Karlsruhe Institute of Technology

Link Prediction in Large-scale Complex Networks

Bachelor's Thesis of

Kolja Esders

at the Department of Informatics
Institute of Theoretical Informatics

Reviewer:	Juniorprof. Dr. Henning Meyerhenke
Second reviewer:	Prof. Dr. Dorothea Wagner
Advisor:	Elisabetta Bergamini
Second advisor:	Christian Staudt

09. February 2015 – 11. May 2015

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 11. May 2015

. . . . . .

(Kolja Esders)

Acknowledgements

I would like to express my gratitude to Juniorprof. Dr. Henning Meyerhenke for giving me the opportunity to pursue my thesis at the research group Parallel Computing in the Institute of Theoretical Informatics.

I am very grateful for the kind advice from my advisors Elisabetta Bergamini and Christian Staudt. Elisabetta provided me with enough freedom and shared valuable comments that were of great help. Christian assisted me with much needed guidance and was always open to questions. I would also like to thank all members of the research group for their assistance.

I am indebted to Ryan Lichtenwalter from the University of Notre Dame for extremely helpful and extensive conversations about all facets of link prediction. A special thanks goes out to Nicola Barbieri from Yahoo Research and Giuseppe Manco from the National Research Council in Italy for their kind and insightful explanations.

I would also like to thank Thomas Fitz for the considerable number of comments on a first draft and Florian König for his highly motivational words.

Finally, I would like to thank my family and friends for their encouragement and support.

Abstract

Link prediction methods try to predict the likelihood of a future connection between two nodes in a given network. This can be used in biological networks to infer protein-protein interactions or to suggest possible friends to a user in an online social network. Due to the enormous amounts of data that is collected today, there is a need for scalable approaches to this problem. In this thesis, we make use of machine learning and evaluate nine unsupervised as well as three supervised methods on six networks with the Receiver Operating Characteristic (ROC) and Precision-Recall (PR) metrics. To investigate the implications of different experimental setups, the testing and training set are varied in size and density. We also introduce a new local index called *Neighborhood Distance*. The experiments show that supervised methods consistently outperform unsupervised methods by up to 7.2% with respect to the ROC metric. The overall prediction quality is also highly dependent on the properties and structure of the analyzed network. In an effort to parallelize predictors, we achieve a speedup of more than ten. The findings suggest that generating custom predictors for networks based on their properties deserves further research.

Zusammenfassung

Link Prediction Methoden versuchen die Wahrscheinlichkeit einer zukünftigen Verknüpfung zwischen zwei Knoten in einem Netzwerk vorherzusagen. Dies kann beispielsweise in biologischen Netzwerken dazu verwendet werden, um Protein-Protein Interaktionen abzuleiten oder um einem Nutzer in einem sozialen Netzwerk potentielle Freunde vorzuschlagen. Aufgrund der enormen Menge an Daten die heutzutagen gesammelt werden besteht außerdem ein Bedarf nach skalierbaren Ansätzen für dieses Problem. In dieser Arbeit werden mithilfe von Maschinellem Lernen neun unüberwachte und drei überwachte Verfahren auf sechs Netzwerken mit den Receiver Operating Characteristic (ROC) und Precision-Recall (PR) Metriken evaluiert. Um die Implikationen von verschiedenen Parametern während der Experimente zu untersuchen, werden Trainings- und Testset in ihrer Größe und Dichte variiert. Es wird zusätzlich ein neuer lokaler Ähnlichkeitsindex mit dem Namen *Neighborhood Distance* vorgestellt. Die Experimente zeigen, dass überwachte Verfahren die unüberwachten Verfahren durchweg mit bis zu 7.2% bezüglich der ROC Metrik übertreffen. Die Qualität der Vorhersagen ist ebenfalls stark abhängig von den Eigenschaften und Strukturen des analysierten Netzwerks. In einem Bestreben die Verfahren zu parallelisieren wird ein Speedup von mehr als zehn erzielt. Die Ergebnisse legen nahe, dass das Erzeugen von netzwerk-spezifischen Verfahren basierend auf den jeweiligen Netzwerk-Eigenschaften weiter erforscht werden sollte.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Background

Many biological, social, and technological phenomena can be described by networks. In such networks, nodes represent domain-dependent entities and links represent relations or interactions between entities. Social interactions, for example, can be represented through a network where a node is a person and a link indicates a friendship between two persons. Link prediction is the problem of estimating the likelihood that a link exists between two currently unconnected nodes, based on the observed nodes and links.

Typical applications that require the prediction of *future* links are social networks like Facebook[1] or Google+[2], where link prediction can be used to suggest friends to a user. This is proving to be very successful for Facebook, where a "significant fraction" [4] of links are created through users that add the suggested people as actual friends on Facebook.

Link prediction can also be used to predict *missing* links in a network. In static biological networks like protein-protein interactions and metabolic networks, the existence of a link between two nodes can only be determined through field or laboratory experiments. As there are less than 0.3% of interactions between human proteins identified [3], reducing the search space through link prediction to protein-pairs where an interaction seems likely (based on the known interactions) could help to significantly reduce experimental costs. In security, link prediction can also be used to analyze terrorist networks with the objective to infer that individuals are working together even though their interaction has not been observed in the current information base [34].

In recent years, the amount of data collected through web services has massively increased because there are more Internet users and users utilize more services. Also, the companies that provide web services store more data in order to conduct data-driven decision-making. The research group SINTEF found in 2013 that 90% of the world's data was generated in the past two years [15]. This implies the need for scalable approaches to link prediction in order to be able to process this magnitude of data.

[1] `http://www.facebook.com`
[2] `http://plus.google.com`

The main issue in link prediction is extreme class skewness. Even though a user on Facebook could connect to 1.35 billion people [17], he will on average connect to only about 100 nodes in the social graph [4]. This leads to a strong class imbalance between existing and absent links. Considering all the absent links for link prediction would be computationally infeasible. This is why there is a need to reduce the number of node-pairs to use for evaluation.

The long-term vision of link prediction could be to make search superfluous. Instead of a user searching for the next rock concert in his town, a *reliable* intelligent system could suggest the next and most relevant concerts to this user based on previously visited concerts, taste of music, and location data. Personal assistants like Google Now or Microsoft Cortana are first attempts at this vision. However, currently they are are only able to assist and enrich search, rather than completely replacing it.

1.2. Scope

The overall objective of this thesis is to investigate the scalability and performance of supervised as well as unsupervised link prediction methods on a number of networks from different domains.

In particular, we investigate twelve link predictors on six different networks and discuss their performances. Our predictors include both unsupervised and supervised methods, which combine multiple unsupervised methods into a single classifier. In particular, we consider nine unsupervised and three supervised classifiers. Apart from analyzing established link predictors, we introduce the *Neighborhood Distance* index and investigate its performance. To achieve acceptable runtimes for large networks, parallelization is investigated in the scope of the development of a link prediction module for NetworKit, an open-source toolkit for high-performance network analysis. The networks we study display phenomena in (co)authorship or citation of research papers, infrastructure, and social interactions. Half of the networks have time stamps to indicate the time of link-creation. Even though the other networks lack time stamps, this does not imply that the original phenomena that are represented through the networks would not lead to evolving networks. Mostly, timestamps are not available because they simply were not recorded during data collection.

This thesis will not consider link prediction for networks with additional node and/or link attributes. Also, the generated predictions will not be weighted. This means that the same importance will be given to all the existing links and timestamps will not be used, for example, to weigh more recently created links stronger than older links.

1.3. Outline

To introduce the terminology used throughout the link prediction community and this thesis, the next chapter will explain graph theory and present basic graph properties. Also, the concept of supervised learning will be illustrated. In chapter 3, we will discuss other approaches to link prediction. In chapter 4, a more formal problem definition of the link prediction problem will be introduced. Afterwards the unsupervised predictors will be described as well as the supervised classifiers and their specific setup. Lastly, the evaluation metrics used in the experiments will be presented. In the following Implementation chapter, the core functionality of the link prediction module will be shown with some remarks about implementation details and module structure. In chapter 6, after talking about the setup of the experiment as well as the used data sets, the results of the experiments are presented. These will be discussed in chapter 7. Finally, a conclusion about the findings will be drawn in chapter 8.

2. Basic Concepts

This chapter introduces definitions and terminology used throughout the thesis. These definitions enable the research community to precisely formulate ideas about link prediction and communicate them effectively. First off, the concepts of graph theory are presented and, based on this, graph-related properties are laid out. Afterwards the concept of machine learning along with basic definitions will be introduced.

2.1. Graph Theory

A *graph* or *network* is a mathematical structure that is used to model pairwise relationships between entities. The entities are called *vertices* or *nodes* and the relationships are called *edges* or *links*. These terms are used interchangeably throughout the thesis. An edge can be *directed* (asymmetric) or *undirected* (symmetric). In the latter case there is a distinction between the *start node*, from which the edge originates, and the *end node*, where the edge ends. We say that an edge is *incident* with each of the nodes it connects. The edges can be associated with *weights* that could represent domain-specific values like costs or distances, and are visualized as (arched) lines with an arrowhead at the end node in case the graph is directed. If there are multiple edges connecting the same two nodes, these edges are called *multi-edges*. A *self-loop* or *buckle* is an edge that connects a vertex to itself. If an undirected graph contains no self-loops or multi-edges, it is referred to as a *simple graph*. A node can be associated with a number of *attributes* like *time of creation*. Nodes are often visualized as circles that can be identified through a unique *identity*. If two nodes are connected through an edge, they are called *adjacent*. A *path* is a sequence of edges connecting a sequence of vertices. [5]

More formally, a graph G is defined through an ordered pair $G = (V, E)$, where $E \subseteq V \times V$. The elements of V are the vertices and the elements of E are the edges. Directed edges are denoted as $(u, v) \in E$ and undirected edges as $\{u, v\} \in E$. We demand that V is not empty and consider only graphs with a *finite* number of vertices. The set of edges is allowed to be empty. A graph can be represented through an *adjacency matrix*, which is a two-dimensional boolean matrix with dimensions $|V| \times |V|$. Under the assumption

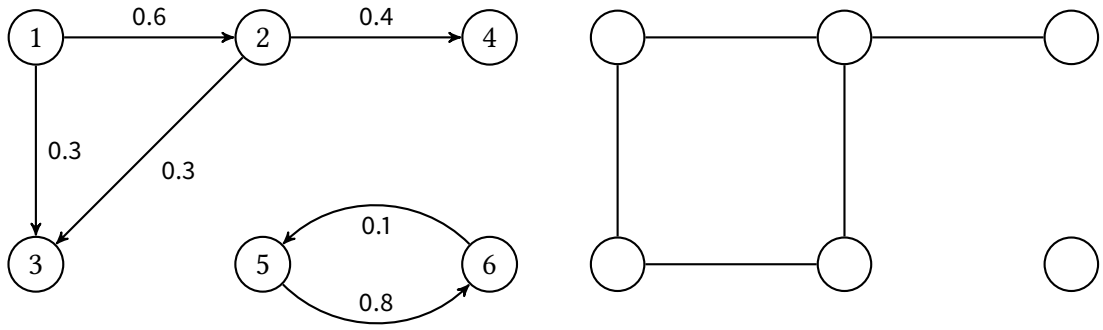

Figure 2.1.: Visualization of two exemplary graphs. On the left is a directed, weighted graph where the nodes possess unique identities. On the right is an undirected simple graph.

that every node can be uniquely identified by a number in $\{0, \ldots, |V|-1\}$, a single entry $a_{ij} \in A$ in the adjacency matrix for a directed graph is defined as

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

An adjacency matrix for an undirected graph is symmetric, as there also exists $\{j,i\} \in E$ for every link $\{i,j\} \in E$.

For the following definitions and metrics we assume that $G = (V, E)$ is an undirected graph.

Path

A path in an undirected graph G is a non-empty list $p = (v_0, \ldots, v_n) \in V^{(+)}$ where $\{v_i, v_{i+1}\} \in E$ holds true for $i \in \mathbb{G}_n$. $V^{(+)}$ denotes all non-empty lists whose elements are in V. The number $n = |p| - 1$ of edges is called the *length* of the path. The *shortest path* between two nodes in an unweighted graph is the path which minimizes the number of links between the nodes while still connecting them. If the graph is weighted, the shortest path is the path that minimizes $\sum_{i=0}^{n-1} w(v_i, v_{i+1})$, where $w(v_i, v_{i+1})$ denotes the weight of the edge $\{v_i, v_{i+1}\}$.

Subgraph

A graph $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V$, and $E' \subseteq E \cap V' \times V'$.

Connectivity

A graph is called *connected* if there is a path connecting every pair of nodes in the graph. If the graph is not connected, it is called *disconnected*.

Neighborhood

Node v is called a *neighbor* of u in G if $\{u, v\} \in E$, and vice versa. The *neighborhood* of u is defined as $\Gamma(u) := \{v \mid \{u, v\} \in E\}$.

Degree

The degree $k(u) := |\Gamma(u)|$ of a node u is simply the number of nodes in the neighborhood $\Gamma(u)$. A node is called *isolated* if its degree is 0.

N-degree neighborhood The *n-degree neighborhood* of a node u is defined as the set of nodes exactly n *hops* away, where the hops are simply pairwise different edges.

Community

Also called *cluster* or *module*, it refers to a group of nodes that "probably share common properties and/or play similar roles within the graph" [19]. The concrete definition of a community depends on the application and/or specific system that is present.

Clique

A *clique* is a subset $C \subseteq V$ of nodes where every node in C is connected to each other. This means $\forall u \in C. \forall v \in C \setminus u \colon \{u, v\} \in E$.

Diameter

The *diameter* $d(G)$ of a graph G is simply the longest shortest path between any two of its nodes. A disconnected graph has an infinite diameter.

Clustering coefficient

The *local clustering coefficient* of a node u quantifies how close its neighbors are to being completely connected to each other. More formally, this can be defined as:

$$C_u = \frac{3 \times \text{\#triangles adjacent to } u}{\text{\#possible triples adjacent to } u}. \tag{2.2}$$

2.2. Machine Learning

Machine learning is the field of scientific study that explores the construction and properties of algorithms that can be said to *learn* from their input data [31], where an algorithm "is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [46]. In the context of link prediction, this means a machine learning algorithm

will try to optimize the prediction-quality of links based on a given network and corresponding link- and node-properties. Machine learning tasks can be classified into three categories depending on the learning feedback: *Unsupervised learning*, *Supervised learning*, and *Reinforcement learning*. Within the scope of this thesis we will concentrate on the first two categories. The task of predicting labels/classes (existent link & absent link) is called *classification* and will be our goal in the experiments. An algorithm performing this task is consequently called a *classifier*. If the goal is to predict continuous values, the process is called *regression*. To evaluate the performance of a classifier, there has to be a *ground truth* to compare predictions from the classifier against. For this purpose the labeled initial dataset is divided into a *training set* and *testing set*. The classifier can make use of the training set to optimize its predictive qualities but it has no knowledge about the training set. This way the classifier can predict labels for the testing set and these can be compared against the ground truth labels.

In the following subsection some key components of machine learning algorithms are presented and afterwards category-dependent characteristics will be introduced.

2.2.1. Key Components

A machine learning framework typically consists of the following components [40]:

Input Space

The *input space* $\mathscr{X}$ consists of the objects under investigation. These are usually represented by *feature vectors* that are descriptive of the objects. *Feature-extraction* describes the process of building non-redundant, informative features from an object and varies across applications. During the thesis we will denote a feature-vector for a node-pair (u, v) as $f_{u,v} = [f_1(u, v), \ldots, f_n(u, v)]$.

Output Space

The *output space* $\mathscr{Y}$ of the task contains the range of possible output values y. For example in *regression* this might be the space of real numbers $\mathbb{R}$. In the case of link prediction, this is usually $\mathscr{Y} = \{0, 1\}$, where 0 indicates an absent link and 1 indicates an existing link between u and v.

Hypothesis Space

The *hypothesis space* defines the class of all functions $h\colon \mathscr{X} \mapsto \mathscr{Y}$ mapping the input space to the output space. In particular, this means the functions operate on the feature vectors and return predictions according to the format specified through the output space.

These components form the core of every machine learning framework. The following subsections introduce some problem-dependent extensions to this core.

2.2.2. Unsupervised Learning

Unsupervised learning is about trying to find a structure in unlabeled data. For this purpose, there are a number of approaches including, but not limited to, *clustering*, *singular value decomposition* (SVD), and *non-negative matrix factorization*. In this thesis, *similarity indices* will be used that base their prediction-quality on assumptions about the structure of the dataset. A simple example would be the assumption that two persons in a social network are more likely to be befriended in the future if they have a large number of common friends.

2.2.3. Supervised Learning

In the case of supervised learning, the input objects are associated with a ground truth label. This allows for feedback-mechanisms to enhance the prediction-quality of an algorithm. For this purpose, a *loss-function* is introduced. As depicted in Figure 2.2, this function "measures to what degree the prediction generated by the hypothesis is in accordance with the ground truth label" [40]. Typical loss-functions are the exponential loss, the squared loss, and the absolute loss.

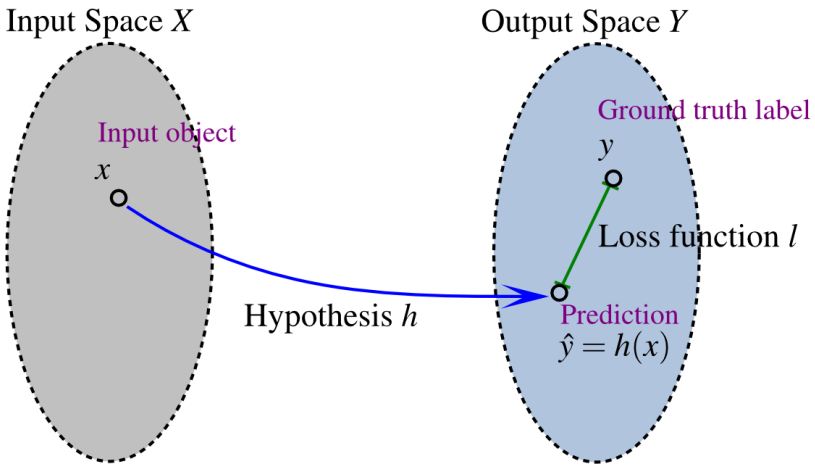

Figure 2.2.: Basic supervised learning framework [40].

3. Related Work

3.1. Matrix Factorization

The use of matrix factorization to model distances in large-scale networks [44] and its utilization in recommender systems [32] have recently inspired many efforts [45, 1, 70] to use matrix factorization for the link prediction problem in homogeneous networks. Matrix factorization treats link prediction as a matrix completion problem on a partially observed network $G \in \{0, 1, ?\}^{n \times n}$, where 0 denotes an absent link, 1 denotes a present link, and $?$ denotes a link with unknown status [45]. The method works by factorizing $G \approx L(U \Lambda U^T)$ with $U \in \mathbb{R}^{n \times k}$, $\Lambda \in \mathbb{R}^{k \times k}$, $k \ll n$ and link function $L(\cdot)$ to approximate G through supervised learning. This generally means to learn a parameter vector θ for the optimization problem

$$\min_{\theta} \sum_{(i,j) \in \mathcal{O}} \ell(G_{ij}, \hat{G}_{ij}(\theta)) + \Omega(\theta), \tag{3.1}$$

where $\mathcal{O} = \{(i, j) \mid G_{ij} \neq ?\}$ is the set of observed links, $\hat{G}_{ij}(\theta)$ is the models predicted similarity score for the node-pair (i, j), $\ell(\cdot, \cdot)$ is a loss function (mostly $\ell(x) = x^2$) and $\Omega(\cdot)$ is a *regularization term* that prevents overfitting. In link prediction, we can set

$$\hat{G}_{ij} = L(u_i^T \Lambda u_j + b_i + b_j), \tag{3.2}$$

where b_i and b_j are node-related biases used as adjustment variables and $L(\cdot)$ is a link function. The most common approach to minimizing equation 3.1 is the *stochastic gradient descent* (SGD) method which tries to reduce the prediction error. An advantage of matrix factorization is the fact that the runtime complexity of training is linear in the number of links through the usage of SGD in favor of the *Markov-Chain-Monte-Carlo* (MCMC) method. Additionally, the approach allows the exploitation of the graph topology which should make it superior to unsupervised methods. A major drawback is the fact that matrix factorization can not be used to only compute predictions for a subset of node-pairs.

3.2. Relational Markov Networks

Given an undirected graph G where there are additional attributes for nodes and links, a Relational Markov Network (RMN) can be used to define a single probabilistic model over the entire graph. This model will be built on the given attributes and can be used to infer new links in the network. This approach was first introduced by Taskar et al. [58] for the link prediction problem.

As an example, let us assume that we want to predict future hyperlinks between webpages. By regarding the webpages and hyperlinks as objects with attributes like the type of the webpage (news site, private homepage) or the anchor text of the hyperlink, a binary (*true* or *false*) *exists*-attribute for the hyperlink indicates whether a hyperlink actually links from one page to another. By considering all the possible hyperlinks between all websites, the task of an RMN can be reduced to inferring the value of the exists-attribute based on all the given attributes of the two webpage objects.

More formally, a RMN makes use of an undirected graph and a set of clique potential functions to represent the joint probability over the attributes of the nodes and links. Let V denote a set of discrete random variables, where v will be a concrete instantiation of the variables in V. If $C(G)$ denotes the set of cliques in G, the joint probability over v is given by

$$p(v) = \frac{1}{Z} \prod_{c \in C(G)} \phi_c(v_c), \tag{3.3}$$

where Z is the standard normalizing partition function, v_c is the set of nodes in clique c, and ϕ_c is a clique potential function defined on the joint domain of v_c. The RMN specifies the cliques and potentials between attributes of related objects at a template level (see [57] for details). Finally, the parameters of the RMN for a fixed set of cliques can be learned from data. In large networks with multiple attributes, exact inference is infeasible because of the amount of data that needs to be processed. For this reason, Taskar et al. propose belief propagation (see [65] for details) for inference.

3.3. Hierarchical Models

Recent studies suggest that many real world networks have an inherent hierarchical organization. In 2003, Barabási et al. [6] analyzed a number of real networks featuring the Internet Movie Database[1] (IMDb), metabolic networks and the World Wide Web. They

[1] `http://www.imdb.com`

found that for several large networks the *clustering coefficient* $C(k)$ is well approximated by $C(k) \sim k^{-1}$ which indicates an inherently hierarchical structure in these networks.

Based on this evidence Clauset et al. [13] proposed in 2008 a probabilistic model for the link prediction problem. The model makes use of dendrograms to represent the hierarchical structure of the network. Closely related node-pairs have a lowest common ancestor that is lower in the dendrogram than more unrelated pairs. Furthermore every inner node r has a corresponding probability p_r assigned to it. The probability of a connection between two nodes is simply the probability p_r that is associated with their lowest common ancestor r in the dendrogram. The authors detect the hierarchical structure of a network by fitting the hierarchical model to the observed data of the network by combining a maximum likelihood approach with a Monte Carlo sampling algorithm on all possible dendrograms. The likelihood of a model $(D, \{p_r\})$ is

$$\mathcal{L}(D, \{p_r\}) = \prod_{r \in D} p_r^{E_r} (1 - p_r)^{L_r R_r - E_r} \tag{3.4}$$

with E_r being the number of edges in the observed network whose endpoints have r as their lowest common ancestor in D. L_r and R_r are the number of leaves in the left and right subtrees (respectively) rooted at r. If the given dendrogram is fixed the probabilities $\{\overline{p}_r\}$ which maximize $\mathcal{L}(D, \{p_r\})$ are simply given by

$$\overline{p}_r = \frac{E_r}{L_r R_r}. \tag{3.5}$$

Figure 3.1 shows an example network with six nodes and the likelihood of two corresponding dendrograms. The internal nodes of the dendrograms are labeled with the maximum-likelihood probability $\overline{p}_r$. The second dendrogram is far more likely because it correctly divides the network into two highly connected subgraphs at the root. We can verify this by using equation 3.4: $\mathcal{L}(D_1, \overline{p}_{r,1}) = (1/3)(2/3)^2 \cdot (1/4)^2(3/4)^6 \approx 0.00165 < 0.04330 \approx (1/9)(8/9)^8 = \mathcal{L}(D_2, \overline{p}_{r,2})$.

