{[t_c,t_d]}\right|$. Moreover, we need values that are consistent for the various networks. Timestamps of $t_b$, $t_c$, and $t_d$ were set so that the proportion of edges in the maturing and probing phase are approximately similar to the settings in [112].

For the Condmat network, this results in a ratio $\left|E_{[t_a,t_b]}\right| : \left|E_{[t_b,t_c]}\right|$ approximately equal to $5:1$. The number of edges are then $\left|E_{[t_a,t_b]}\right| \approx 50000$ and $\left|E_{[t_b,t_c]}\right| \approx \left|E_{[t_c,t_d]}\right| \approx 10000$.

### 3.5.3 Training and testing

In the case of random splitting, the instances $X_{[t_a,t_b]}$ should be split into two disjoint and independent sets, as explained in Subsection 3.3.2A. A 75% of the instances are used for training and the remainder for testing, i.e., $\left|X^{\text{train}}_{[t_a,t_b]}\right| = 3\left|X^{\text{test}}_{[t_a,t_b]}\right|$.

### 3.5.4 Improved performance

Below, we report the choices made regarding two hyper-parameters used in the XGBoost algorithm to improve performance. First, we adjusted the *class weights* of the positive instances to equal the total weight of the positive and negative samples. In a five-fold cross-validation setting applied to the training data, we determined for each network separately whether the adjusted class weights improve performance on the train set. Second, we determined the optimal *maximum tree depth* using the same five-fold cross-validation.

### 3.5.5 Robustness checks

The experimental setup of the *robustness checks* is as follows. We repeat the full procedure of selecting time intervals (see Subsection 3.5.2) on the Ask Ubuntu network ten times. Ten non-overlapping snapshots are obtained 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). The robustness is then checked by comparing the Average Precision performance of the random split with that of the temporal split.

## 3.6 Results of the two different splitting strategies

The Average Precision (AP) (see Subsection 3.3.5) score of the classifiers for the six networks with the random split and temporal split method is shown in Table 3.2. This metric shows significant performance differences between the random and temporal split. The performance of the temporal split is for all networks lower than the random split. It may indicate that the random split provides an overly optimistic indication of the performance value. Furthermore, the difference between the random and temporal splits varies widely between the networks, indicating that the extent to which the test set is reused varies by network. Notably, the AP of the Ask Ubuntu network drops by 80%, demonstrating that the test set reuse can be extensive.

### Robustness checks

We checked the robustness of the findings by following the procedure (ten times performing the full procedure, as outlined in Subsection 3.5.5). We find an AP of $0.025 \pm 0.009$ (mean$\pm$standard deviation) when using the random split, while an AP of only $0.0061 \pm 0.0016$ is found for the temporal split. The different AP curves are shown in Figure 3.3. From our results, we may conclude that the random split precision-recall curves dominate their temporal counterparts in all snapshots.

Table 3.2: Link prediction performances for different split strategies (applied on the six temporal networks using the AP metric).

| # | dataset | random split | temporal split |
|---|---|---|---|
| 1 | AskUbuntu | 0.023 | 0.0046 |
| 2 | Condmat | 0.012 | 0.0048 |
| 3 | Digg | 0.0043 | 0.0014 |
| 4 | Enron | 0.016 | 0.012 |
| 5 | Slashdot | 0.0076 | 0.0021 |
| 6 | Stack Overflow | 0.0029 | 0.0013 |

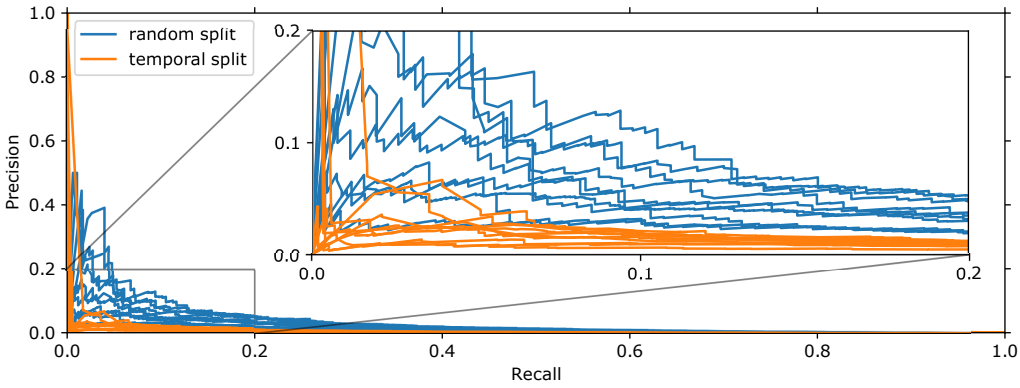



Figure 3.3: Precision-recall curves of the AskUbuntu network for robustness checks. (Performed on ten different snapshots.)

## 3.7 Chapter conclusion and outlook

In the present chapter, we analyzed two different ways of splitting data viz. into disjoint and independent sets in network data for training, validation, and testing of link prediction models. The results indicate that the *random split* consistently obtains higher performance estimates than the *temporal split*.

So, we are able to answer Research question 2: "How can we obtain accurate estimates of the performance of link prediction models by using adequate splits into train, validation, and test sets?". The answer is: "We obtain accurate estimates of the link prediction performance by using the temporal split, because the alternative, the random split, shows signs of overfitting." Based on our experiments we may conclude that the temporal split method provides more accurate estimates of the link prediction model performance.

### Chapter outlook

While the procedure of the temporal split prevents using the same temporal information of a given node, it still allows the same node to be used in multiple sets. Future work should devise more rigorous strategies to ensure to a further extent that the train, validation, and test set are disjoint and independent. Further research should be conducted to establish the relation between the extent of overfitting and the (domain of the) network.