To obtain a similarity score for a given node-pair (i, j) one can calculate the mean probability $\langle p_{ij} \rangle$ by averaging over the related probabilities p_{ij} on all sampled dendrograms. By sorting the mean probabilities for all unconnected node-pairs in the given network and setting a threshold we can create a list of node-pairs, which are most likely missing connections.

While the approach taken by Clauset et al. has the benefit of avoiding overfitting the observed data, it should be used with caution as not every real network adheres to a hierarchical structure. For strongly assortative networks (e.g. metabolic networks), shortest-path heuristics should be used instead as they provide better prediction quality. Overall,

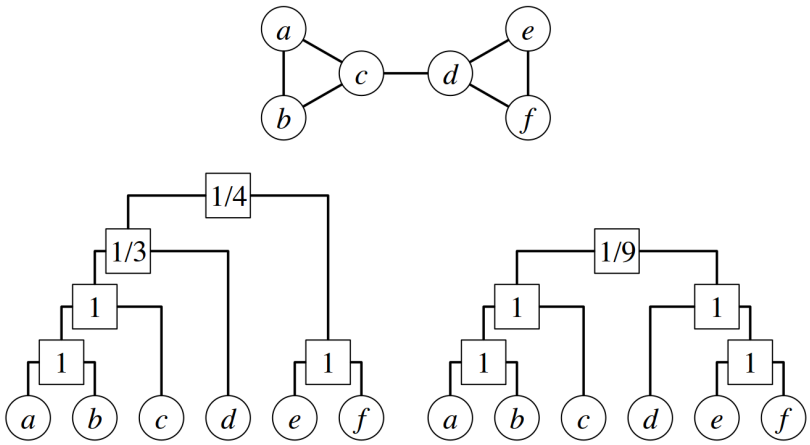

Figure 3.1.: An example network with two corresponding dendrograms.[13]

the hierarchical model is not very flexible and can only be used for a limited set of networks.

3.4. Random-Walk Indices

Given a graph G and a starting node u, a *random walk* is a sequence of nodes that is generated by moving from the starting node to a randomly selected neighbor from where we again move to a randomly selected neighbor and so on [42]. There are a variety of random walk indices used for link prediction.

One of the most fundamental indices based on random walks is *hitting time*. For two given nodes u and v, the hitting time $H_{u,v}$ is defined as the expected number of steps for a random walk starting at node u to reach v [24]. The shorter the hitting time the more similar the nodes u and v which implies there is a higher chance for a future link between the nodes. This index is asymmetric which means the score for the hitting time starting at u might differ from the score obtained when starting the random walk at v. To obtain a symmetric score, the (negated) *commute time* $C_{u,v} = C_{v,u} = H_{u,v} + H_{v,u}$ can be used. A disadvantage of the hitting time are the effects of nodes with higher stationary probability (e.g. a celebrity in social networks). Even though these nodes might be far away from nodes u and v, it might be hard for a random walk to escape their neighborhood which leads to a high variance for the generated scores. This problem can be avoided by allowing the random walk to return to the starting node at every step with a fixed probability α. To account for nodes with very high stationary probability π, the index can be normalized:

$$\text{normalized-hitting-time}(u, v) = -(H_{u,v} \cdot \pi_v + H_{v,u} \cdot \pi_u). \tag{3.6}$$

PageRank [51] is another well known random walk algorithm that has been adapted for link prediction by Chung and Zhao [12]. PageRank was initially proposed to objectively rate webpages with the aim of measuring the human interest and attention devoted to them by utilizing random walks. Called *rooted PageRank* (RPR), PageRank has been adapted for the link prediction problem since the original PageRank is only defined on a single node instead of a node-pair. RPR is defined to be the stationary probability of v in a random walk that returns to u with probability α while moving to a neighbor with probability $(1 - \alpha)$. The rooted PageRank can be calculated for all node-pairs with

$$\text{rooted-page-rank}(A) = (1 - \alpha)(I - \alpha D^{-1}A)^{-1}, \tag{3.7}$$

where $D^{-1}A$ is effectively the adjacency matrix with row sums normalized to 1 with D being a diagonal *degree matrix* ($D_{i,i} = \sum_j A_{i,j}$) and A being the adjacency matrix [24].

Even though random walk indices make use of all the topological information, this comes with the cost of increased computational complexity. This makes it infeasible for large-scale link prediction.

4. Methodology

This chapter presents the methods that are used in the experiments in chapter 6. The chapter begins with a more formal definition of the link prediction problem and, based on this, methods to approach the problem are introduced. These can roughly be divided into unsupervised and supervised approaches. The unsupervised methods are commonly referred to as *similarity indices* and simply represent measurements that can be applied to graphs. Supervised approaches make use of classifiers that can be trained on a number of features.

4.1. Problem Definition

4.1.1. Time-invariant Link Prediction

Let $G = (V, E)$ be an undirected graph, where $V = \bigcup_i V_i$ is the union of various kinds of node-types and $e = \{u, v\} \in E$ represents a single interaction between u and v. There are no self-loops or multi-edges in G. From the given graph, a training subgraph $G_{train} = (V, E_{train})$ and a testing subgraph $G_{test} = (V, E_{test})$ are extracted, where $E_{train} \subseteq E_{test} \subseteq E$. A set $S_{test} \subseteq (V \times V) \setminus E_{train}$ of node-pairs is used for evaluation. We formulate link prediction as a *label prediction problem*, where existing links are labeled as positive instances ("1"), and non-existent links as negative instances ("0"). The goal of link prediction is now to obtain a link prediction model M, built with node-pairs from G_{train}, that can be applied to different node-pairs in G_{test}, predicting their labels. This means there is a function $f_M \colon L \to \{0, 1\}, l \mapsto \begin{cases} 1 & \text{if } l \in E_{test} \\ 0 & \text{otherwise} \end{cases}$ for every model M that generates predictions [68]. Note that the general objective of link prediction is to predict links that are missing from the *observed* network.

If we take protein-protein interactions as an example, predicting a link means finding two proteins that interact with each other but where the interaction has not been observed yet. Besides analyzing missing links in a given dataset, link prediction can also be used to predict links that may appear in the *future*. An example of this would be social networks like Facebook, where potential friends can be suggested to a user by predicting future

friendships. There is some ambiguity regarding this definition. For example, predicting links on Facebook could also be regarded as predicting *missing links* as two persons might be friends but have not added each other as friends on Facebook.

4.1.2. Time-variant Link Prediction

In a concrete implementation of the previous definition, the partitioning of G into G_{train} and G_{test} is fairly easy. An idea would be to randomly select a given percentage of edges from E for E_{test} and again randomly select a given percentage of edges from E_{test} for E_{train}.

A major disadvantage of this technique is that it distorts the actual mechanisms that lead to the creation of new links in the network. This is why this method is only used if there are no timestamps available and there is no way to determine which links were created at which point in time. This is the case for time-invariant networks (e.g. protein-protein interactions). If timestamps are available, the links can be sorted by time of creation and two time-intervals can be used to acquire a E_{train} (older links) and E_{test} (including more recently created links).

More formally, given two timestamps t and t', we define $G[t, t']$ as the subgraph of G that consists of all the edges created between t and t' inclusive, where $t < t'$. To retrieve G_{test} and G_{train}, we pick four timestamps $t_0 < t'_0 < t_1 < t'_1$, where we call $[t_0, t'_0]$ the *training interval*, and $[t_1, t'_1]$ the *testing interval*. The training graph is now defined as $G[t_0, t'_0]$ and the testing graph as $G[t_1, t'_1]$. The goal of a link predictor would now be to predict links, based on the training graph $G[t_0, t'_0]$, that are not present in $G[t_0, t'_0]$ but appear in $G[t_1, t'_1]$. [38]

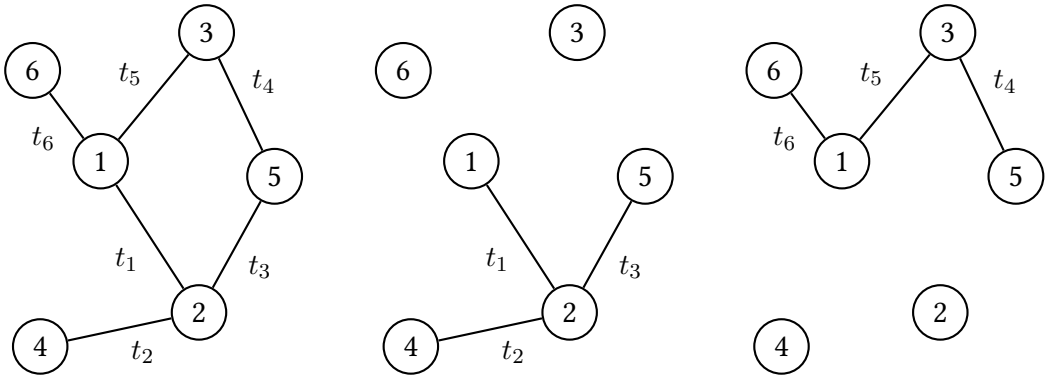

Figure 4.1.: On the left is an initial graph G where every link has an associated timestamp t_i, where $\forall i \in \{1, \ldots, 5\} : t_i < t_{i+1}$. In the center is the training graph $G[t_1, t_3]$ and on the right the testing graph $G[t_4, t_6]$, which consists only of the three most recently created links.

4.2. Similarity Indices

In this section, a number of unsupervised methods to approach the link prediction problem will be presented. All methods assign a *score* to a pair of nodes (u, v) that indicates how likely a connection between the given node-pair is. These scores are based on a given undirected input graph $G = (V, E)$ and a higher score indicates a greater probability that a link exists. These methods are called similarity indices because they compute a measure of proximity or similarity for two nodes using the network topology.

There are a number of ways to categorize similarity indices such as parameter-free vs. parameter-dependent, node-based vs. path-based, and so on [43]. We will divide the indices into intuitive categories, namely *local indices*, *quasi-local indices* and *global indices*. Local indices make use of very limited topological information with respect to a given node-pair, typically only accessing the node-neighborhoods. An example would be the common neighbors of nodes u and v. Quasi-local indices utilize more information than local indices but do not make use of all the available topological data [21]. As the number of steps during the execution increases, quasi-local indices become computationally more expansive (often ending in exponential complexity [41, 43]). Lastly there are global indices, which make use of all the available topological information. This implies the highest computational complexity of all categories but also better prediction quality. For the purpose of comparing the quality of the local indices, we have included one quasi-local index as well as one global index for the experiments.

4.2.1. Local

Common Neighbors (CN) This index is based on the common sense that two nodes, u and v, are more likely to have a link if they have many common neighbors. This is just an implication of the network transitivity property [28]. It was shown that in collaboration networks there exists a positive correlation between the number of common collaborators for researchers u and v, and the probability of a collaboration between u and v in the future [48]. In the context of analyzing large-scale social networks, Kossinets [33] found that there is a high chance for two students having many mutual friends to become friends in the future. More formally, the score can be calculated in the following way:

$$\text{common-neighbors}(u, v) = |\Gamma(u) \cap \Gamma(v)| \tag{4.1}$$

The score for the node-pair (u, v) is also given by $(A^2)_{u,v}$, where A denotes the adjacency matrix of G.

Jaccard Coefficient (JC) This coefficient is a commonly used metric in information retrieval [52] and measures the probability that a common neighbor of u and v is selected if the selection is made randomly from the union of the neighborhoods. This effectively normalizes the Common Neighbors index. The index is defined as

$$\text{jaccard}(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}. \tag{4.2}$$

Adamic/Adar (AA) In 2003 introduced by Adamic and Adar [2], this index refines the counting of common features (e.g. neighbors) by weighting rarer features more heavily. The authors proposed this index in the context of deciding whether individual home pages are related to each other. For a set of features shared by u and v, $shared\text{-}features(u, v)$, the measure is defined as

$$\sum_{z:shared\text{-}features(u,v)} \frac{1}{log(frequency(z))}. \tag{4.3}$$

In the context of link prediction, Liben-Nowell and Kleinberg[38] customized the Adamic/Adar measure in the following way:

$$\text{adamic/adar}(u, v) = \sum_{z \in \Gamma(u) \cap \Gamma(v)} \frac{1}{log(k(z))}. \tag{4.4}$$

Resource Allocation (RA) Zhou et al. [69] proposed this index based on the resource allocation process on complex networks [50]. The index considers two unconnected nodes u and v, where u can send resources to v, with their common neighbors playing the role of transmitters. Assuming that every transmitter has a unit of resource and distributes it equally, we can define the amount of resources that v receives as

$$\text{resource-allocation}(u, v) = \sum_{z \in \Gamma(u) \cap \Gamma(v)} \frac{1}{k(z)}. \tag{4.5}$$

Even though the form of the index is very similar to Adamic/Adar, they were created with different motivations. If the degree $k(z)$ is small, the difference to Adamic/Adar is minimal. For large $k(z)$ the Resource Allocation index will punish these high-degree neighbors more heavily, resulting in a considerable difference [43].

Preferential Attachment (PA) Preferential attachment is a model of the growth of networks [47], with the basic assumption that the probability that a new edge connects to node u is proportional to $k(u)$. This can be used to generate evolving scale-free networks [7] or scale-free networks without growth [63], where each time a new link is added, an old link gets removed. Newman [48] and Barabási et al. [8] found that the probability of co-authorship between two authors correlates with the product of the collaborators of the two authors. More formally, this measure can be defined as

$$\text{preferential-attachment}(u, v) = k(u) \times k(v). \tag{4.6}$$

Hasan et al. [25] also showed that the *summation* of the neighbor-count of a node-pair is a good similarity index as well. Due to the fact that the actual neighbors are not required, this index has the least computational complexity [43].

Adjusted Rand (AR) First introduced by Hubert and Arabie in 1985 [29], this index was found to be the best method for the comparison of partitions of a finite set of objects. More recently, the Adjusted Rand index was proposed by Hoffman et al. [27] to be used in the link prediction domain. The index is formally defined as

$$\text{adjusted-rand}(u, v) = \frac{2(ad - bc)}{(a + b)(b + d) + (a + c)(c + d)}, \tag{4.7}$$

where the values of a to d can be obtained from the following contingency table:

	u		
v	adjacent	not adjacent	total by v
adjacent	a	c	$k(v)$
not adjacent	b	d	$\|V\| - k(v)$
total by u	$k(u)$	$\|V\| - k(u)$	$\|V\|$

In the above table, c, for example, is the number of nodes that are *not* adjacent to u but adjacent to v. The index produces scores in the range $[-1, 1]$, where a score greater than zero indicates that the probability that a link u, v exists is above random chance [27]. A consequence of this fact is the advantage of using zero as a threshold to differentiate whether a link will form or not.

Neighborhood Distance (ND) This index was used by Wu et al. [62] to address the hop-distance ambiguity in range-free localization for wireless ad hoc and sensor net-

works, where a node has a same distance estimation to all of its neighbors. This makes it a strong candidate for link prediction, due to the similar goals. Here, we will use a modified version, which is mirrored on the x-axis and omits any terms that do not influence the order of scores:

$$\text{neighborhood-distance}(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{k(u) \times k(v)}}, \tag{4.8}$$

To the knowledge of the author, this measure hasn't been used before in the link prediction domain, and because of this, deserves special attention.

4.2.2. Quasi-local

Neighbors-Measure (NM) This measure was first introduced by Fire et al. as *Friends-Measure* [18], referring to its usage in social networks. The index is based on the intuitive assumption that the more connections there are between the neighborhoods of two nodes, the more likely it becomes that the two nodes are connected. This idea is very similar to the *k-nearest neighbors* (k-NN) algorithm used in supervised learning. In this case the parameter k is flexible and two nodes are "near" to each other if they are connected. The formal definition for the Neighbors-Measure index is

$$\text{neighbors-measure}(u, v) = \sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \delta(x, y), \tag{4.9}$$

where $\delta(x, y) = \begin{cases} 1 & \text{if } x = y \text{ or } x, y \in E \\ 0 & \text{otherwise} \end{cases}$. This is effectively a special case of the following Katz index, where $\beta = 1$, $l_{min} = 2$ and $l_{max} = 3$. In contrast to local indices, this measure makes use of an extended neighborhood, which qualifies it as an quasi-local index.

4.2.3. Global

Katz (KA) One of the first global indices that was adopted for the link prediction problem is the Katz index. Leo Katz [30] proposed this index in order to provide a more viable method for determining the status of a person in inter-personal and inter-group relations. This is done by not only considering direct relations but also the influence through indirect relations [67]. For a given network, this means that the index considers the number of paths of varying lengths between two nodes. Longer paths are damped and contribute less to the score than shorter paths. More formally,

the behaviour of the Katz index can be defined as

$$\text{katz}(u, v) = \sum_{l=1}^{\infty} \beta^l |paths_{uv}^{<l>}|. \tag{4.10}$$

In the above definition, $paths_{uv}^{<l>}$ denotes the set of paths with length l connecting u and v. The free parameter β is used as a damping-factor with a value less or equal to 1, in order to exponentially damp the contribution of longer paths to the overall score by β^l. Even though the Katz index generally beats local indices in terms of prediction quality, it comes with the cost of cubic runtime complexity which isn't feasible for large networks [24]. As it would be computationally infeasible to consider all paths between two nodes in a network, the maximal path length to consider will be set to $l_{max} = 5$. The damping factor will be set to $\beta = 0.005$.

4.3. Supervised Learning

In recent years, an increasing amount of researchers [4, 25, 14, 39] in the link prediction community have explored the possibility of using supervised learning to approach the link prediction problem. If there is the possibility to obtain ground truth from part of a network, there is no practical disadvantage to the use of supervised methods. In fact, even if a supervised method only makes use of a single unsupervised index as a feature, it could still outperform this index.

For the purpose of scalable classification, we restrict our usage of supervised methods to the ensemble methods bagging and boosting as well as the naive Bayes classifier. Bagging and boosting will both be used on decision trees, which are generally not very strong classifiers on their own.

In the training stage, we select training samples for a set of node-pairs. A sample is a pair consisting of a feature-vector and its ground truth class. We use the class label "0" for an absent link and "1" for an existing link. The feature-vectors will be created from a number of unsupervised methods (Common Neighbors etc.) or topological properties (see 4.3.3), whose inputs are the node-pairs. After the classifiers have been trained, they can be used to generate a class label for a given feature-vector.

4.3.1. Classifier

In the following, the used classifiers will be briefly explained and their properties presented. There will also be remarks about the concrete parameters used in the experiments.

4.3.1.1. Decision Tree

Decision tree learning is a simple method to learn a discrete-valued target function, where the learned function is represented by a decision tree. Figure 4.2 depicts an exemplary decision tree. The maximal depth of the shown tree is set to three and the feature-vector is represented through the features F-1 to F-10. The inner nodes consist of a *decision rule* and also show the value of the gini-criterion which is used to determine how "good" a specific split actually is. Below the gini-criterion there are the number of training samples that were assigned to the node based on the previous decisions. The leafs contain a value-array which shows the number of training samples for all the classes. In this case there are only two classes.

If we want to classify a concrete node-pair (u, v) with ten discrete feature-extractors, we would construct its feature vector $f_{u,v} = [f_1(u, v), f_2(u, v), \ldots, f_{10}(u, v)]$, which could be $[153.0, 0.1242, 0.3937, 290.0, \ldots, -0.9834, 84.0, 0.0038]$ for example, and feed it to the decision tree classifier. We now consider the root node and see that $f_4(u, v) = 290 \leq 326.5$ holds true. As a consequence, we proceed with the left subtree. We end up in the third leaf by continuing this pattern. Considering the majority vote in respect to the classes of the training samples that ended there, 689 of the first class and 396 of the second class, we assign (u, v) the first class, which is in our case equal to "absent link".

In the experiments, the CART (Classification And Regression Tree, see [10] for details) decision tree algorithm is used in conjunction with the gini-criterion. The used decision trees are also *unpruned*, which means there are no sections of the trees removed in order to reduce their complexity.

The decision tree classifier will only be used in conjunction with the ensemble methods bagging and boosting. Even though a decision tree is fast to compute, the predictive qualities of a single decision tree are not sufficient for our purposes.

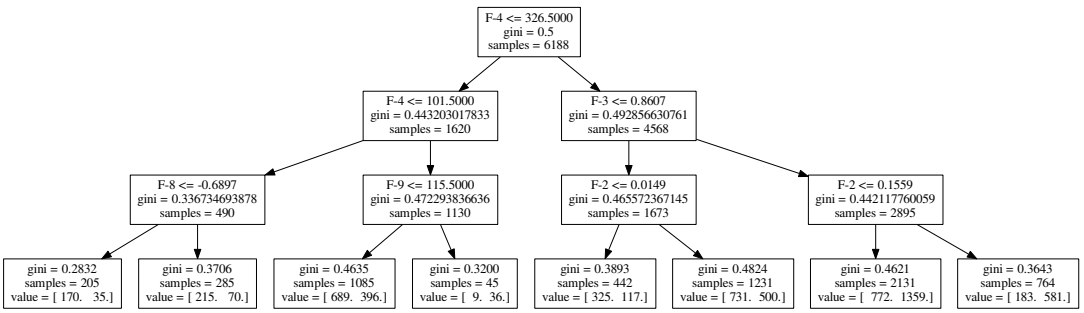

Figure 4.2.: Decision tree produced by the Python module sci-kit learn.

4.3.1.2. Naive Bayes

Naive Bayes classifiers are a family of machine learning algorithms based on Bayes' theorem with the *naive* assumption that all given features are pairwise independent of each other. Figure 4.3 demonstrates this as there are no connections between features which would represent dependences between them. This naive independence assumption allows it to easily combine the contributions of the different features without having to worry about how the features influence each other. Bayes' theorem states the following relationship:

$$P(y|f_1, \ldots, f_n) = \frac{P(y)P(f_1, \ldots, f_n|y)}{P(f_1, \ldots, f_n)}. \tag{4.11}$$

If we make use of the independence assumption

$$\forall i \in \{1, \ldots, n\} \colon P(f_i|y, f_1, \ldots, x_{i-1}, f_{i+1}, \ldots x_n) = P(f_i|y), \tag{4.12}$$

the equation 4.11 can be simplified by using the chain rule to

$$P(y|f_1, \ldots, f_n) = \frac{1}{P(f_1, \ldots, f_n)} P(y) \prod_{i=1}^{n} P(f_i|y). \tag{4.13}$$

As $P(f_1, \ldots, f_n)$ only depends on the input vector, it is constant and does not need to be taken into consideration. This gives us the final equation for a naive Bayes classifier:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(f_i|y). \tag{4.14}$$

Here, the probability of any possible output class y gets calculated based on the given feature-vector. Finally, the corresponding node-pair gets assigned the class $\hat{y}$, which has the highest probability.

Even though $P(y)$ is simply given by the relative frequency of class y in the training set, there needs to be an assumption about the distribution of $P(f_i|y)$. In our case, the Gaussian distribution will be used:

$$P(f_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(f_i - \mu_y)^2}{2\sigma_y^2}\right). \tag{4.15}$$

The parameters μ_y and σ_y can be estimated using *maximum likelihood estimation* (MLE). A classifier using this distribution is also called *Gaussian Naive Bayes* classifier.

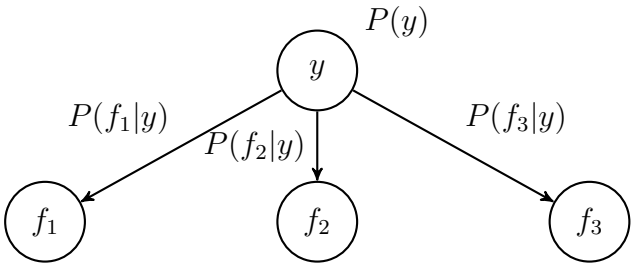

Figure 4.3.: Naive Bayes classifier assuming independent features.

4.3.2. Ensemble Methods

The goal of ensemble methods is to improve prediction quality and stability by combining a number of "black box" base estimators that are built based on a specific learning algorithm. The term black box refers to the fact that the estimators can be called repeatedly but their properties cannot be observed or manipulated [53]. For a lot of base estimators, this leads to stronger performance in comparison to using a single estimator instance.

In general, there is a distinction between two families of ensemble methods: *averaging methods* and *boosting methods*. The idea of averaging methods is to construct a set of independent estimators and to combine their predictions by averaging them into a final output. The main advantage of this technique is the reduced variance of its output in respect to the variance of the single estimators. In this thesis we will make use of *bagging*, one of the most widely used averaging methods. Boosting methods try to enhance the performance of weak estimators by sequentially training estimators on subsets of the initial training data.

4.3.2.1. Bagging

Bagging was proposed by Breiman [9] in 1996 and its idea is to build a number of base estimators on random subsets of the initial training data and average over the individual results to obtain an aggregated score. This can also be regarded as a *committee* of estimators each casting a vote for the predicted class [26]. As Breiman notes, this works very well for unstable but strong classifiers like decision trees or neural networks, yielding substantial improvements in overall prediction quality. Bagging methods are also less susceptible to overfitting, which favors the use of decision trees as well as they tend to overfit. On the other hand, bagging can slightly degrade the performance of very stable classifiers that have low variance in their predictions. As a consequence, in this thesis bagging is used by averaging the results of 25 unpruned CART decision trees whose input consists of random subsets of 20% of the features. Using only a random subset of

the feature-space is also known as *Random Subspaces* [54]. Figure 4.4 demonstrates the effect of bagging by comparing three neural networks on the left with the result of ten bagged neural networks on the right for a two-dimensional feature space with two different classes.

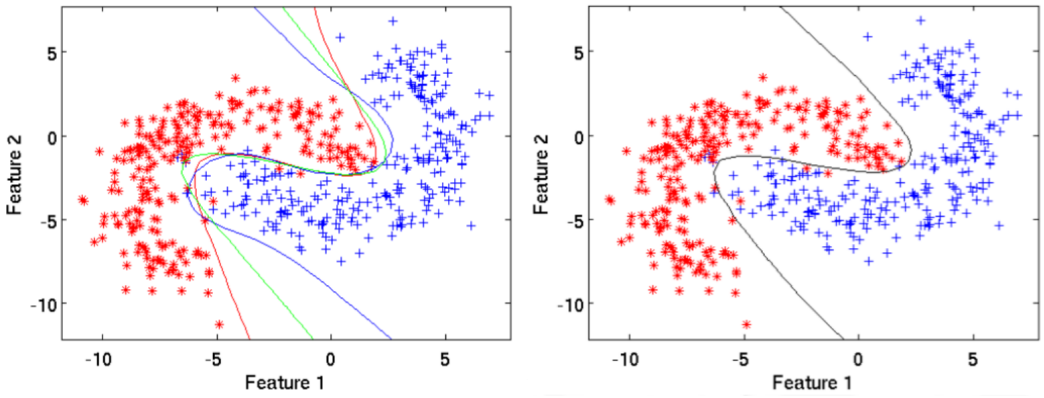

Figure 4.4.: Comparison between three neural networks on the left and the result of ten bagged neural networks on the right. [66]

4.3.2.2. Boosting

Boosting was first introduced in 1990 by Schapire and Freund [20] through the presentation of *AdaBoost*, which will be used during the experiments as well. The key idea in boosting is to choose training sets for the base estimator in such a way as to force the estimator to infer something new about the data each time it is called [53]. This is done by applying weights $w_1, w_2, \ldots, w_N$ to each of N training samples provided. Initially, those weights are set to $w_i = \frac{1}{N}$ ($i \in \{1, \ldots, N\}$), so that in the first step the classification is simply performed on the original training set [26]. In each successive step the weights will be altered and the classifier will be reapplied to the weighted data. This way classifiers that perform only slightly better than random chance can be combined into a single, strong classifier. Figure 4.5 illustrates a schematic of AdaBoost, where classifiers are trained on weighted samples of the training set and then combined to produce a final prediction. The majority vote $G(x)$ is produced by weighting the contribution of each respective classifier $G_m(x)$ by α_m, where M is the number of weak classifiers and $m \in 1, \ldots, M$.

In contrast to bagging, boosting works best with weak classifiers like shallow decision trees. During the experiments we use an AdaBoost classifier with fifty unpruned CART decision trees.

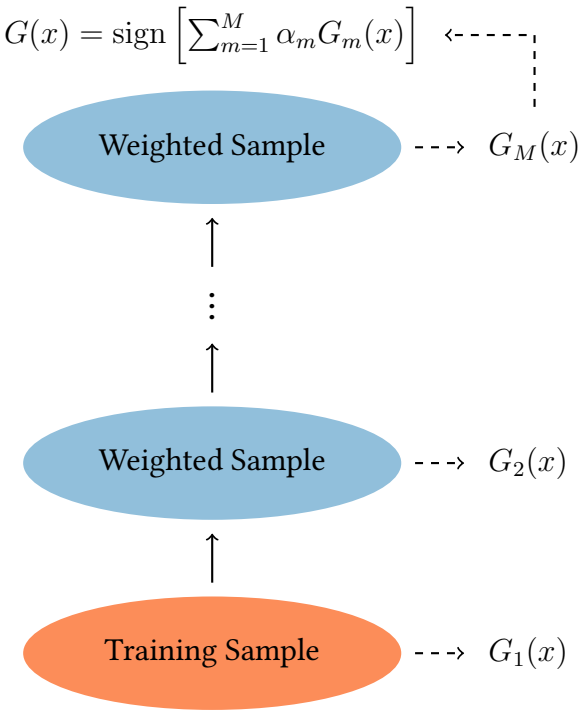

Figure 4.5.: Schematic of AdaBoost.

4.3.3. Features

In this subsection the process of feature selection for supervised learning will be discussed and new features will be introduced. The higher the number of features, the more potential there is to make use of for classifiers. They can infer which features are good predictors for the label of a node-pair. But especially in the scope of scalable link prediction, the potential contribution of a feature has to be weighed against its computational costs. For this reason, the global Katz index as well as the quasi-local Neighbors-Measure will not be used as features in supervised learning. They are computationally very expensive and thus would heavily extend the time of training and, more importantly, actual classification. On the other hand, there are measures that are inexpensive but not worth exploring on their own. This could for example simply be the number of actual neighbors of a node. This is not a good measure on its own, but it is computationally cheap and thus suitable as a feature for supervised learning. In conjunction with the local indices introduced previously, the following new features will be used for supervised learning:

Total-Neighbors The number of neighbors in the neighborhood-union of nodes u and v:

$$\text{total-neighbors}(u, v) = |\Gamma(u) \cup \Gamma(v)|. \tag{4.16}$$

Node Degree The degree of a node can give an indication about how likely it is to connect to other nodes. For this reason, $\text{degree}_\text{u}(u, v) = k(u)$ and $\text{degree}_\text{v}(u, v) = k(v)$ can be used as two distinct features for a given node-pair (u, v).

Same Community Let C be the set of all communities for a given graph $G = (V, E)$ and $\forall C' \in C : C' \subseteq V$. Then we can define the following measure:

$$\text{same-community}(u, v) = \begin{cases} 1 & \text{if } \exists\, C' \in C : u, v \in C' \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

The actual implementation is the Parallel Louvain Method proposed by Staudt and Meyerhenke [55].

Algebraic Distance The algebraic distance between two nodes was first proposed by Chen and Safro [11] and defines a structural distance between nodes. Let $w_{u,v}$ denote the non-negative weight of edge $\{u, v\}$ in graph $G = (V, E)$. Then we can, given a parameter ω and an initial random-vector $x^{(0)} \in \mathbb{R}^{|V|}$, define the preprocessing Algorithm 1.

Algorithm 1 Algebraic Distance Preprocessing

1: $\tilde{x}_u^{(k)} \leftarrow \sum_v w_{u,v} x_v^{(k-1)} / \sum_v w_{u,v}, \forall u \in V.$
2: $x^{(k)} \leftarrow (1 - \omega) x^{(k-1)} + \omega \tilde{x}^{(k)}$

This preprocessing algorithm can be independently used on R initial vectors $r^{(0,r)}$ ($r \in \{1, \ldots, R\}$) and, based on this, the actual extended 2-normed algebraic distance can be defined as

$$\text{algebraic-distance}(u, v) = \left(\sum_{r=1}^{R} \left| x_u^{(k,r)} - x_v^{(k,r)} \right|^2 \right)^2, \tag{4.18}$$

where $x^{(k,r)}$ denotes the *k-th* iteration of the *r-th* initial random vector.

To get an overview, Table 4.1 shows all the previously presented methods and their usage throughout the experiments.

Name	Unsupervised	Supervised
common-neighbors(u, v)	✓	✓
jaccard(u, v)	✓	✓
adamic/adar(u, v)	✓	✓
resource-allocation(u, v)	✓	✓
preferential-attachment(u, v)	✓	✓
adjusted-rand(u, v)	✓	✓
neighborhood-distance(u, v)	✓	✓
neighbors-measure(u, v)	✓	
katz(u, v)	✓	
degree_u(u, v)		✓
degree_v(u, v)		✓
same-community(u, v)		✓
total-neighbors(u, v)		✓
algebraic-distance(u, v)		✓

Table 4.1.: Feature listing

4.4. Evaluation

Evaluating methods is one of the most challenging tasks in link prediction. Most networks are sparse, which means only a small fraction of all the *possible* links are actual links. Saying that a network is sparse is equivalent to saying that the network has a low density or that there is a high class imbalance in the network. High class imbalance just highlights the fact that there are an enormous amount of class 0 instances (no link) in comparison to class 1 instances (existing link) in a network.

Typically the evaluation methods used in link prediction are the same that get used in a classical binary classification problem. All metrics have in common that they expect an ordered list of scores to work on. Those scores are in the case of link prediction generated by the previously introduced unsupervised and supervised methods for a given set of node-pairs. By utilizing a *discrimination threshold* T, there is an easy way to classify the node-pairs based on their scores. If the scores are sorted descendingly, all the node-pairs with scores above T will be regarded as links and all the node-pairs with scores below T as absent links. This would be called a *fixed-threshold* metric and the classification could simply be evaluated by comparing the predicted classes to the actual classes. This is generally not a good idea because most of the time there is no reasonable estimation about a good threshold available. *Threshold curves* provide a good alternative and are used throughout the research community. In this case the threshold T is varied and for every threshold there is a concrete classification of the node-pairs that can be compared against the actual classes. Figure 4.6 demonstrates this variation on a number of node-pairs with corresponding scores.

Node-pair	Score	
(0, 2)	9.84	
(5, 9)	9.43	
(3, 4)	9.29	
(0, 5)	8.40	$T = 8.5$
(5, 7)	8.03	
(3, 7)	7.27	
(4, 8)	5.16	
(1, 5)	5.13	
(1, 3)	5.02	
(6, 8)	4.12	

Figure 4.6.: Varying decision threshold for concrete link predictions.

In Figure 4.6 the entries are sorted descendingly by score and the decision threshold with value 8.5 is marked as a red line and divides the predictions that will be treated as links (above threshold) and those that will be treated as absent links (below threshold). Varying the threshold by increasing/decreasing its value will thus remove/add predicted links.

A metric that generates a threshold curve would now incorporate every concrete classification into the curve by evaluating different statistical measures, depending on the actual metric. The metrics used during this thesis will be explained in detail in the next subsection.

The used metrics utilize a common set of statistical measures to produce threshold curves: *true positives* (TP), *false positives* (FP), *true negatives* (TN) and *false negatives* (FN). The number of *positives* P is thus given by $P = TP + FN$ and the number of *negatives* N is $N = TN + FP$. Figure 4.7 shows the measures in the link prediction domain.

	Actual	
Prediction	Link	No link
Link	True positive	False positive
No link	False negative	True negative

Figure 4.7.: Confusion matrix for link prediction.

To demonstrate the calculation of the statistical measures for a concrete set of predictions, Figure 4.8 shows the statistical categorization of the predictions by comparing them to the ground truth.

Finally, there is the need to compress the information provided through the threshold curve into a single measurement. This allows for an easier interpretation of the evaluation results and permits the comparison between different curves. In this thesis the *area under the curve* (AUC) is used. Even though this indicates the performance of a classifier, we should still be cautious not to completely reduce the performance of a classifier to its AUC. Even though the performance of two classifiers might yield the same AUC, this does not mean that the classifiers actually perform identical. They can still rate the same node-pairs differently and they could also have different true positive rates for the same false positive rate.

Typically, numerical integration is used to approximate the area under a given threshold curve. For this purpose, the *trapezoidal rule* is used throughout the experiments to determine the AUC. In case the threshold curve should be fitted or smoothed (e.g. to reduce noise) as in Figure 4.9, there is often a functional description of the fitted curve

Node-pair	Score	Actual link	**Outcome**
(0, 2)	9.84	✓	TP
(5, 9)	9.43	✓	TP
(3, 4)	9.29	✓	TP
(0, 5)	8.40		FP
(5, 7)	8.03	✓	TP
(3, 7)	7.27		FP
(4, 8)	5.16		TN
(1, 5)	5.13	✓	FN
(1, 3)	5.02		TN
(6, 8)	4.12		TN

$T = 6.2$

Figure 4.8.: Classification outcomes for threshold $T = 6.2$ on the example data from Figure 4.6. The red background for rows indicates incorrect predictions like false positives or false negatives whereas a green background indicates successful predictions like true positives and true negatives.

available. In this case, the AUC can also be calculated through the definite integral on $[0, 1]$.

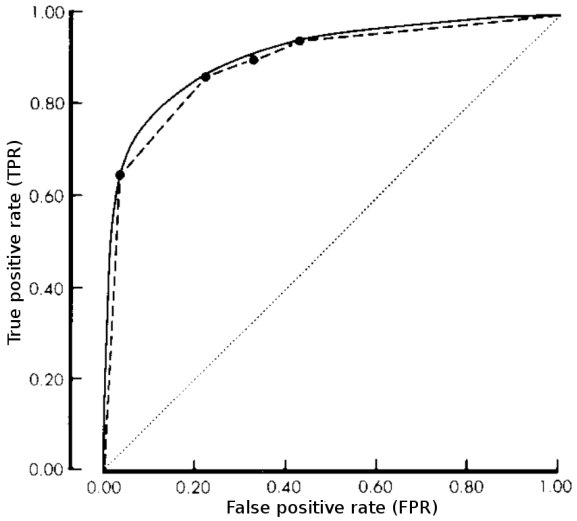

Figure 4.9.: Gaussian-based curve to smooth a threshold curve (ROC). [23]

The advantage of obtaining the AUC with the trapezoidal rule is that the resulting scalar is always smaller than any smoothed curve and that it is more sensitive to the location and spread of the points defining the curve [23]. This way we do not overestimate the actual AUC value.

As we are also interested in the performance of link predictors in comparison to a random classifier, all the generated plots in the experiments will also contain a dashed line, which indicates the performance of a random classifier.

4.4.1. Threshold Curves

4.4.1.1. Receiver Operating Characteristic

The receiver operating characteristic (ROC) curve is currently the most popular evaluation metric in the link prediction community. A major reason for this is the fact that the area under the ROC (AUROC) is not influenced by the class distribution of the input it is used on [18]. The metric plots the *false positive rate* (FPR) against the *true positive rate* (TPR), where

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{4.19}$$

and

$$FPR = \frac{FP}{P} = \frac{FP}{TP + FN}. \tag{4.20}$$

TPR is also known as *sensitivity* or *recall*, and FPR as *fall-out* or *1 - specificity*.

A perfect link predictor would have a sensitivity of 100% (all links in the test set would be predicted) and a fall-out of 0% (all absent links in the test set would *not* be predicted as links) which is equivalent to the point $(0, 1)$ in the ROC space depicted in Figure 4.10.

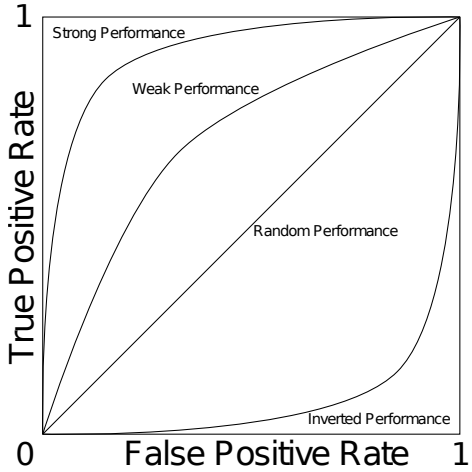

Figure 4.10.: ROC space and four curves indicating a random, strong, inverted, and weak performance. The inverted performance refers to a strong performance that has been inverted.

The worst possible classifier would thus be a random classifier which is characterized by a diagonal line from $(0, 0)$ to $(1, 1)$. A classifier whose curve is below the diagonal

could simply be inversed by treating predicted links as *absent* links and vice versa. This would be equal to a reflection of the curve with respect to the diagonal.

The area under the ROC curve can also be interpreted as the probability that a randomly chosen link from the input is given a higher score than a randomly chosen non-existent link from the input with respect to the ground truth [43]. If there are n independent comparisons, and n_h occurrences where an actual link has been ranked higher than an absent link against n_e cases where an actual link ranks identical to an absent link, the AUC can be defined as

$$AUC = \frac{n_h + \frac{1}{2}n_e}{n}. \tag{4.21}$$

The ROC metric meets our needs very well for balanced or slightly imbalanced data sets with relatively small sizes ($[10^3, 10^6]$). This is no problem if we can select a subset of all the node-pairs that are currently not connected and where a relatively large percentage ($> 2\%$) will be connected in the ground truth. A problem occurs if we try to use the metric on extremely imbalanced data sets. Take for example link prediction on a typical sparse graph. There will be one newly created link for millions of unestablished links. If a link predictor now tries to predict future links, there is a high chance that the predictor will predict most of the positive instances correctly, but also that it will predict way more negatives incorrectly as positives as well. Even though this is a bad prediction performance (very few correct positives under all node-pairs classified as positive), it will yield a very high AUC, as there is a very high true positive rate but also a very low false positive rate, because there are so much negatives. This effect will most likely occur in deployment scenarios, as Lichtenwalter et al. [39] showed, that the imbalance ratio of a network is *lower* bound by the number of its vertices. For this reason, the Precision-Recall metric can provide better insights about predictor performance.

4.4.1.2. Precision-Recall

The Precision-Recall (PR) metric shows the *precision* of a prediction as a function of the recall, where the precision is defined as

$$Precision = \frac{TP}{TP + FP} \tag{4.22}$$

and recall is defined as in 4.19. Figure 4.11 visualizes precision and recall with respect to the ground truth and a number of positively predicted data points.

The performance of a random predictor is equal to the *baseline*, which is simply the percentage of positives in the given set of node-pairs to use for prediction. This way,

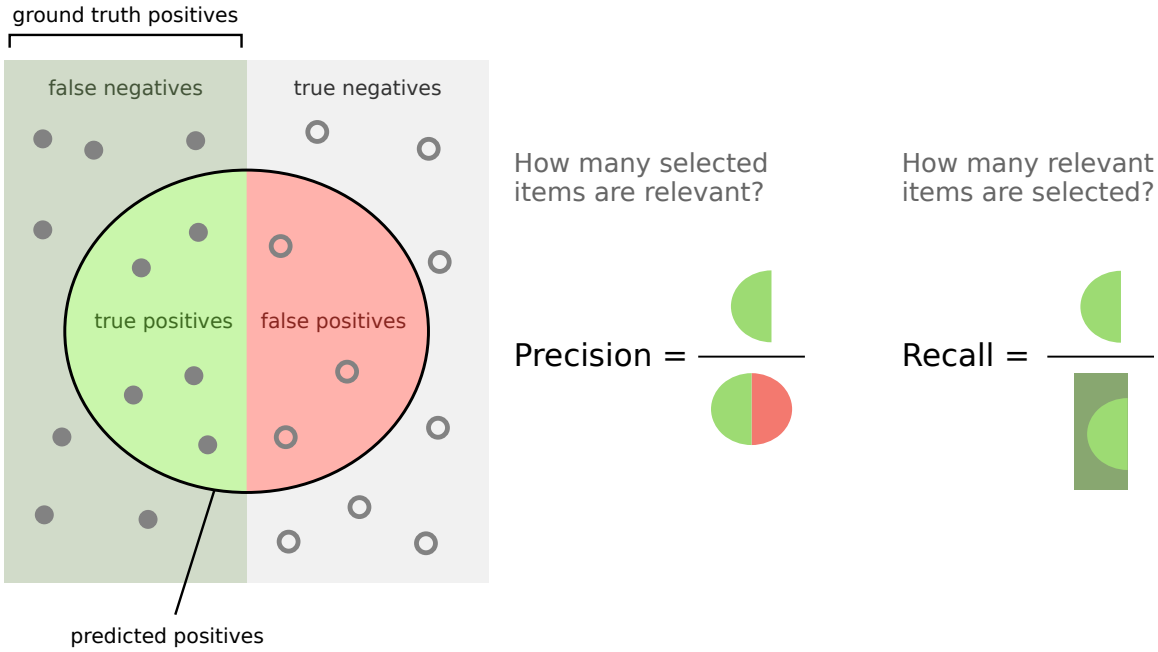

Figure 4.11.: Visualized precision and recall. Modification based on [61].

we can categorize different curves as very strong or nearly random predictors, as is done in Figure 4.12. The perfect score would occur at 100% recall and 100% precision, which corresponds to the point $(1, 1)$ in the PR space. The area under the Precision-Recall curve (AUPR) is once again used as a scalar measure for performance. It should be noted that there is an expected decline regarding the AUPR for an increasing imbalance. This is due to increasing baseline difficulty.

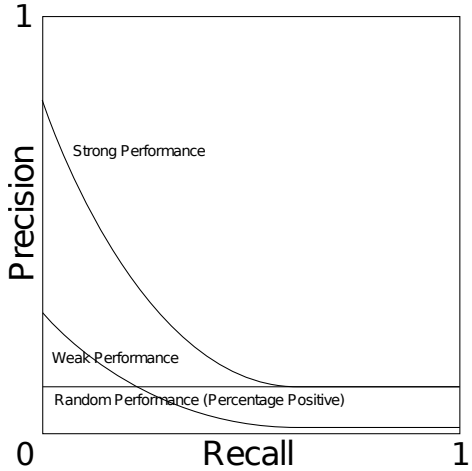

Figure 4.12.: Precision-Recall space and three curves indicating a random, strong, and weak performance.

5. Implementation

Within the scope of this thesis, a link prediction module has been developed for the open-source NetworKit [56] toolkit for high performance network analysis. NetworKit makes use of Cython, which is a Python to C/C++ source code translator, to make C++ code available through a Python module. This approach combines the productivity and flexibility of Python with the performance of C++ and allows for the usage of external C++ libraries as well as Python modules. To make use of supervised methods, the link prediction module works in conjunction with the sci-kit learn Python module.

This chapter will familiarize the reader with the structure of the module and also inform about implementation details and parallelization efforts.

5.1. Module Structure

There are three major problems when conducting experiments in link prediction: preprocessing, actual prediction, and evaluation. During the preprocessing, the graph will be read from disk and the goal is to extract a testing and training set. Afterwards, depending on whether unsupervised or supervised methods are used, there will be a set of predictors used to generate scores for the given node-pair sets. Finally, the predictions generated with the obtained scores will be compared against ground truth labels to determine the prediction quality.

This inherent partitioning into those three components can also be found in the structure of the link prediction module. The classes can roughly be divided into link/graph manipulation operations (LGM), link predictors (LIP), and evaluation metrics (EVM). Classes in the LGM group are used for preprocessing tasks specific to link prediction. This includes the partitioning into subgraphs and the extraction of training and testing set. The LIP group provides the predictors that are used to generate scores for given node-pairs. Lastly, classes in EVM are used to generate threshold curves during evaluation.

The three components in the implementation are decoupled, making it easy to add new preprocessors, link predictors or evaluation metrics. The heart of the implementation is the `LinkPredictor` class, which is an abstract base class for similarity indices. The class

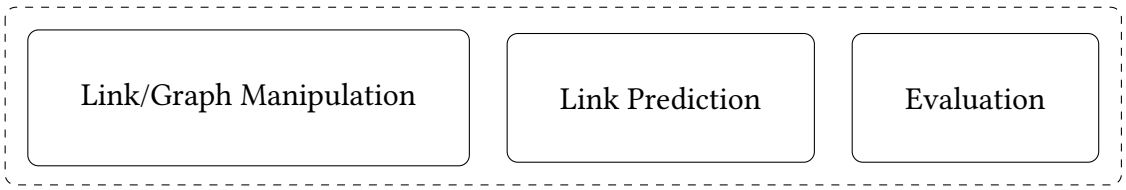

Figure 5.1.: Link prediction module structure.

provides three overridable methods to generate prediction scores: `run`, `runOn`, and `runAll`. The first method can be used to generate a score for a single given node-pair (u, v). The `runOn` method expands upon this by taking a vector of node-pairs to generate scores for. This is done by making use of parallelization. Finally, `runAll` considers all the node-pairs that are not connected and generates scores for them.

A description of the core classes that are in the module can be found in Table 5.1.

	Class	Description
LGM	MissingLinkFinder	Provides methods to find missing links for a given distance.
	RandomLinkSampler	Has methods to randomly sample links from a graph.
LIP	AdamicAdarIndex	Implements definition 4.4.
	AdjustedRandIndex	Implements definition 4.7.
	AlgebraicDistanceIndex	Implements definition 4.18.
		⋮
EVM	PrecisionRecallMetric	Implements the PR metric (see 4.4.1.2).
	ROCMetric	Implements the ROC metric (see 4.4.1.1).

Table 5.1.: Description of core classes in the link prediction module. The classes are associated with their respective categories (LGM, LIP, and EVM).

5.2. Parallelization

There are multiple efforts in the link prediction module to make use of parallelization through OpenMP. Luckily, the scores for node-pairs do not depend on each other and can thus be calculated independently. This implies there are no waiting times between calculations. This problem is also called *embarrassingly parallel* because there is no partitioning into subproblems necessary. Experiment 6.3.4.2 shows the speedup achieved through parallelization and 7.3 discusses the obtained results.

5.3. Usage

In this section, the workflow during link prediction in Python will be described. There can be differentiated between using unsupervised methods and using supervised methods with the sci-kit learn module.

5.3.1. Similarity Indices

One of the most straight-forward ways of using the link prediction module is to simply get the node-pairs with the highest scores. The following code will calculate the Common Neighbors index for all the missing links in the Jazz graph. The output will be the five node-pairs with the highest score.

Listing 1: Top five predictions for CN on Jazz.

```
from networkit import linkprediction as lp, readGraph, Format
G = readGraph("jazz.graph", Format.METIS)
predictions = lp.CommonNeighborsIndex(G).runAll()
lp.LinkThresholder.byCount(predictions, 5)
# Returns [(6, 53), (53, 135), (59, 169), (59, 177), (135, 194)]
```

The following code expands upon the previous example by also evaluating the Adamic/Adar index on Jazz. This is done by randomly sampling 90% of the initial edges to be used in the training graph. The Adamic/Adar index will then calculate the scores for all the node-pairs that are two hops apart in the training graph. The evaluation is done by creating a new `ROCMetric` that obtains its ground truth from the testing graph.

Listing 2: Evaluating Adamic/Adar with ROC on Jazz.

```
from networkit import linkprediction as lp, readGraph, Format
testGraph = readGraph("input/jazz.graph", Format.METIS)
trainingGraph = lp.RandomLinkSampler.byPercentage(testGraph, 0.9)
testingSet = lp.MissingLinksFinder(trainingGraph).findAtDistance(2)
predictions = lp.AdamicAdarIndex(trainingGraph).runOnParallel(testingSet)
rocMetric = lp.ROCMetric(testingGraph)
falsePositives, truePositives = rocMetric.getCurve(predictions)
AUROC = rocMetric.getAreaUnderCurve() # AUROC = ~0.88
# Draw ROC curve using matplotlib...
```

5.3.2. Supervised Learning

The link prediction module provides some helper methods to simplify the training of a classifier. The following code makes use of decision tree bagging to predict labels for the determined testing set. Training a sci-kit learn classifier can easily be accomplished by using the helper function `trainClassifier`. The method produces the necessary features for the given node-pairs and determines their ground truth labels. The classifier is then trained with this data and can be used afterwards to predict labels for new node-pairs. The training graph consists of 90% of the initial edges and the feature graph consists of 70% of the training graph edges. All edges are selected randomly. The testing and training set are the two-hop missing links in the training and feature graph respectively.

Listing 3: Decision tree boosting on Jazz.

```
from networkit import linkprediction as lp, readGraph, Format
from sklearn import ensemble
testGraph = readGraph("input/jazz.graph", Format.METIS)
trainingGraph = lp.RandomLinkSampler.byPercentage(testGraph, 0.9)
featureGraph = lp.RandomLinkSampler.byPercentage(testGraph, 0.7)
# Select all 2-hop missing links in train./feat. graph as test./train. set
testingSet = lp.MissingLinksFinder(trainingGraph).findAtDistance(2)
trainingSet = lp.MissingLinksFinder(featureGraph).findAtDistance(2)
# Default estimator for AdaBoost is DecisionTreeClassifier
adaBoostClassifier = ensemble.AdaBoostClassifier(n_estimators=50)
# Select predictors to use as feature extractors and train classifier
LPs = [lp.JaccardIndex(featureGraph), lp.KatzIndex(featureGraph)]
lp.trainClassifier(trainingSet, trainingGraph, adaBoostClassifier, *LPs)
predictions = adaBoostClassifier.predict(lp.getFeatures(testingSet, *LPs))
```

6. Experiments

6.1. Setup

The experimental setup is an important step to ensure that the experiments are executed correctly and that the obtained results are meaningful. The aim of this section is to describe the process leading to the selection of a training and testing set. The testing set is used to evaluate the performance of the similarity indices as well as the supervised methods. The training set is used to train the supervised methods. In the first step, the initial graph will be partitioned into subgraphs that can in return be used to obtain the required sets for testing and training.

The basic procedure in the first step is to partition the initial graph $G = (V, E)$ into three subgraphs, which will be the testing graph $G_{test} = (V, E_{test})$, training graph $G_{train} = (V, E_{train})$, and feature graph $G_{feat} = (V, E_{feat})$ with $E_{feat} \subseteq E_{train} \subseteq E_{test} \subseteq E$:

testing graph Provides ground truth for node-pairs that will be used to evaluate a predictor.

training graph Provides ground truth for the node-pairs that will be used to train a supervised method. This graph is also used by the feature extractors (see Table 4.1) to obtain a feature set that is needed for the supervised methods during testing.

feature graph This graph will be used by the feature extractors to generate features for the training set.

The testing graph will be equal to the given graph G. The training graph will be a subgraph of the test graph. If not noted otherwise, the training graph will consist of 90% of the edges of the testing graph, which means $|E_{train}| \approx 0.9 \times |E_{test}|$. The feature graph will again consist of 70% of the edges from the *training* graph: $|E_{feat}| \approx 0.7 \times |E_{train}|$. If the network is time-variant, the edges will be ordered by date of creation and the most recently created edges will be in the testing graph, followed by the edges created before them, which go into the testing graph. The remaining edges (oldest) will be in the feature graph. For time-invariant graphs, the edges will be randomly selected for every

partitioning. The partitioning into testing, training, and feature subgraphs is clarified in Figure 6.1. In the figure, the orange part denotes the edges that can be used as positive instances for the testing set. The yellow part consists of edges that can be used as positive instances in the training set. The edges are ordered by time of creation, where the most recently created edges are on the right. If the initial graph is time-invariant, the edges are randomly divided.

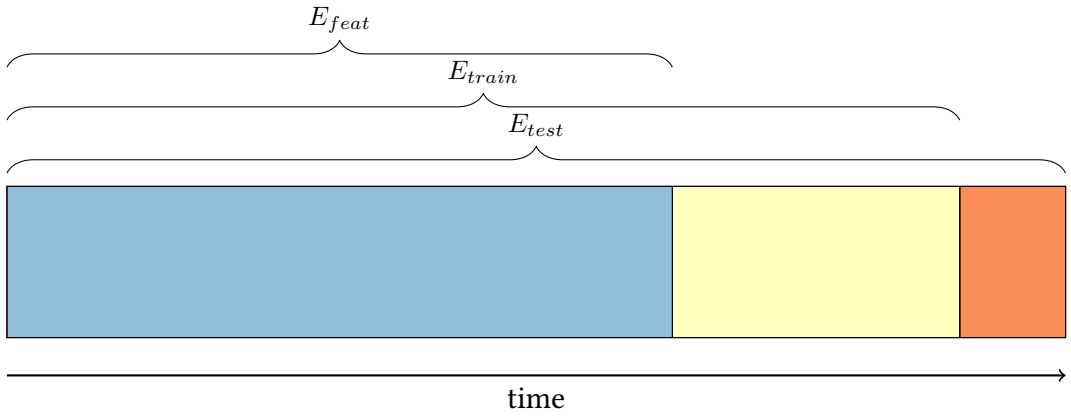

Figure 6.1.: Partitioning of the initial network into subgraphs.

The advantage of using percentage based partitioning in favor of absolute numbers becomes most relevant for graphs that largely differ in size. In such a case, the absolute number of edges that should be used for training might even already exceed the total number of edges in a graph. On the other hand, there might be graphs that would need a larger number of edges for this task. The actual percentages for partitioning depend on multiple factors. The amount of edges used for testing should be as small as possible and as large as necessary. This means the evaluation shouldn't be influenced through a too small number of available testing node-pairs but there should also be as much information as possible available for training. The same holds true for the feature and training graph. The percentage used to split training and feature graph is also a compromise between feature quality and the number of node-pairs in the training set. If the percentage is too large, there will be very few positive instances in the training set for small graphs and this would impact performance. On the other hand, if the percentage is too small, the feature graph will be very fragmented and the features generated for the training set will not be meaningful. This will also lead to deficiencies during training and the performance of the supervised methods will suffer as a result. The experiment 6.3.2 showed that the highest evaluation scores are generated with a large feature graph with respect to the training graph. To provide enough node-pairs for small graphs while maintaining high evaluation scores, 70% of the links in the training graph will be assigned to the feature graph.

The next step is to select a training set as well as a testing set based on the subgraphs. There will be two sets of node-pairs, the testing set $S_{test} \subseteq V \times V$ and the training set $S_{train} \subseteq V \times V$. Both sets should consist of both positive (node-pairs that will form a link) and negative instances (node-pairs that will not form a link). For unsupervised methods, the easiest way would be to simply select all the missing links from the training graph as the testing set. This way, all the edges in $E_{test} \setminus E_{train}$ will be positive instances and the remaining node-pairs will be negative instances. The problem that arises is that the number of missing links $\frac{|V|\times(|V|-1)}{2} - |E_{train}|$ will be too big, especially for sparse networks. Consequently, there needs to be a reduction in the problem space. As most of the new links will be created at hop-distance two, predicting links at this distance is the most important problem a predictor should be good at. Just looking at node-pairs with hop-distance two is a good start to limit the number of node-pairs that should be considered for prediction. This reduction will still lead to millions of node-pairs to consider in larger networks. This is why the total number of node-pairs in the problem space needs to be limited as well. The size of the testing set will thus be limited to 100,000 node-pairs. This ensures that the computations are feasible and that there are enough positive instances to receive smooth threshold curves. Even though a modified data distribution does not represent the same challenge as the real-world distribution, for the purpose of comparability to other research [39, 18] done in the link prediction domain, the testing set will be balanced. This is achieved by undersampling the negative instances in the 2-hop distance subproblem. The positive instances will also be undersampled if there are more than 50,000. Experiment 6.3.3 shows the consequences for evaluation if the imbalance ratio of the testing set is changed.

The training set can be constructed in a similar way by considering all the two-hop missing links in the feature graph. The positive instances will be obtained from the withheld links $E_{train} \setminus E_{feat}$ in the training graph. To reduce the number of node-pairs in the training set, the same procedure used for the testing set will be applied. Experiment 6.3.1 showed that a total of 60,000 node-pairs is sufficient for training, as there is no significant performance improvement for larger training sets. The training set will also be balanced because Weiss and Provost [60] showed that a balanced distribution is "within the optimal range" if AUROC is used as the performance measure.

All experiments will be conducted on a Transtec PHI 4230 Workstation. The hardware specifications are listed in Table 6.1.

phipute1.iti.kit.edu	
Compiler	G++ 4.8.2
#Cores	2×8
#Threads	32
CPU Frequency	2.7 GHz
RAM	256 GB

Table 6.1.: Hardware specifications.

6.2. Data sets

The success of link prediction depends heavily upon the given network. For this reason, analyzing networks from different domains and with different properties becomes important if we want to obtain a meaningful estimation about the performance of a link predictor. This is why some of the selected networks have timestamps associated with their edges while others do not. Time-invariant data poses new problems to a predictor because during the split of the data into training and testing sets, the inherent mechanism that leads to new links gets distorted in an unpredictable way. Unfortunately, there are often no time-variant sources available and for this reason the evaluation on time-invariant networks has high practical relevance. The included networks also strongly vary in their sizes. This allows us to look at scenarios where there is very little information available versus cases in which there is abundant information. Another important consideration was the comparability with other works on link prediction. This is why all the chosen networks have been studied by other researchers before and are mostly well known in the link prediction community.

All the networks are undirected or will be regarded as undirected. Edge-weights are not taken into consideration for link prediction. In case there are multi-edges, only the first edge created will be used. Self-loops are also excluded.

The networks DBLP, Facebook, and Hep-Th have been obtained through the Koblenz Network Collection (KONECT) [35].

6.2.1. Time-invariant Networks

The following networks have no timestamps associated with their edges.

Cond-Mat This is a co-authorship network about condensed matter (Cond-Mat) physics collected by Newman [49]. The basis for this dataset are preprints posted to the arXiv E-Print Archive between Jan 1, 1995, and March 31, 2005. The network is bipartite and the nodes represent authors or papers. An edge indicates that a specific

author co-authored a paper. There are no timestamps associated with the edges and the network has been projected onto a network with just one generic node-type.

Grid The Grid network has been collected by Watts and Strogatz [16], and is a network showing the high-voltage power grid in the Western United States of America. The nodes represent generators, substations and transformers, whereas the edges are high-voltage transmission lines.

Jazz This network was collected by Gleiser and Danon [22] in order to analyze the topology and community structure of the collaboration network of jazz musicians. A node represents a single jazz musician and an edge between two musicians indicates that they have played together in a band.

	Network		
Property	Cond-Mat	Grid	Jazz
Nodes	40,421	4,941	198
Edges	175,692	6,594	2,742
Components	36,458/1,798	4,941/1	198/1
Density	0.000215	0.000540	0.140594
Clustering Coefficient	0.7221	0.1045	0.6374
Degrees	0/278	1/19	1/100
Assortativity	0.1863	0.0035	0.0202

Table 6.2.: Properties of time-invariant networks. The first number of the components is the number of nodes in the largest component and the second number indicates the total number of components in the network. The first degree is the minimal degree of a node in the network and the second value indicates the largest degree of a node in the network.

6.2.2. Time-variant Networks

Time-variant networks contain timestamps associated with the time of creation in respect to their edges. This makes it possible to divide a given network on the basis of the timestamps. This way the training set could consist of the first months of the given network and the testing set can be selected by looking at the edges that were added after the end of the training period. This way any inherent mechanism that leads to new edges will not be influenced through the splitting.

DBLP DBLP is a collaboration network of authors of scientific papers from the DBLP computer science bibliography[1] which was released by Ley [37]. DBLP provides researchers with meta-data and links to electronic editions of publications. Nodes represent authors of papers and an edge between two authors indicates a common publication. The timestamp of an edge indicates the date of publication. In case two authors have written multiple publications together, only the first publication is considered.

Facebook This network is an excerpt of all the user-to-user links in the Facebook New Orleans networks and was made available through Viswanath et al. [59] in a paper for the Workshop on Social Networks (WOSN) in 2009. A node represents a Facebook user and a link indicates a friendship on Facebook between two persons, where the timestamp refers to the creation of the friendship.

Hep-Th Hep-Th is a citation graph that was provided as part of the KDD Cup in 2003 and made available through Leskovec et al. [36]. The network shows the collaboration between authors of scientific papers from the High Energy Physics - Theory (Hep-Th) section in the arXiv E-Print Archive. The network was collected in the period from January 1993 to April 2003. This is essentially the complete history of the hep-th section until April 2003, as the arXiv started just a few months before. A node represents an author and an edge between two authors symbolizes that the authors have both co-authored the same paper. If a paper has been authored by k scientists, these will form a completely connected subgraph with k nodes. The timestamp of an edge indicates the publication date of the co-authored paper.

	Network		
Property	DBLP	Facebook	Hep-Th
Nodes	1,314,050	63,731	22,908
Edges	5,362,414	817,035	2,444,798
Components	1,167,956/49,111	63,392/144	22,721/74
Density	0.000006	0.000402	0.009318
Clustering Coefficient	0.734778	0.250456	0.614132
Degrees	1/1,545	1/1,098	1/8,718
Assortativity	0.1028	0.1770	-0.0339

Table 6.3.: Properties of time-variant networks. The notation is identical to Table 6.2.

[1] http://dblp.uni-trier.de

6.3. Results

In this section the results of the conducted experiments will be presented. A discussion and interpretation of the obtained results can be found in chapter 7. Any values with a bold font in the following tables indicate the strongest performance for the respective column.

6.3.1. Training Set Size

In the first experiment, the performance of the decision tree bagging classifier will be tested for different training set sizes on the Cond-Mat network. This allows us to determine a training set size that is large enough to provide enough information to the classifier in order to achieve its highest AUROC scores. The testing was done based on a balanced data set sampled from two-hop missing links, where the testing graph consists of 10% of the initial edges and 70% of the training graph are used as the feature graph. The number of node-pairs in the testing set was limited to 100,000 edges. Figure 6.2 and Table 6.4 show the performances.

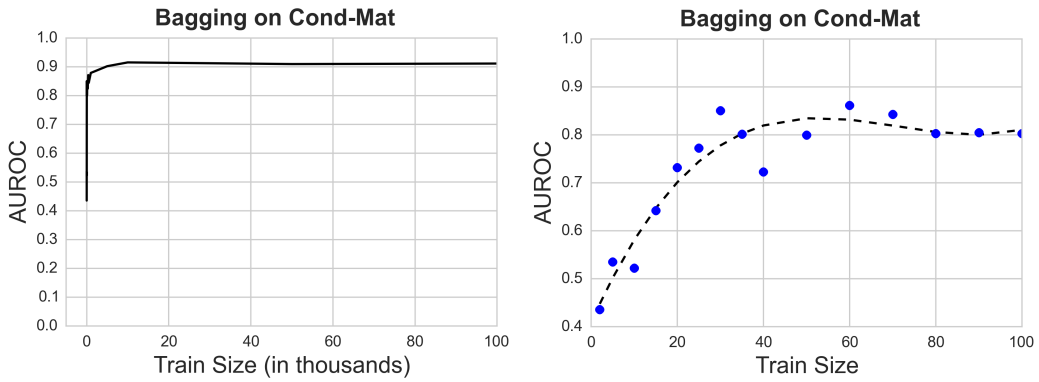

Figure 6.2.: Plots showing performance of bagging for different training set sizes. The right plot shows an excerpt of the left plot.

	Training set Size				
	10^1	10^2	10^3	10^4	10^5
AUROC	0.5222	0.8029	0.8788	0.9153	0.9113

Table 6.4.: Performance of bagging for different training set sizes.

6.3.2. Feature and Training Graph Partitioning

The goal of this experiment is to determine how much the proportions between feature and training graph influence prediction quality. This is done by assigning different percentages of the training graph to the feature graph. Table 6.5 shows the percentage of edges in the training graph that are assigned to the feature graph as well. For every percentage, the area under the ROC and PR are shown. The scores have been obtained on the Cond-Mat network.

	Bagging		Boosting		Naive Bayes	
Percentage	ROC	PR	ROC	PR	ROC	PR
10	0.8842	0.8918	0.9213	0.9294	0.8518	0.9294
20	0.9039	0.9133	0.9270	0.9342	0.8511	0.9342
30	0.9123	0.9188	0.9299	0.9368	0.8514	0.9368
40	0.9128	0.9170	0.9302	0.9364	0.8539	0.9364
50	0.9178	0.9203	0.9359	0.9394	**0.8542**	0.9394
60	0.9137	0.9158	0.9371	0.9423	0.8481	0.9423
70	0.9259	0.9292	0.9394	0.9429	0.8515	0.9429
80	0.9301	0.9340	**0.9414**	**0.9457**	0.8505	**0.9457**
90	**0.9305**	**0.9352**	0.9407	0.9451	0.8454	0.9451

Table 6.5.: ROC and PR performance of supervised methods for varying feature graph percentages.

6.3.3. Testing Set Imbalance

To reason about the obtained evaluation scores, the influence of the percentage of positive instances in the testing set should be analyzed. For this purpose, the Facebook data set has been evaluated at different testing set densities, starting with the network-inherent imbalance ratio (~0.2%) for all the two-hop missing links in the training set. Afterwards different imbalance ratios are tested until the testing set is balanced. For every density, the area under ROC and PR have been obtained for all predictors and are shown in Table 6.6 and 6.7. There are no limitations regarding the size of the testing set.

6.3.4. Predictor Evaluation

After determining the best way to train and test the predictors, the following experiments will determine the performance of the actual predictors for the data sets. For every network there will be two plots showing the ROC and PR performance for supervised meth-

	Density					
Predictor	~0.2%	1%	5%	10%	25%	50%
Adamic/Adar	0.8495	**0.8686**	**0.8609**	0.8572	0.8545	0.8579
Adjusted Rand	0.8683	0.8125	0.7725	0.7669	0.7730	0.7960
Common Neighbors	0.7393	0.7874	0.8034	0.8057	0.8032	0.7933
Jaccard	0.8733	0.8211	0.7815	0.7761	0.7822	0.8052
Preferential Attachment	0.3592	0.5277	0.5934	0.6048	0.6061	0.5881
Resource Allocation	0.8788	0.8564	0.8230	0.8119	0.8052	0.8118
Neighborhood Distance	**0.8840**	0.8093	0.7524	0.7421	0.7436	0.7649
Neighbors-Measure	0.5910	0.7060	0.7452	0.7526	0.7535	0.7478
Katz	0.7242	0.7932	0.8171	0.8211	0.8208	0.8172
Decision Tree Bagging	0.8012	0.8282	0.8312	0.8356	0.8342	0.8406
Decision Tree Boosting	0.8129	0.8453	0.8566	**0.8593**	**0.8644**	**0.8677**
Naive Bayes	0.6875	0.7318	0.7340	0.7351	0.7350	0.7404

Table 6.6.: AUROC for all predictors at varying testing set densities on Facebook.

	Density					
Predictor	~0.2%	1%	5%	10%	25%	50%
Adamic/Adar	0.0282	0.1427	0.4099	0.5484	0.7306	0.8721
Adjusted Rand	0.0456	0.0679	0.1551	0.2649	0.5264	0.7917
Common Neighbors	0.0228	0.1237	0.3757	0.5155	0.7010	0.8417
Jaccard	0.0468	0.0703	0.1598	0.2716	0.5349	0.7979
Preferential Attachment	0.0025	0.0173	0.0847	0.1625	0.3556	0.5993
Resource Allocation	0.0510	0.1109	0.2509	0.3585	0.5723	0.7917
Neighborhood Distance	0.0438	0.0595	0.1345	0.2298	0.4747	0.7534
Neighbors-Measure	0.0113	0.0689	0.2444	0.3719	0.5828	0.7750
Katz	0.0222	0.1216	0.3723	0.5137	0.7039	0.8493
Decision Tree Bagging	0.0168	0.0937	0.3314	0.4984	0.6977	0.8609
Decision Tree Boosting	0.0197	0.1295	0.4090	**0.5531**	**0.7489**	**0.8826**
Naive Bayes	**0.2815**	**0.3081**	**0.4169**	0.5110	0.6792	0.8297

Table 6.7.: AUPR for all predictors at varying testing set densities on Facebook.

ods as well as two plots for the similarity indices. There are also measurements about the runtime of the different predictors which are especially interesting since we are interested in scalable approaches to link prediction. For every method on every network there are also two graphs in the appendix showing just the performance of this single measurement with respect to ROC and PR.

Size	Network					
	Cond-Mat	Grid	Jazz	DBLP	Facebook	Hep-Th
Training set	53,050	266	1354	60,000	60,000	60,000
Testing set	27,378	180	504	100,000	100,000	100,000

Table 6.8.: Training and testing set size for all networks.

Due to the large differences in size, the number of node-pairs for training and testing set vary across networks. While all time-variant networks exhaust the limit of 60,000 node-pairs for training and 100,000 node-pairs for testing, the smaller time-invariant networks do not. Table 6.8 shows the actual number of node-pairs for training and testing set during the experiments.

6.3.4.1. Runtime

To decide whether a predictor should be used in a deployment scenario, not only prediction quality is important but also the actual runtime. Even the best method will not be used if the runtime turns out to be infeasible. For this reason, the following tables are showing the rounded runtimes for the predictors. First up, the runtimes of the unsupervised methods will be presented in Table 6.9. These include the seven local indices as well as the quasi-local and global indices, Neighbors-Measure and Katz respectively.

Predictor	Network					
	Cond-Mat	Grid	Jazz	DBLP	Facebook	Hep-Th
Adamic/Adar	0.07s	0.01s	0.00s	0.20s	0.20s	0.34s
Adjusted Rand	0.03s	0.00s	0.00s	0.26s	0.30s	0.89s
Common Neighbors	0.07s	0.00s	0.01s	0.06s	0.08s	0.12s
Jaccard	0.03s	0.00s	0.00s	0.08s	0.10s	0.22s
Preferential Attachment	0.02s	0.00s	0.00s	0.16s	0.18s	0.14s
Resource Allocation	0.06s	0.00s	0.00s	0.06s	0.09s	0.12s
Neighborhood Distance	0.02s	0.00s	0.00s	0.07s	0.08s	0.12s
Neighbors-Measure	0.07s	0.00s	0.00s	1.13s	1.77s	167.33s
Katz	19.63s	0.01s	0.01s	3731.09s	476.28s	817.59s

Table 6.9.: Runtime of the unsupervised methods on all networks.

The effective runtime of the supervised approaches can be divided into three categories: feature-generation, fitting, and prediction. The feature-generation includes the generation of features for the training set as well as for the testing set. In a real application, the

testing set would be replaced with a set of node-pairs where the user wants to know which node-pairs will form a link in the future. Consequently, only the runtime for the feature generation of these node-pairs is of interest, since the feature generation for training has to be done only once. The same holds true for fitting, which is the actual process of training a supervised classifier. This is not done repeatedly. Besides the runtime needed for feature extraction during testing, the runtime needed by the classifier to determine a class label based on the given feature-vectors is also relevant. Table 6.10 shows the runtime of the supervised methods which is divided into fitting runtime and prediction runtime. Afterwards, Table 6.11 informs about feature generation runtime by showing the runtime for training and testing set on all networks.

		Network					
Predictor	Process	Cond-Mat	Grid	Jazz	DBLP	Facebook	Hep-Th
	FIT	0.67s	0.38s	0.34s	1.25s	1.35s	1.36s
DT Bagging	PRED	0.49s	0.30s	0.31s	1.41s	1.53s	1.62s
	FIT	2.36s	0.04s	0.10s	2.66s	3.02s	3.75s
DT Boosting	PRED	0.17s	0.01s	0.01s	0.59s	0.61s	0.58s
	FIT	0.01s	0.00s	0.00s	0.01s	0.01s	0.01s
Naive Bayes	PRED	0.01s	0.00s	0.00s	0.03s	0.03s	0.02s

Table 6.10.: Runtime of supervised methods on all networks. *FIT* indicates the fitting runtime and *PRED* the runtime for actual prediction with given features.

	Network					
Features	Cond-Mat	Grid	Jazz	DBLP	Facebook	Hep-Th
Training	1.41s	0.00s	0.05s	1.25s	2.01s	2.13s
Testing	0.77s	0.00s	0.05s	3.82s	3.18s	4.10s

Table 6.11.: Runtime of feature generation for training and testing set on all networks.

6.3.4.2. Parallelization Speedup

Here, we display the results of a speedup analysis. The plots in Figure 6.3 are showing the speedups for different OpenMP scheduling algorithms with respect to the number of threads used. The speedup for every number of threads is averaged over 10 runs. The speedups show the performance for the Common Neighbors index on all two-hop missing links (2,402,571 in total) in the Cond-Mat (see 6.2) network. All the tested scheduling algorithms have been used with the default parameters of OpenMP. This means the chunk

size was set to 2,402,571/number_of_threads for the static schedule, 1 for the dynamic schedule, and 1 for the minimum chunk size for the guided schedule.

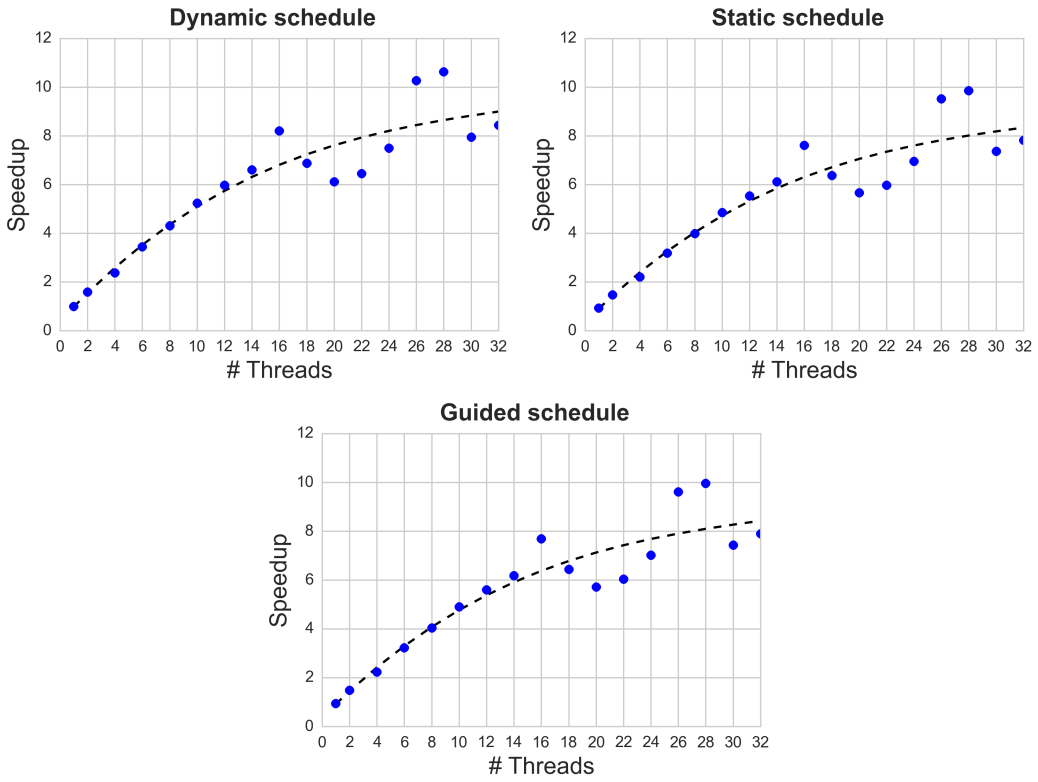

Figure 6.3.: Speedups for three OpenMP scheduling algorithms.

6.3.4.3. ROC and PR Performance

The ROC and PR performance of all predictors for all networks will be summed up in Table 6.12 and Table 6.13.

To get a more detailed impression about the performance of the predictors at different false positive rates (in case of ROC) or recall rates (in case of PR), Figure 6.4 to Figure 6.15 show the performances of the individual predictors for all networks.

	Network					
Predictor	Cond-Mat	Grid	Jazz	DBLP	Facebook	Hep-Th
Adamic/Adar	0.9054	0.3875	0.8893	0.6840	0.8584	0.9133
Adjusted Rand	0.8644	0.4914	0.8168	0.4803	0.8125	0.9736
Common Neighbors	0.7445	0.2593	0.8650	0.7230	0.7767	0.9007
Jaccard	0.8659	0.4895	0.9038	0.4816	0.8213	0.9757
Preferential Attachment	0.3503	0.4494	0.5909	0.6682	0.5663	0.5102
Resource Allocation	0.8990	0.3716	0.9180	0.6057	0.8164	0.9229
Neighborhood Distance	0.8342	0.5213	0.9217	0.3982	0.6899	0.9798
Neighbors-Measure	0.5801	0.4778	0.7561	0.7491	0.7398	0.7101
Katz	0.7408	0.5894	0.8656	0.7689	0.8135	0.8895
Decision Tree Bagging	0.9129	0.5865	0.9031	0.7896	0.8554	0.9740
Decision Tree Boosting	**0.9377**	**0.6321**	**0.9395**	**0.8083**	**0.8721**	**0.9815**
Naive Bayes	0.8479	0.5278	0.8214	0.6861	0.7455	0.9078

Table 6.12.: Area under ROC curve for all predictors and networks.

	Network					
Predictor	Cond-Mat	Grid	Jazz	DBLP	Facebook	Hep-Th
Adamic/Adar	0.9240	0.4246	0.9065	0.7189	0.8732	0.8815
Adjusted Rand	0.8697	0.4987	0.8488	0.5347	0.8141	0.9760
Common Neighbors	0.8254	0.4161	0.8893	0.7773	0.8342	0.8680
Jaccard	0.8704	0.4978	0.9212	0.5351	0.8198	0.9759
Preferential Attachment	0.4271	0.4529	0.6042	0.7157	0.5837	0.4866
Resource Allocation	0.9145	0.4128	0.9298	0.5897	0.8006	0.9011
Neighborhood Distance	0.8255	0.5133	0.9260	0.4870	0.6591	0.9762
Neighbors-Measure	0.6582	0.5779	0.7955	0.7803	0.7702	0.6456
Katz	0.8229	0.6028	0.8900	0.8037	0.8472	0.8573
Decision Tree Bagging	0.9212	0.6134	0.9033	0.8146	0.8716	0.9744
Decision Tree Boosting	**0.9449**	**0.6370**	**0.9456**	**0.8331**	**0.8868**	**0.9821**
Naive Bayes	0.8838	0.6329	0.8598	0.7848	0.8369	0.9484

Table 6.13.: Area under PR curve for all predictors and networks.

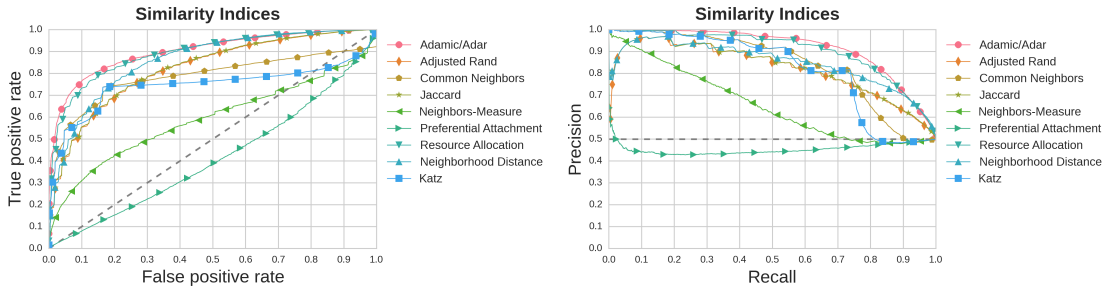

Figure 6.4.: AUROC and AUPR for all unsupervised indices on Cond-Mat.

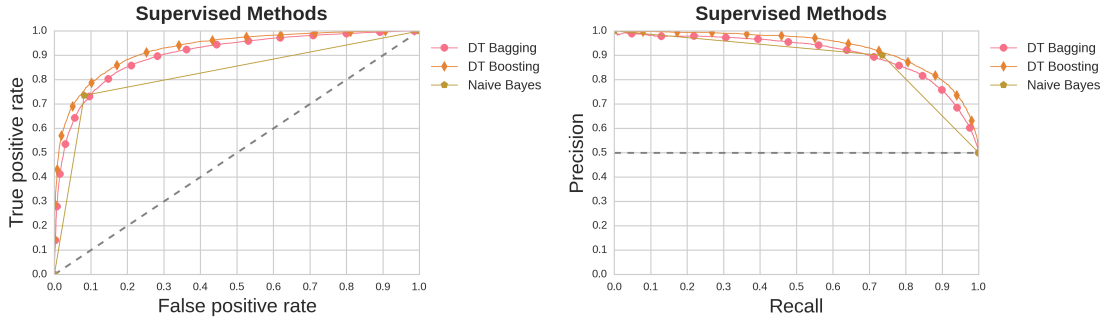

Figure 6.5.: AUROC and AUPR for all supervised methods on Cond-Mat.

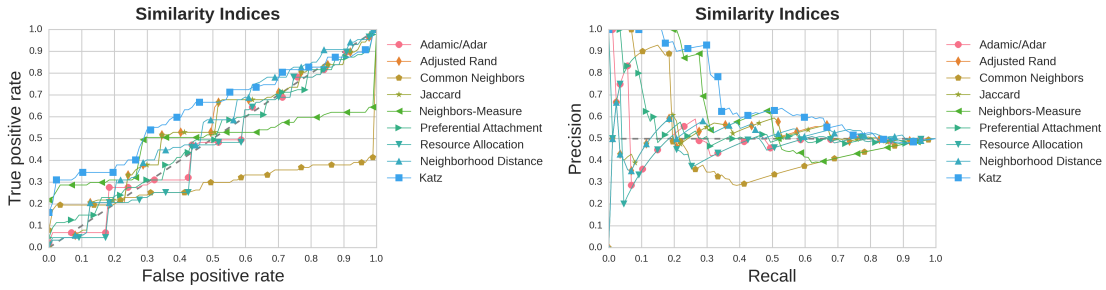

Figure 6.6.: AUROC and AUPR for all unsupervised indices on Grid.

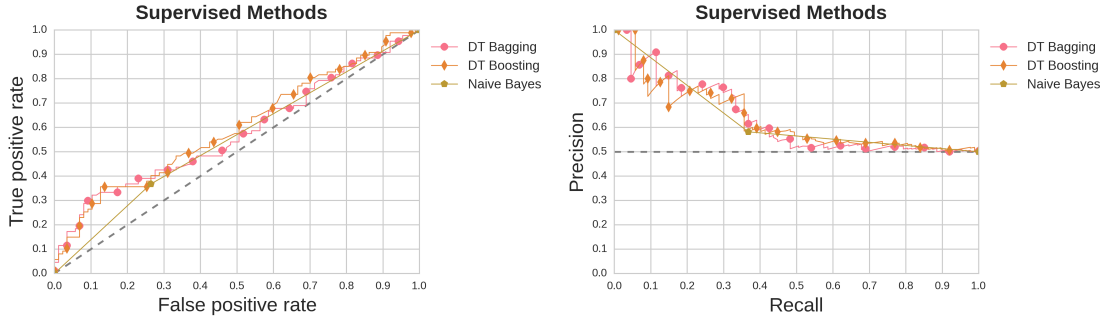

Figure 6.7.: AUROC and AUPR for all supervised methods on Grid.

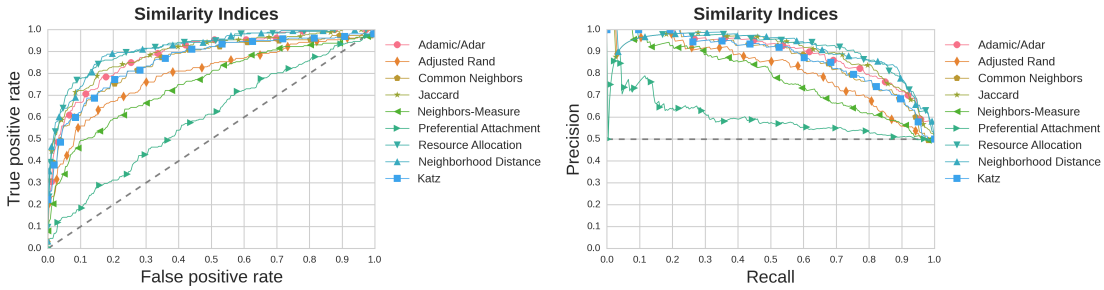

Figure 6.8.: AUROC and AUPR for all similarity indices on Jazz.

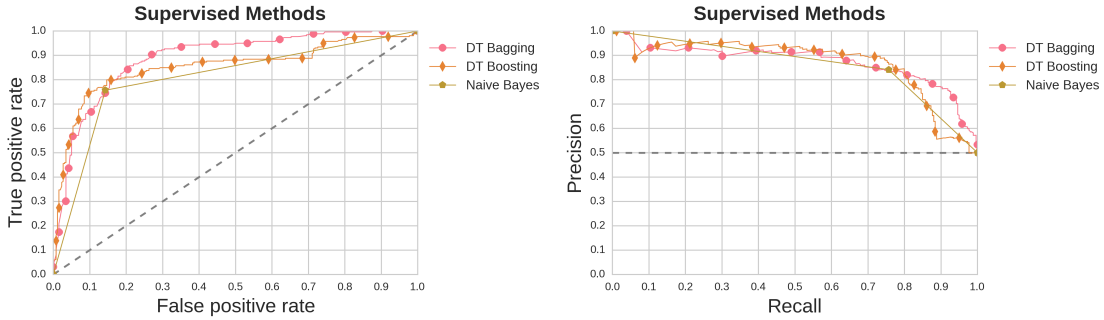

Figure 6.9.: AUROC and AUPR for all supervised methods on Jazz.

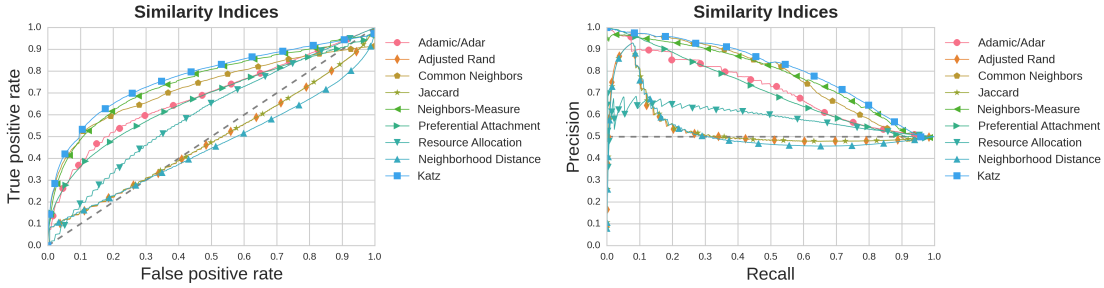

Figure 6.10.: AUROC and AUPR for all unsupervised methods on DBLP.

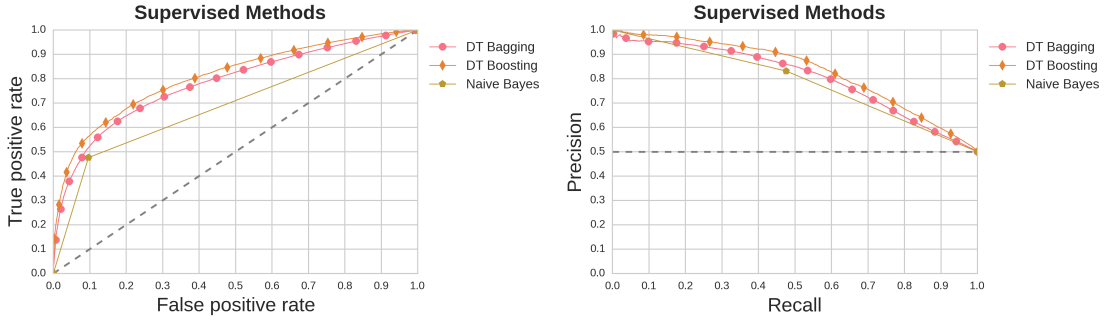

Figure 6.11.: AUROC and AUPR for all supervised methods on DBLP.

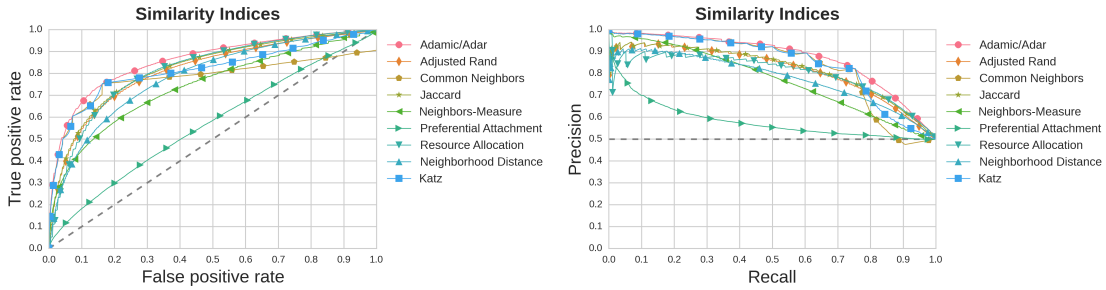

Figure 6.12.: AUROC and AUPR for all unsupervised methods on Facebook.

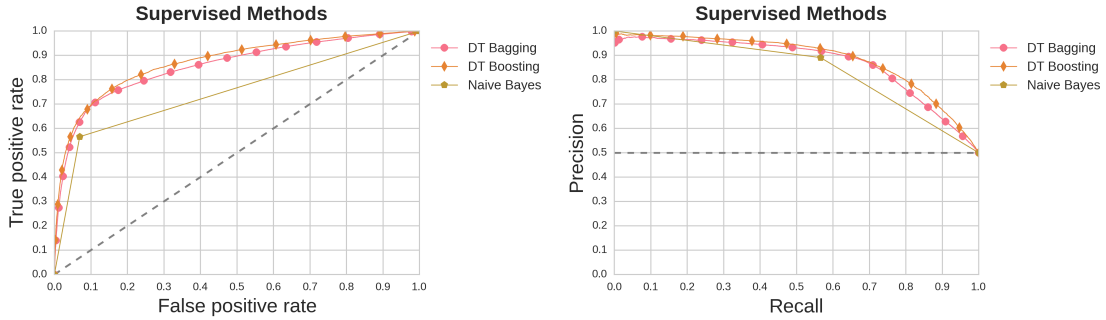

Figure 6.13.: AUROC and AUPR for all supervised methods on Facebook.

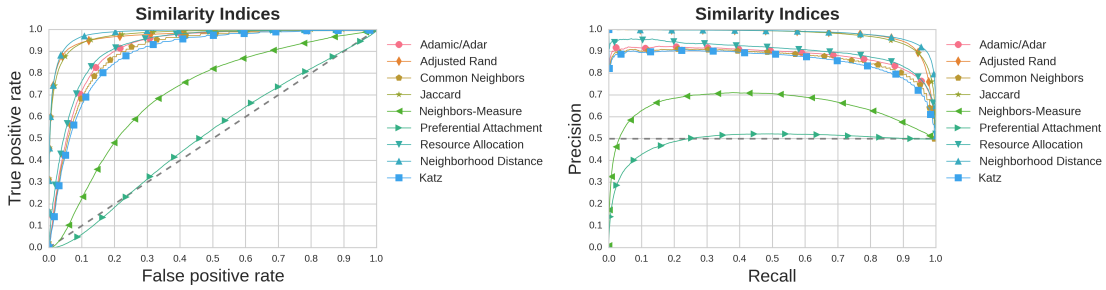

Figure 6.14.: AUROC and AUPR for all unsupervised methods on Hep-Th.

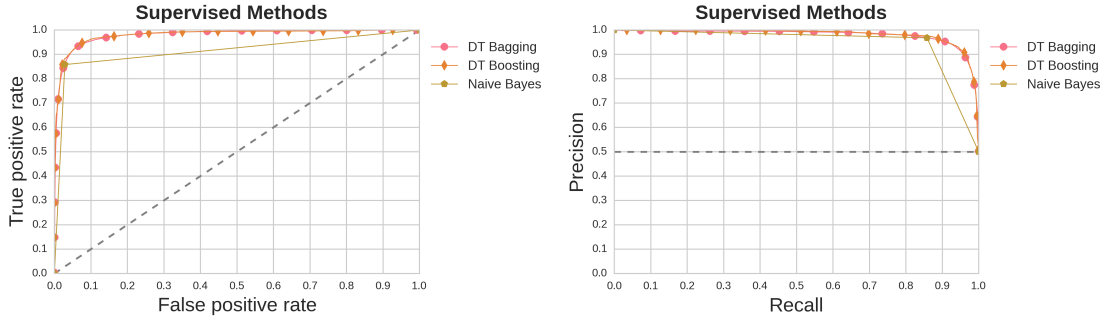

Figure 6.15.: AUROC and AUPR for all supervised methods on Hep-Th.

7. Discussion

The goal of this chapter is to provide an overview over the obtained results and, more importantly, try to make assumptions about the mechanisms that lead to these results.

7.1. Predictor Performances

In the previous chapter, twelve link predictors have been evaluated on six different networks with two evaluation metrics. Here, we want to discuss the obtained evaluations.

In general, the supervised methods outperformed the unsupervised similarity indices across the board. This is expected and agrees with the experiments conducted by Lichtenwalter et al. [39], where the authors also compared unsupervised similarity indices to supervised methods by using the similarity indices as features. Supervised algorithms are able to detect interdependencies between multiple features and this can lead to performance improvements. The difference between supervised and unsupervised performances varies for every network from an AUROC improvement of just 0.0017 for Hep-Th to an improvement of up to 0.0427 for Grid, where the best score is 0.9815 for Hep-Th and 0.6321 for Grid. Hep-Th is thus the easiest task and Grid the hardest task for link prediction in the examined networks. This indicates that supervised methods are especially useful if the structure of the networks exacerbate link prediction.

If prediction quality is more important than runtime, enriching the features for supervised learning with more diverse features could substantially increase performance. Currently, only local features are used as we are interested in scalable link prediction and local indices have the lowest computational complexity. A disadvantage of this approach is the similarity of the features. For example, Common Neighbors, Jaccard, and Neighborhood Distance all share the same numerator which can lead to similar predictions. Using global or quasi-local indices like the length of the shortest-path between two nodes provides a different view on to the relationship between two nodes and can help the supervised classifier. The results of Fire et al. [18] support this assumption. The authors were able to obtain an AUROC score of 0.923 for the Facebook network (identical to the network we used) by using the Random Forest ensemble method with 34 features.

It should be noted that the experimental setup differs from the one used in this thesis because their testing set is randomly sampled from all unconnected node-pairs whereas we only investigated unconnected node-pairs with two-hop distance.

Link prediction was most successful on Hep-Th, Jazz, and Cond-Mat while still performing well on Facebook and DBLP. Only the performance on Grid was insufficient. The most reasonable explanation for these results is the exceptional structure of Grid. Even though there are no major differences in the determined properties of Grid in comparison to the other networks (see 6.2 and 6.3), Grid is a strongly localized network that lacks short loops as these would reduce the efficiency of the grid by introducing unneeded power supply lines. This characteristic is in stark contrast to the other investigated networks which are based on social behaviour and thus consist of more short loops. The bad performance of local indices as well as the comparably strong performance of Katz support this hypothesis. In fact, the performance of the Common Neighbors index is so bad that inversing its predictions (e.g. mirror its generated scores at the x-axis) would yield the by far strongest performance on Grid with respect to AUROC. The Katz index is able to consider longer paths between two nodes which should be stronger indicators for future links. Another explanation would be the very low number of node-pairs in the training and testing set. This could promote instabilities in prediction and the number of node-pairs might not be sufficient to capture the mechanisms in the network that lead to new links.

A counterexample to the last explanation is the overall strong performance on Jazz. There are only marginally more node-pairs in the training and testing set in comparison to Grid (difference of 1088 node-pairs for training and 324 for testing, see 6.8) while the difference in performance is enormous with the highest score of 0.9395 for Jazz and 0.6321 for Grid. This suggests that a small number of node-pairs is sufficient for unsupervised as well as supervised methods. Experiment 6.3.1, used to determine a large enough training set size for supervised methods, supports this assumption. Here, the AUROC improvement from a training set size of 1,000 node-pairs to 100,000 node-pairs amounts to only 4.6%.

The performance of the newly introduced Neighborhood Distance index turns out to be the strongest of all unsupervised methods on Jazz and Hep-Th while performing poorly on DBLP. In a direct comparison regarding the correlation between AUROC performance and network properties, the density of the network seems to be the best indicator for Neighborhood Distance performance. Jazz is extremely dense with 14% of all possible links being existent while Hep-Th is still at least 17 times denser (0.9% of all possible links being existent) than all other networks except Jazz. The bad performance on DBLP is

reflected in the density as well, DBLP being the sparsest network with a density of just 0.0006%.

The quasi-local and global indices did not perform as well as expected. There were stronger unsupervised methods on all networks except for Grid and DBLP with respect to AUROC and AUPR. This suggests that if we select two nodes from one of the networks where the local indices perform better than the quasi-local and global indices, the paths with length two that connect the nodes would have a stronger influence on the probability that the nodes will connect in the future than paths with a length greater than two.

Another explanation could be the reduction of the original problem space to a limited set of node-pairs with two-hop distance. If all node-pairs with a non-existent link are considered for testing, most of the considered node-pairs would not have any common neighbors. In this scenario, many local indices would generate a score of 0 for all the node-pairs that have no common neighbors, whereas the Katz index would also consider paths with length greater than two which would lead to more "fine-grained", non-zero scores because of the additional information. The effect that many node-pairs get assigned the same score is also called the "degeneracy of states" [69], referring to the degeneracy of energy levels. The better distinguishability between scores is expected to increase the performance of the Katz index which might be of interest in deployment scenarios where the whole problem space is of interest. The findings of Zhou et al. [69] are backing this assumption by showing that the quasi-local *Local Path* index outperformed the local indices on all but one network, where the Local Path index is a variant of the Katz index that considers the number of paths with length two or three between two nodes. Their experiments were conducted on the complete problem space.

7.2. Class Imbalance

Class imbalance is a major problem in link prediction. Sparse networks usually consist of only a tiny fraction of the $\frac{|V|\times(|V|-1)}{2}$ possible links. Consequently, the number of node-pairs to consider for link prediction would be infeasible with respect to runtime and memory constraints. At least for testing purposes, there is a need for undersampling the data to reduce the number of node-pairs to consider for prediction. This impacts the meaningfulness of results obtained through undersampling.

By varying the density of the testing set from ~0.2% to 50%, experiment 6.3.3 suggests that the ROC metric is only slightly influenced by the density, whereas the precision-recall metric is strongly influenced by the testing set density. As mentioned in 4.4.1.2, the PR performance can partly be explained by the increased baseline difficulty. Another reason

for this behaviour is that precision is a probability that is conditioned on the estimated class label, which means it indicates the probability that a node-pair is actually connected after estimating that there is a connection. In theory, the ROC should not be affected by class imbalance as the true positive rate is a probability that is conditioned on the actual class label. In practice, the variations in the AUROC score can be ascribed to instabilities in the ranking of the scores [64] and the random selection of node-pairs for the testing set. It is noticeable that some predictors can achieve substantially higher PR scores at different imbalance ratios in the testing set than others. The naive Bayes classifier performed exceptionally well for ~0.2% to 5% while decision tree boosting performed strongest on the remaining densities. For the unsupervised indices, the Adamic/Adar, Katz and Common Neighbors indices showed a good performance at imbalance levels between 1% and 50% while the Resource Allocation index is the strongest unsupervised method at the original imbalance ratio of 0.2% for all the two-hop missing links. There are a multitude of factors that can influence the PR performance of predictors at different imbalance ratios. Some of them might be network size as well as individual network and predictor properties.

7.3. Scalability

The importance of scalability is growing as companies like Google and Facebook acquire enormous amounts of data and there is an increased need for algorithms that can cope with these data collections. To handle this problem with the set of currently available hardware, making use of massive parallelization in conjunction with scalable algorithms that have low computational complexity is an established strategy.

Experiment 6.3.4.2 analyses the speedup of the implemented Common Neighbors index on all two-hop missing links in Cond-Mat for three different scheduling algorithms. It shows that the speedup is not strongly influenced by the scheduling algorithm, even though dynamic scheduling tends to perform slightly better in the long run. Starting at 16 cores, the speedup becomes increasingly unstable for all scheduling algorithms. The highest speedup is 11.39 for the guided scheduling at 256 cores. There are a number of effects that can produce the instabilities that start at 16 cores. These include additional CPU cycles used to manage parallelism, memory delays, and influences through caching. Overall the speedup does not meet our expectations, even though link prediction is an embarrassingly parallel problem which would suggest a greater speedup since there is no part of the problem that can not be parallelized. An explanation could be that the implementation does not utilize caching to its fullest potential. Optimizing this could turn out to be a hard problem because the local similarity indices access the graph structure

based on node-neighborhoods which means there is a high likelihood that a wide range of nodes and edges will be accessed for the score calculation of every node-pair. This leads to low spatial locality which in return leads to low cache efficiency.

There are also significant differences between the runtime of the local indices and the runtime of the Neighbors-Measure and Katz index. The calculation of scores through the Katz index took more than an hour on Cond-Mat while all local indices finished in less than a second. The Neighbors-Measure performed quite well for all networks except for the denser Hep-Th. This is understandable because a higher density implies a higher average degree which worsens performance for measures that look at paths with varying lengths.

8. Conclusion

The aim of this thesis has been to investigate the scalability and performance of supervised and unsupervised predictors on six different networks while also analyzing runtime and speedup for parallelization as well as the influences of training and testing set on overall performance.

The evaluation of all link predictors showed that supervised classifiers performed stronger than unsupervised indices on all networks. The increase in AUROC performance for the best supervised method in comparison to the best unsupervised method on every network ranged from 0.2% for Hep-Th to an increase of 7.2% for Grid. Quasi-local and global indices performed worse than the best local indices on all networks except Grid. This can be partly attributed to the problem space reduction to all the node-pairs with 2-hop distance.

We were also able to show that the structure and properties of a network have substantial influence on the quality of the predictions. The performance on Grid was only slightly better than random guessing, whereas the predictors were able to generate almost perfect predictions on Hep-Th.

The newly introduced Neighborhood Distance index turns out to be a viable alternative to established indices, outperforming all unsupervised methods on Jazz and Hep-Th. Even though the index does not perform consistently well on all networks, it is powerful for networks that have a comparably high density.

The parallelization efforts achieved a maximal speedup of more than ten with dynamic scheduling at 28 threads. As all the scores for a network can be calculated in parallel, a greater speedup seems very likely if spatial locality is exploited to its fullest. Apart from that, the implementation could be enhanced by adding disk-based link prediction. Even though this would drastically reduce performance, the user would be able to run predictors on a larger number of node-pairs without facing memory constraints.

Class imbalance also plays a vital role in the evaluation of predictors. Even though AUROC is only very slightly influenced by class imbalance, the Precision-Recall metric heavily responds to changes in class imbalance. A high accuracy for a highly imbalanced testing set is hard to achieve.

Finally, as the results show that the performance of the predictors is highly dependent upon the selected network, researching the influences of network properties on predictor performances could help to obtain a better understanding of this interaction. This could help to choose, based on network properties, predictors that will perform well for a network without testing the predictor beforehand on the network. One step further, there could be an algorithm that generates an individual unsupervised index for a specific network based on the networks properties and structure.

Bibliography

[1] E. Acar, D.M. Dunlavy, and T.G. Kolda. "Link Prediction on Evolving Data Using Matrix and Tensor Factorizations". In: *2009 IEEE International Conference on Data Mining Workshops*. ICDMW '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 262–269. ISBN: 978-1-4244-5384-9. DOI: 10.1109/ICDMW.2009.54. URL: http://dx.doi.org/10.1109/ICDMW.2009.54.

[2] Lada a. Adamic and Eytan Adar. "Friends and neighbors on the Web". In: *Social Networks* 25.3 (2003), pp. 211–230. ISSN: 03788733. DOI: 10.1016/S0378-8733(03)00009-1.

[3] Luis a Nunes Amaral. "A truer measure of our ignorance." In: *Proceedings of the National Academy of Sciences of the United States of America* 105.19 (2008), pp. 6795–6796. ISSN: 0027-8424. DOI: 10.1073/pnas.0802459105.

[4] L. Backstrom and J. Leskovec. "Supervised Random Walks: Predicting and Recommending Links in Social Networks". In: (2010). DOI: 10.1145/1935826.1935914. arXiv: 1011.4071. URL: http://arxiv.org/abs/1011.4071.

[5] R. Balakrishnan and K. Ranganathan. *A Textbook of Graph Theory*. 2012, p. 305. ISBN: 1461445299. DOI: 10.1007/978-1-4614-4529-6.

[6] Al Barabási, Z Dezso, and E Ravasz. "Scale-free and hierarchical structures in complex networks". In: *AIP Conference …* 661.1 (2003), pp. 1–16. ISSN: 0094243X. DOI: 10.1063/1.1571285. URL: http://link.aip.org/link/?APC/661/1/1%5C&Agg=doi$%5Cbackslash$nhttp://link.aip.org/link/?APCPCS/661/1/1.

[7] Albert-László Barabási and Réka Albert. "Emergence of Scaling in Random Networks". In: *Science* 286.October (1999), pp. 509–512. DOI: 10.1126/science.286.5439.509. URL: http://www.sciencemag.org/content/286/5439/509.abstract.

[8] Albert-László Barabási et al. "Evolution of the social network of scientific collaboration". In: *Physica A* 311 (2002), pp. 590–614. ISSN: 03784371. DOI: 10.1016/S0378-4371(02)00736-7. arXiv: 0104162 [cond-mat]. URL: http://arxiv.org/abs/cond-mat/0104162.

[9] Leo Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. ISSN: 0885-6125. DOI: 10.1007/BF00058655.

[10] L. Breiman et al. *Classification and Regression Trees*. Taylor & Francis, 1984. ISBN: 9780412048418. URL: http://books.google.de/books?id=JwQx-WOmSyQC.

[11] Jie Chen and Ilya Safro. "Algebraic Distance on Graphs". In: *SIAM Journal on Scientific Computing* 33.6 (2011), pp. 3468–3490. ISSN: 1064-8275. DOI: 10.1137/090775087.

[12] Fan Chung and Wenbo Zhao. "PageRank and random walks on graphs". In: *Bolyai Society Mathematical Studies* 20 (2010), pp. 43–62. ISSN: 12174696. DOI: 10.1007/978-3-642-13580-4_3.

[13] Aaron Clauset, Christopher Moore, and M Newman. "Hierarchical Structure and the Prediction of Missing Links in Networks". In: *Nature* 453 (2008), pp. 98–101. ISSN: <null>. DOI: 10.1038/nature06830. arXiv: arXiv:0811.0484v1.

[14] Darcy Davis, Ryan Lichtenwalter, and Nitesh V. Chawla. "Supervised methods for multi-relational link prediction". In: *Social Network Analysis and Mining* (2012), pp. 127–141. ISSN: 1869-5450. DOI: 10.1007/s13278-012-0068-6.

[15] Åse Dragland. *Big Data – for better or worse*. 2013. URL: http://www.sintef.no/home/corporate-news/big-data--for-better-or-worse/ (visited on 05/03/2015).

[16] Duncan J. Watts and Steven H. Strogatz. "Collective dynamics of small-world networks". In: *Nature* 393.June (1998), pp. 440–442.

[17] Matthias Eck et al. "Extracting Translation Pairs from Social Network Content". In: *Proceedings of the 11th International Workshop on Spoken Language Translation* (2014), pp. 200–205.

[18] Michael Fire et al. "Computationally efficient link prediction in a variety of social networks". In: *ACM Transactions on Intelligent Systems and Technology* 5.1 (2013), pp. 1–25. ISSN: 21576904. DOI: 10.1145/2542182.2542192. URL: http://dl.acm.org/citation.cfm?doid=2542182.2542192.

[19] Santo Fortunato. "Community detection in graphs". In: *Physics Reports* 486.3-5 (2010), pp. 75–174. ISSN: 03701573. DOI: 10.1016/j.physrep.2009.11.002. arXiv: 0906.0612.

[20] Y Freund and Re Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139. URL: http://link.springer.com/chapter/10.1007/3-540-59119-2%5C_166.

[21] Dario Garcia-Gasulla and Ulises Cortès. "Link Prediction in Very Large Directed Graphs: Exploiting Hierarchical Properties in Parallel". In: *Proceedings of the 3rd Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data co-located with 11th Extended Semantic Web Conference {(ESWC} 2014), Crete, Greece, May 25, 2014.* Ed. by Johanna Völker et al. CEUR-WS.org, 2014. URL: http://ceur-ws.org/Vol-1243/paper5.pdf.

[22] Pablo Gleiser and Leon Danon. "Community Structure in Jazz". In: *Advances in Complex Systems* 6.4 (2003), pp. 565–573. ISSN: 0219-5259. DOI: 10.1142/S0219525903001067. arXiv: 0307434 [cond-mat]. URL: http://arxiv.org/abs/cond-mat/0307434.

[23] J. Hanley and B. McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." In: *Radiology* 143.1 (1982), pp. 29–36. ISSN: 0033-8419. DOI: 10.1148/radiology.143.1.7063747. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve%5C&db=PubMed%5C&dopt=Citation%5C&list%5C_uids=7063747.

[24] Mohammad Al Hasan and Mohammed J. Zaki. "A Survey of Link Prediction in Social Networks". In: *Social Network Data Analytics*. Ed. by Charu C Aggarwal. Springer US, 2011, pp. 243–275. ISBN: 1441984615. DOI: 10.1007/978-1-4419-8462-3_9. URL: http://dx.doi.org/10.1007/978-1-4419-8462-3%5C_9.

[25] Mohammad Al Hasan et al. "Link prediction using supervised learning". In: *SDM'06: Workshop on Link ...* 2006. URL: http://www.cs.rpi.edu/~zaki/PaperDir/LINK06.pdf.

[26] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. 2nd. Springer, 2009. ISBN: 978-0-387-84858-7. DOI: 10.1007/b94608. URL: http://www.springerlink.com/index/D7X7KX6772HQ2135.pdf.

[27] Michaela Hoffman, Douglas Steinley, and Michael J. Brusco. "A note on using the adjusted Rand index for link prediction in networks". In: *Social Networks* 42 (2015), pp. 72–79. ISSN: 03788733. DOI: 10.1016/j.socnet.2015.03.002. URL: http://linkinghub.elsevier.com/retrieve/pii/S0378873315000210.

[28] P W Holland and S Leinhardt. "Transitivity in Structural Models of Small Groups". In: *Small Group Research* 2.2 (1971), pp. 107–124. DOI: 10.1177/104649647100200201.

[29] Lawrence Hubert and Phipps Arabie. "Comparing partitions". In: *Journal of Classification* 2.1 (1985), pp. 193–218. ISSN: 01764268. DOI: 10.1007/BF01908075.

[30] Leo Katz. "A new status index derived from sociometric analysis". In: *Psychometrika* 18.1 (1953), pp. 39–43. ISSN: 00333123. DOI: 10.1007/BF02289026.

[31] Ron Kohavi and Foster Provost. "Glossary of Terms". In: *Machine Learning* 30.2-3 (1998), pp. 271–274. ISSN: 0885-6125. URL: http://dl.acm.org/citation.cfm?id=288808.288815.

[32] Y. Koren, R. Bell, and C. Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (2009), pp. 42–49. ISSN: 0018-9162. DOI: 10.1109/MC.2009.263. arXiv: ISSN0018-9162.

[33] Gueorgi Kossinets. "Effects of missing data in social networks". In: *Social Networks* 28 (2003), pp. 247–268. arXiv: 0306335v2 [cond-mat]. URL: http://arxiv.org/abs/cond-mat/0306335v2.

[34] Valdis E Krebs. "Mapping Networks of Terrorist Cells". In: *Connections* 24.3 (2002), pp. 43–52. ISSN: 0226-1776. DOI: 10.1.1.16.2612. URL: http://www.insna.org/pubs/connections/v24.html.

[35] Jérôme Kunegis. "The Koblenz Network Collection". In: *Proceedings of the 22Nd International Conference on World Wide Web Companion*. Rio de Janeiro, Brazil: International World Wide Web Conferences Steering Committee, 2013, pp. 1343–1350. ISBN: 978-1-4503-2038-2. URL: http://dl.acm.org/citation.cfm?id=2487788.2488173.

[36] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. "Graph Evolution: Densification and Shrinking Diameters". In: 1.1 (2006). ISSN: 15564681. DOI: 10.1145/1217299.1217301. arXiv: 0603229 [physics]. URL: http://arxiv.org/abs/physics/0603229.

[37] Michael Ley. "The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives". In: *Proceedings of the 9th International Symposium (SPIRE 2002)* (2002), pp. 481–486. DOI: http://dx.doi.org/10.1007/3-540-45735-6_1.

[38] David Liben-Nowell and Jon Kleinberg. "The Link Prediction Problem for Social Networks". In: *Proceedings of the Twelfth Annual ACM International Conference on Information and Knowledge Management (CIKM)* November 2003 (2003), pp. 556–559. ISSN: 1532-2882. DOI: 10.1002/asi.v58:7.

[39] Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. "New perspectives and methods in link prediction". In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10*. ACM Press, 2010, p. 243. URL: http://dl.acm.org/citation.cfm?doid=1835804.1835837.

[40] Tie-Yan Liu. *Applications of Learning to Rank*. 2011, pp. 181–191. ISBN: 978-3-642-14266-6. DOI: 10.1561/1500000016.

[41] Weiping Liu and Linyuan Lu. "Link prediction based on local random walk". In: *EPL (Europhysics Letters)* 89 (2010), p. 6. ISSN: 0295-5075. DOI: 10.1209/0295-5075/89/58007. arXiv: 1001.2467.

[42] L Lovász. "Random walks on graphs: A survey". In: *Combinatorics Paul Erdos is Eighty* 2.Volume 2 (1993), pp. 1–46. ISSN: 03044149. DOI: 10.1.1.39.2847. URL: http://www.cs.yale.edu/publications/techreports/tr1029.pdf.

[43] L Lü and T Zhou. "Link Prediction in Complex Networks: A Survey". In: *Physica A: Statistical Mechanics and its Applications* 390.6 (2010), pp. 1150–1170. ISSN: 0378-4371. DOI: http://dx.doi.org/10.1016/j.physa.2010.11.027. URL: http://www.sciencedirect.com/science/article/pii/S037843711000991X.

[44] Yun Mao et al. "Modeling distances in large-scale networks by matrix factorization". In: *Proceedings of the 4th ACM SIGCOMM conference on Internet Measurement*. IMC '04. New York, NY, USA: ACM, 2004, pp. 278–287. ISBN: 1-58113-821-0. DOI: 10.1145/1028788.1028827. URL: http://portal.acm.org/citation.cfm?id=1028827.

[45] Ak Menon and C Elkan. "Link prediction via matrix factorization". In: *Machine Learning and Knowledge Discovery in Databases*. Vol. 6912. ECML PKDD'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 437–452. ISBN: 978-3-642-23782-9. DOI: 10.1007/978-3-642-23783-6_28. URL: http://link.springer.com/chapter/10.1007/978-3-642-23783-6%5C_28.

[46] Tom Mitchell. *Machine Learning*. Springer, 1997. ISBN: 9783662124055. DOI: 10.1145/242224.242229. URL: https://books.google.de/books?id=e8UknQEACAAJ.

[47] Michael Mitzenmacher. "A brief history of generative models for power law and lognormal distributions". In: *Internet Mathematics* 1.2 (2001), pp. 226–251. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.8075.

[48] M. E. J. Newman. "Clustering and preferential attachment in growing networks". In: *Phys. Rev. E* 64.2 (2001). DOI: 10.1103/PhysRevE.64.025102. URL: http://link.aps.org/doi/10.1103/PhysRevE.64.025102.

[49] M. E. J. Newman. "The structure of scientific collaboration networks." In: *Proceedings of the National Academy of Sciences of the United States of America* 98.2 (2001), pp. 404–9. ISSN: 0027-8424. DOI: 10.1073/pnas.021544898. arXiv: 0007214 [cond-mat]. URL: http://www.ncbi.nlm.nih.gov/pubmed/11149952.

[50] Qing Ou et al. "Power-law strength-degree correlation from resource-allocation dynamics on weighted networks". In: *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* 75.2 (2007), pp. 1–4. ISSN: 15393755. DOI: 10.1103/PhysRevE.75.021102. arXiv: 0603081 [physics].

[51] L Page et al. "The PageRank citation ranking:bringing order to the web." In: *Technical report, Stanford Digital Library Technologies Project* (1998), pp. 1–17. URL: http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf.

[52] G Salton and M J McGill. *Introduction to modern information retrieval.* New York, NY, USA, 1983. URL: http://search.ebscohost.com/login.aspx?direct=true%5C&db=lxh%5C&AN=ISTA2001897%5C&site=ehost-live.

[53] Robert E. Schapire and Yoav Freund. *Boosting*. 2012. ISBN: 9780262017183.

[54] Marina Skurichina and Robert P W Duin. "Bagging, boosting and the random subspace method for linear classifiers". In: *Pattern Analysis and Applications* 5.2 (2002), pp. 121–135. ISSN: 14337541. DOI: 10.1007/s100440200011.

[55] Christian L Staudt and Henning Meyerhenke. "Engineering High-Performance Community Detection Heuristics for Massive Graphs". In: *CoRR* abs/1304.4 (2013), pp. 1–16. arXiv: arXiv:1304.4453v4. URL: http://arxiv.org/abs/1304.4453.

[56] Christian L Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. "NetworKit : An Interactive Tool Suite for High-Performance Network Analysis". In: *Arxiv* 1403.3005 (2014), pp. 1–25. arXiv: 1403.3005.

[57] Ben Taskar, Pieter Abbeel, and Daphne Koller. "Discriminative Probabilistic Models for Relational Data". In: Vapnik (1995).

[58] Ben Taskar et al. "Link Prediction in Relational Data". In: *Networks* 15 (2004), p. 7. URL: http://scholar.google.com/scholar?hl=en%5C&btnG=Search%5C&q=intitle:Link+prediction+in+relational+data%5C#0.

[59] Bimal Viswanath et al. "On the Evolution of User Interaction in Facebook". In: *Proceedings of the 2nd ACM workshop on Online social networks - WOSN '09* (2009), p. 37. ISSN: 1605584452. DOI: 10.1145/1592665.1592675. URL: http://portal.acm.org/citation.cfm?doid=1592665.1592675.

[60] Gary M. Weiss and Foster Provost. "Learning when training data are costly: The effect of class distribution on tree induction". In: *Journal of Artificial Intelligence Research* 19 (2003), pp. 315–354. ISSN: 10769757. DOI: 10.1613/jair.1199. arXiv: 1106.4557.

[61] Wikimedia Commons. *Precision and recall*. 2014. URL: http://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg.

[62] Guang Wu et al. "A novel range-free localization based on regulated neighborhood distance for wireless ad hoc and sensor networks". In: *Computer Networks* 56.16 (2012), pp. 3581–3593. ISSN: 13891286. DOI: 10.1016/j.comnet.2012.07.007. URL: http://dx.doi.org/10.1016/j.comnet.2012.07.007.

[63] Yan Bo Xie, Tao Zhou, and Bing Hong Wang. "Scale-free networks without growth". In: *Physica A: Statistical Mechanics and its Applications* 387.7 (2008), pp. 1683–1688. ISSN: 03784371. DOI: 10.1016/j.physa.2007.11.005. arXiv: 0512485 [cond-mat].

[64] Yang Yang, Ryan N. Lichtenwalter, and Nitesh V. Chawla. "Evaluating link prediction methods". In: *Knowledge and Information Systems* (2014). ISSN: 0219-1377. DOI: 10.1007/s10115-014-0789-0. URL: http://link.springer.com/10.1007/s10115-014-0789-0.

[65] Jonathan S Yedidia, William T Freeman, and Yair Weiss. "Generalized Belief Propagation". In: *Nips* (2000), pp. 689–695. ISSN: 10495258. DOI: 10.1109/ITW.2004.1405304. URL: http://citeseer.ist.psu.edu/yedidia00generalized.html.

[66] Richard Zanibbi. *Pattern Recognition*. Rochester, 2009. URL: http://www.cs.rit.edu/~rlaz/prec20092/slides/Bagging%5C_and%5C_Boosting.pdf.

[67] Aidong Zhang. *Protein Interaction Networks*. Cambridge University Press, 2009. ISBN: 9780521888950. DOI: 10.1017/CBO9780511626593.

[68] Jiawei Zhang and Sy Philip. "Link Prediction across Heterogeneous Social Networks: A Survey". In: (2014). URL: http://www.cs.uic.edu/~jzhang2/files/2014%5C_survey%5C_paper.pdf.

[69] Tao Zhou, Linyuan Lü, and Yi Cheng Zhang. "Predicting missing links via local information". In: *European Physical Journal B* 71.4 (2009), pp. 623–630. ISSN: 14346028. DOI: 10.1140/epjb/e2009-00335-8. arXiv: 0901.0553.

[70] Linhong Zhu, Greg Ver Steeg, and Aram Galstyan. "Scalable Link Prediction in Dynamic Networks via Non-Negative Matrix Factorization". In: *CoRR* abs/1411.3 (2014). URL: http://arxiv.org/abs/1411.3675.

A. Appendix

A.1. ROC curves

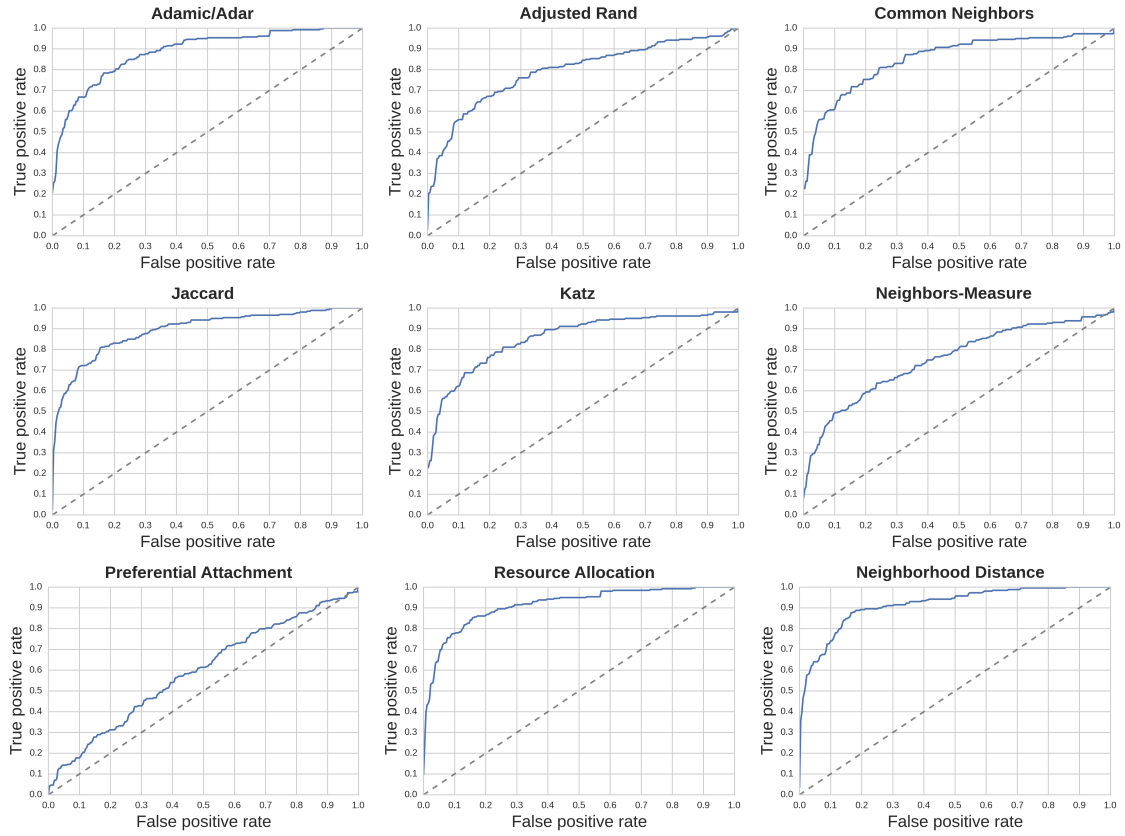

Figure A.1.: ROC curves for similarity indices on Jazz.

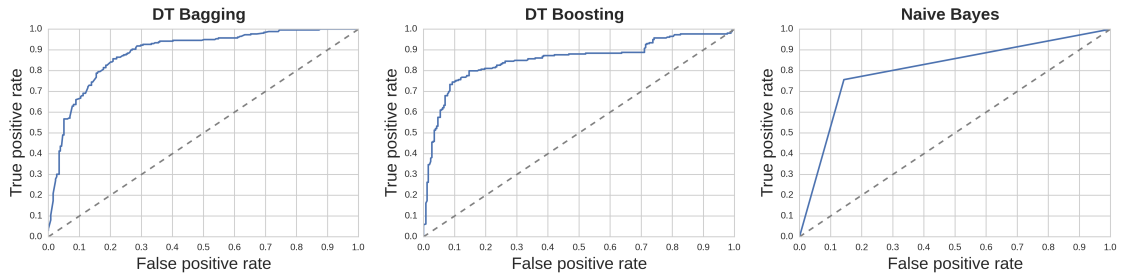

Figure A.2.: ROC curves for supervised methods on Jazz.

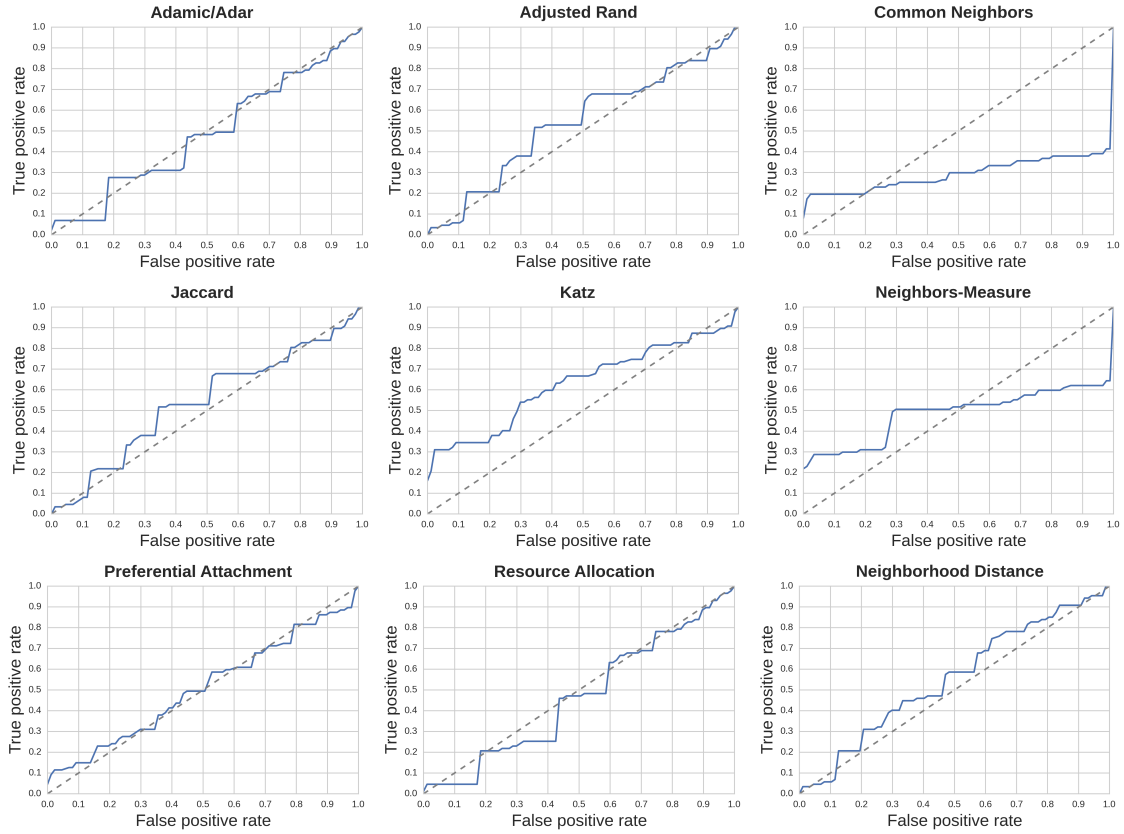

Figure A.3.: ROC curves for similarity indices on Grid.

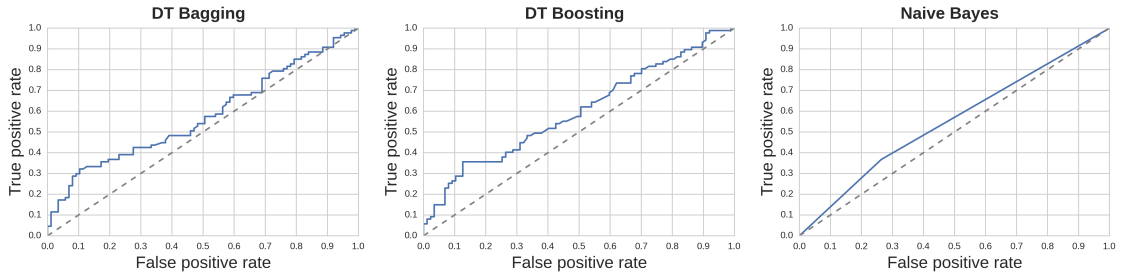

Figure A.4.: ROC curves for supervised methods on Grid.

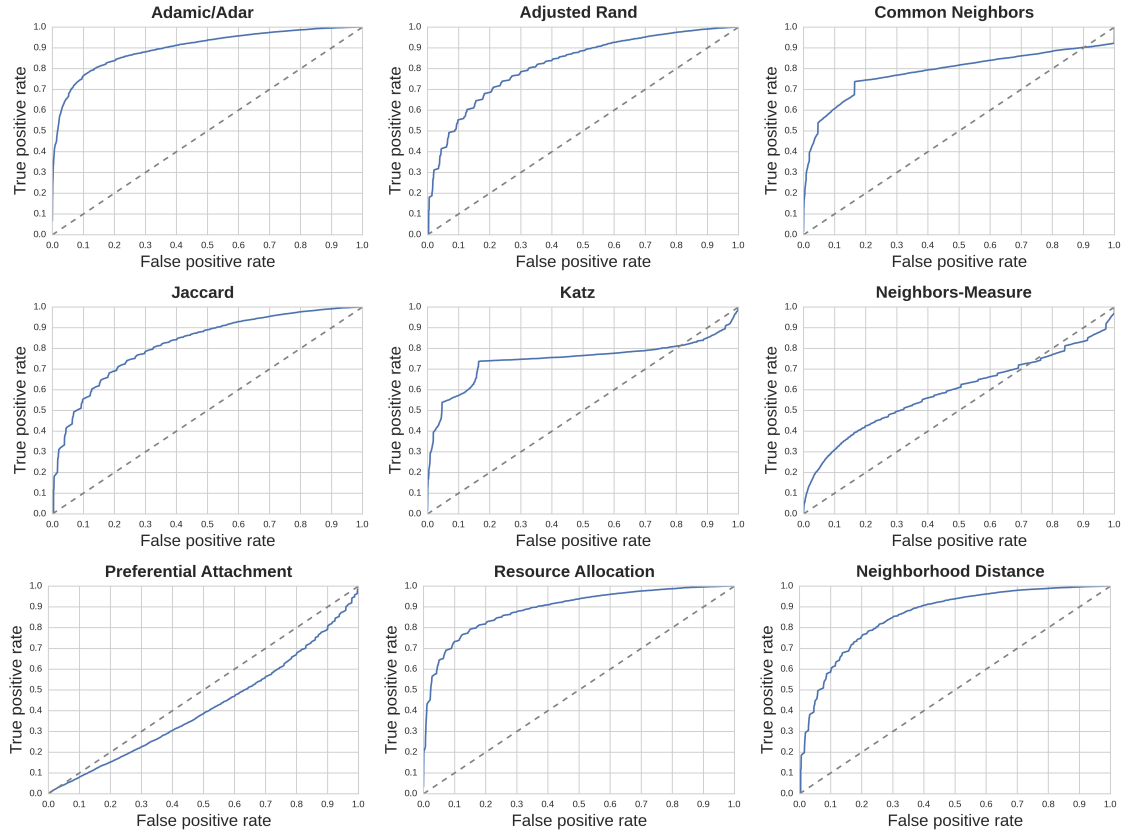

Figure A.5.: ROC curves for similarity indices on Cond-Mat.

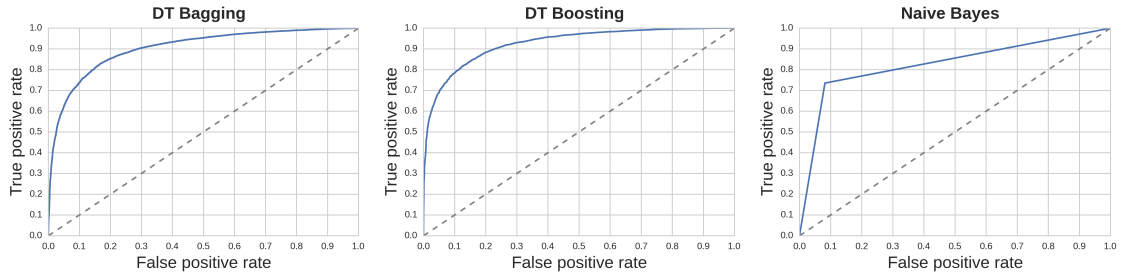

Figure A.6.: ROC curves for supervised methods on Cond-Mat.

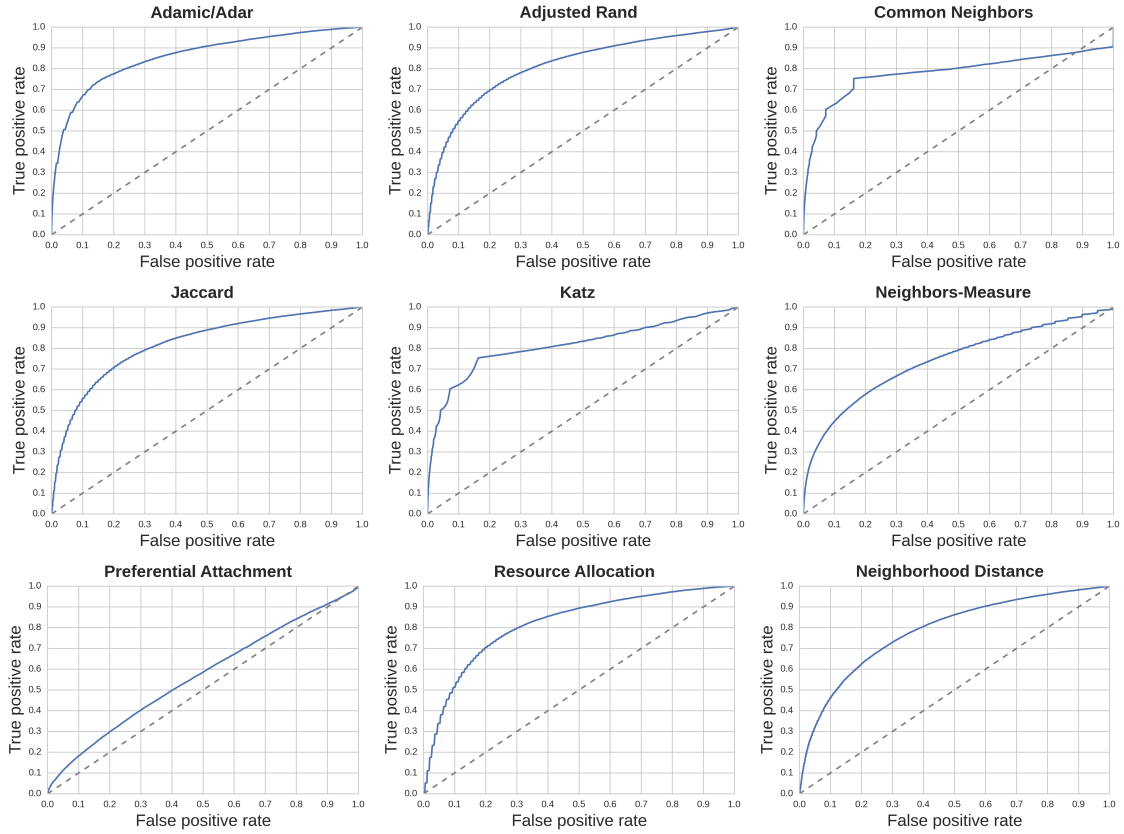

Figure A.7.: ROC curves for similarity indices on Facebook.

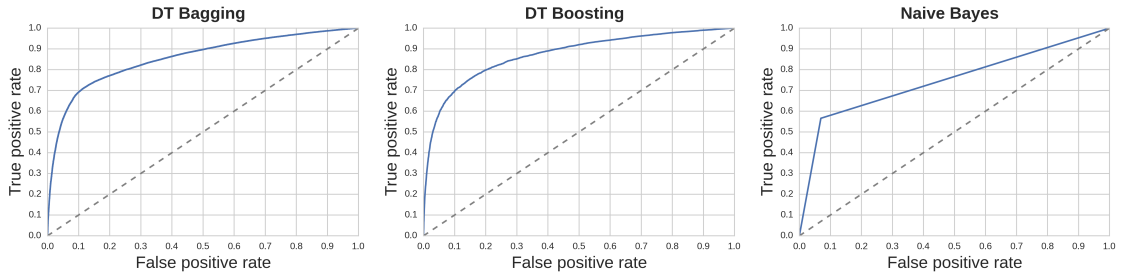

Figure A.8.: ROC curves for supervised methods on Facebook.

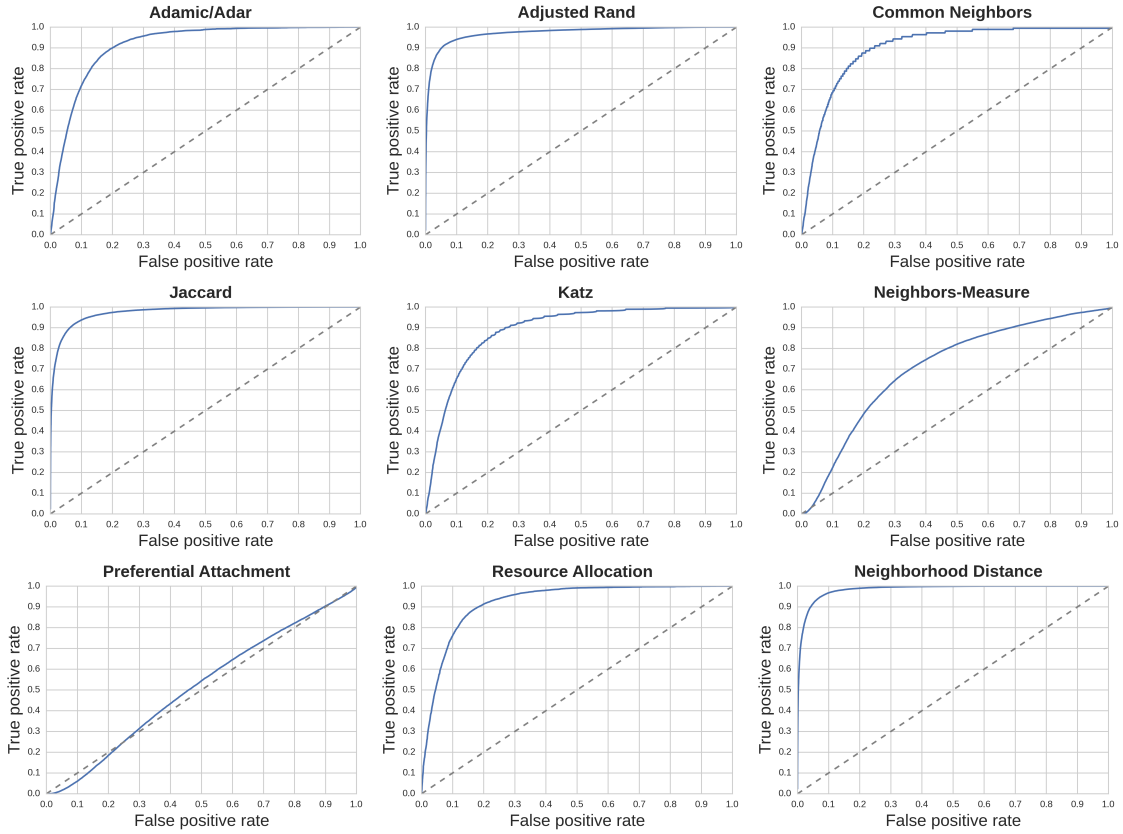

Figure A.9.: ROC curves for similarity indices on Hep-Th.

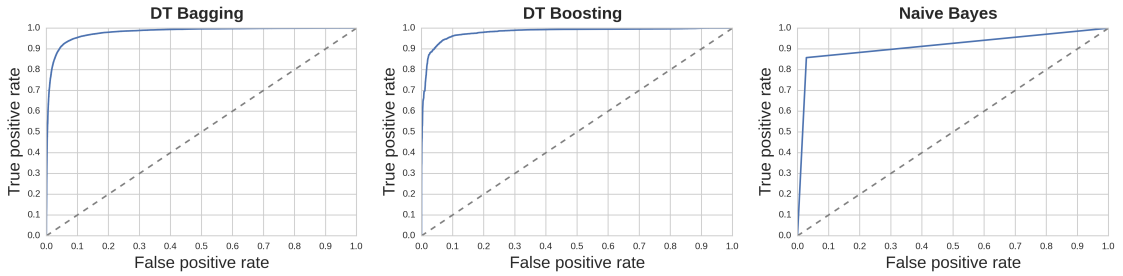

Figure A.10.: ROC curves for supervised methods on Hep-Th.

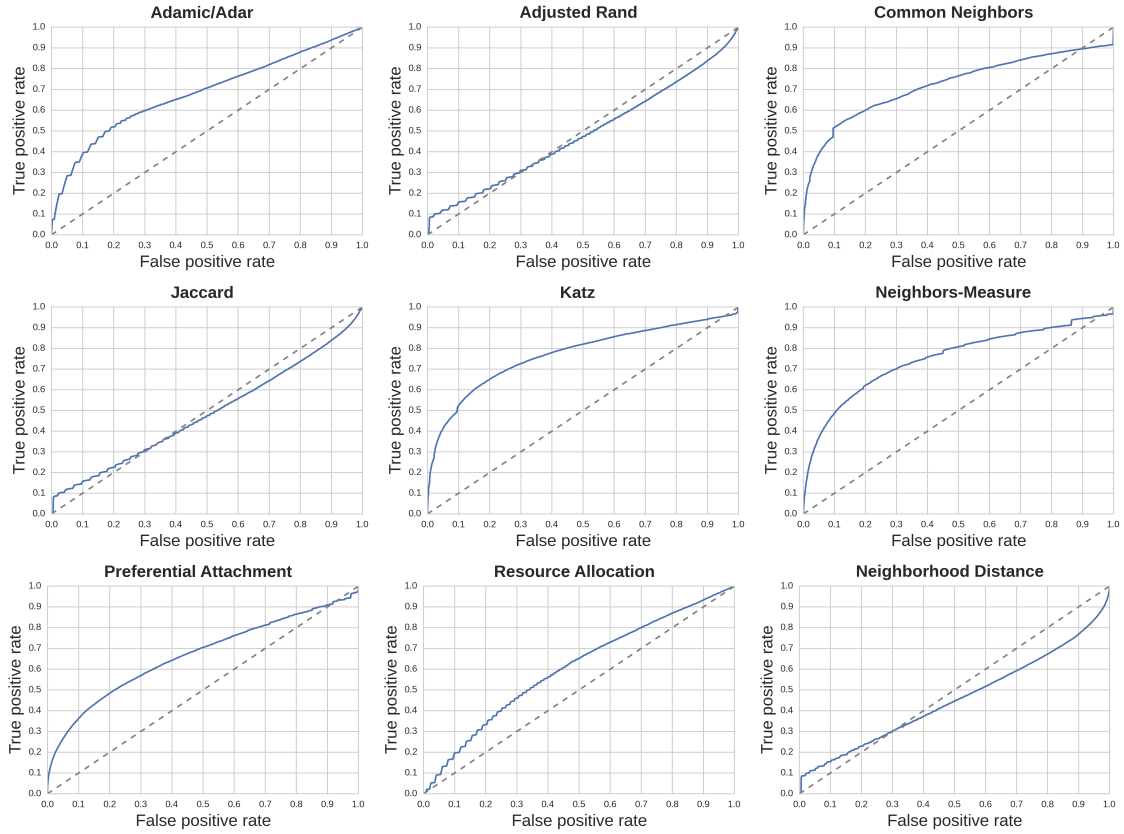

Figure A.11.: ROC curves for similarity indices on DBLP.

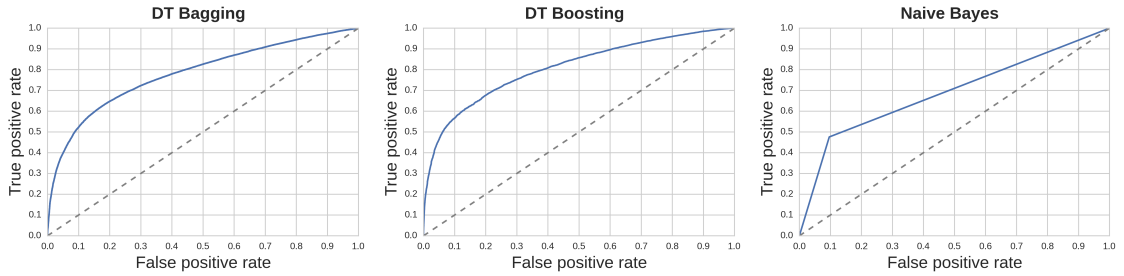

Figure A.12.: ROC curves for supervised methods on DBLP.

A.2. PR curves

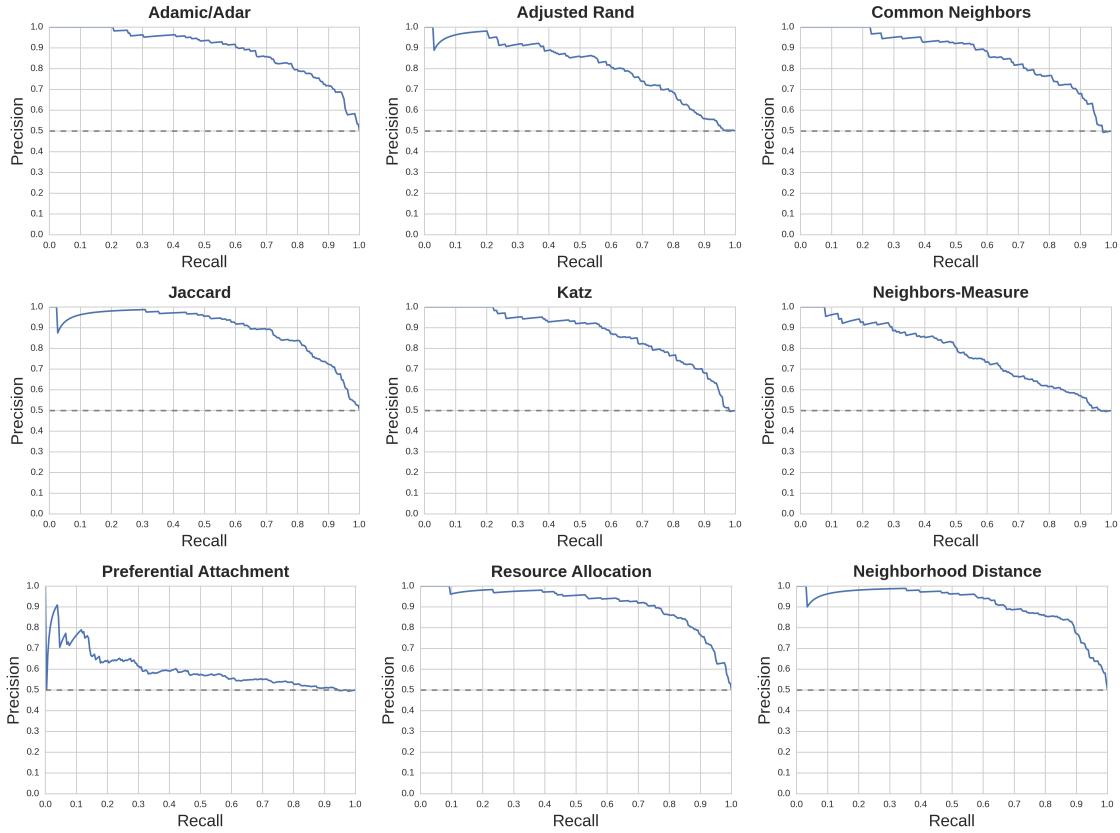

Figure A.13.: PR curves for similarity indices on Jazz.

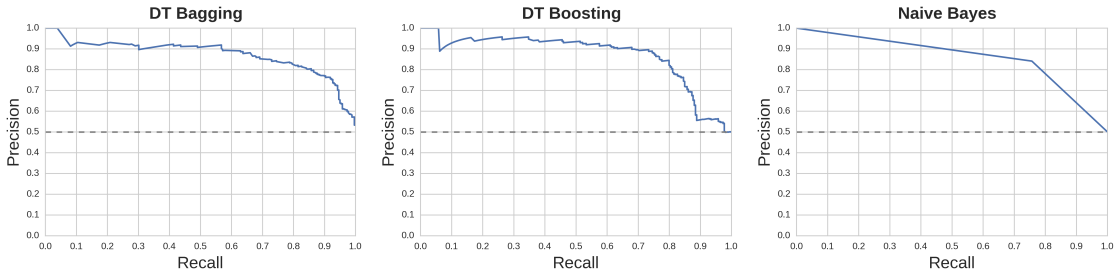

Figure A.14.: PR curves for supervised methods on Jazz.

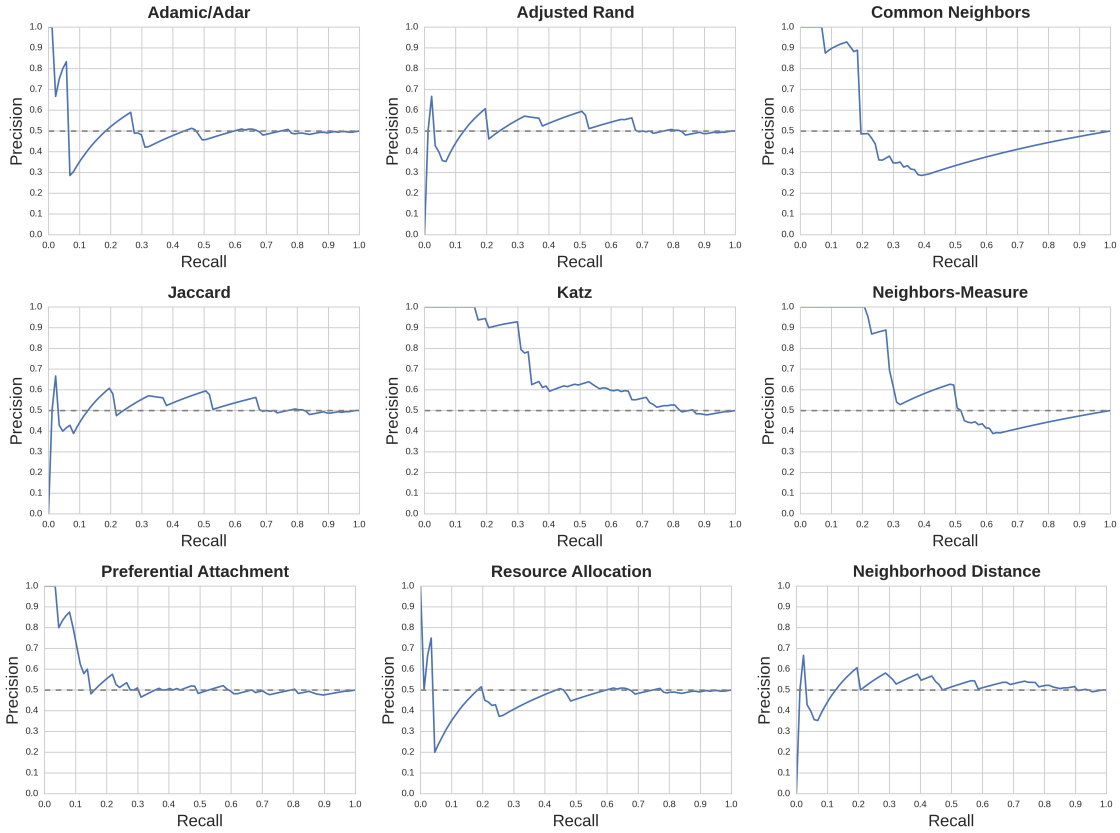

Figure A.15.: PR curves for similarity indices on Grid.

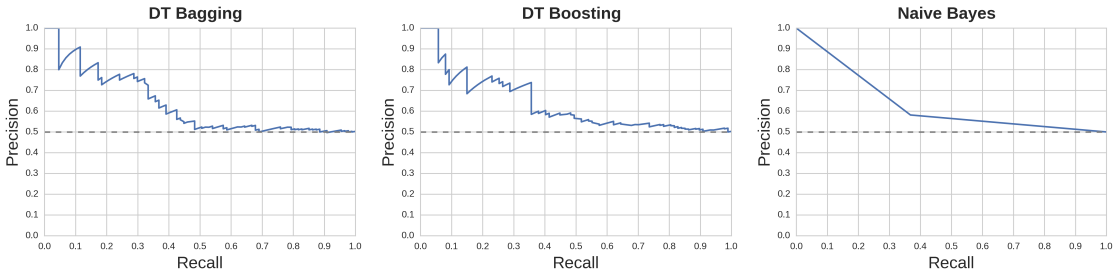

Figure A.16.: PR curves for supervised methods on Grid.

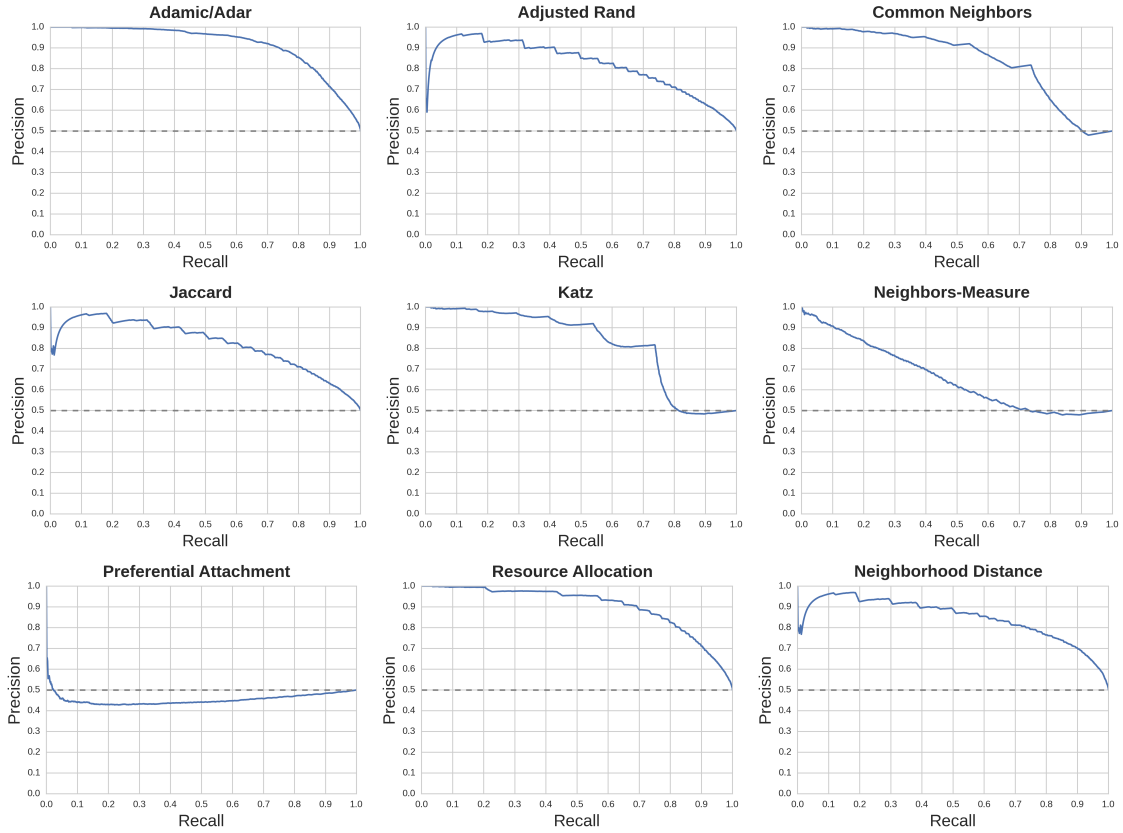

Figure A.17.: PR curves for similarity indices on Cond-Mat.

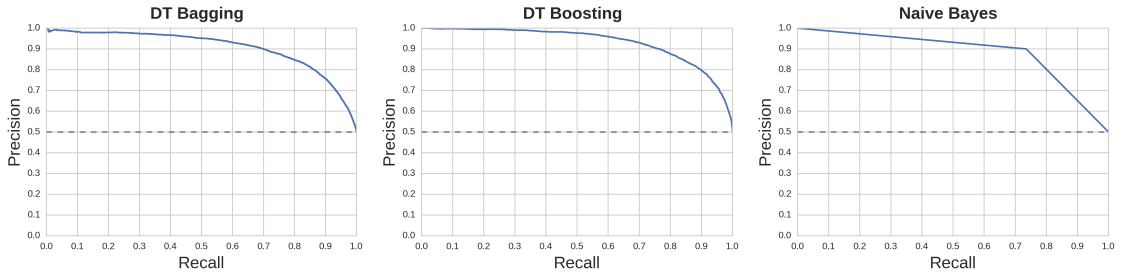

Figure A.18.: PR curves for supervised methods on Cond-Mat.

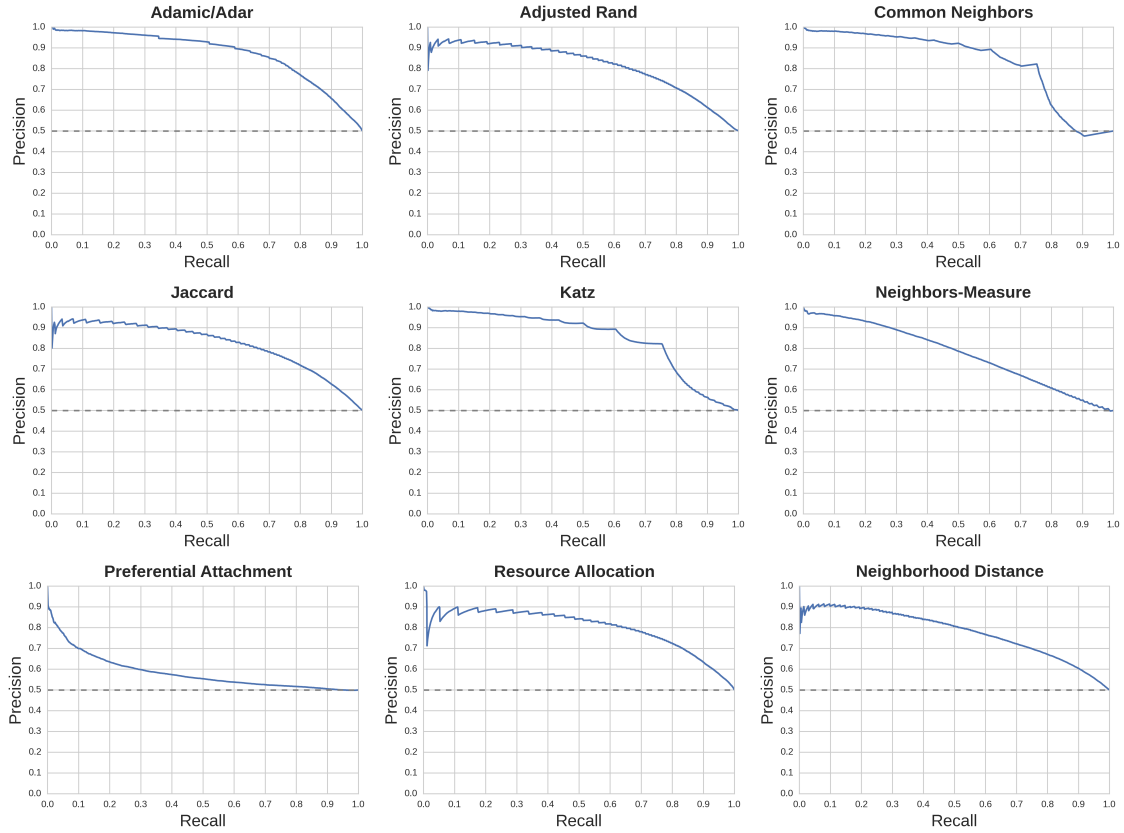

Figure A.19.: PR curves for similarity indices on Facebook.

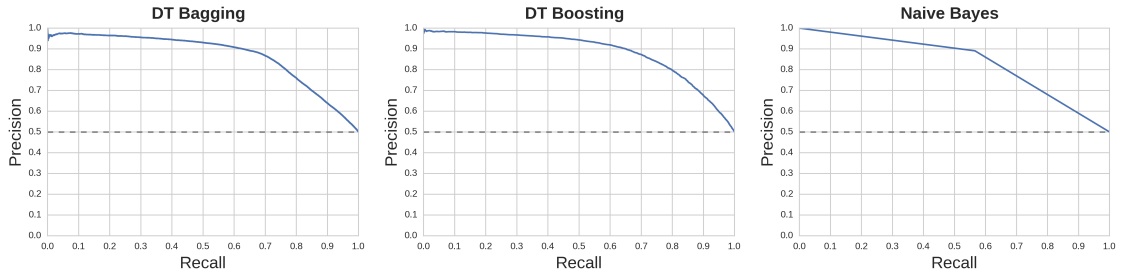

Figure A.20.: PR curves for supervised methods on Facebook.

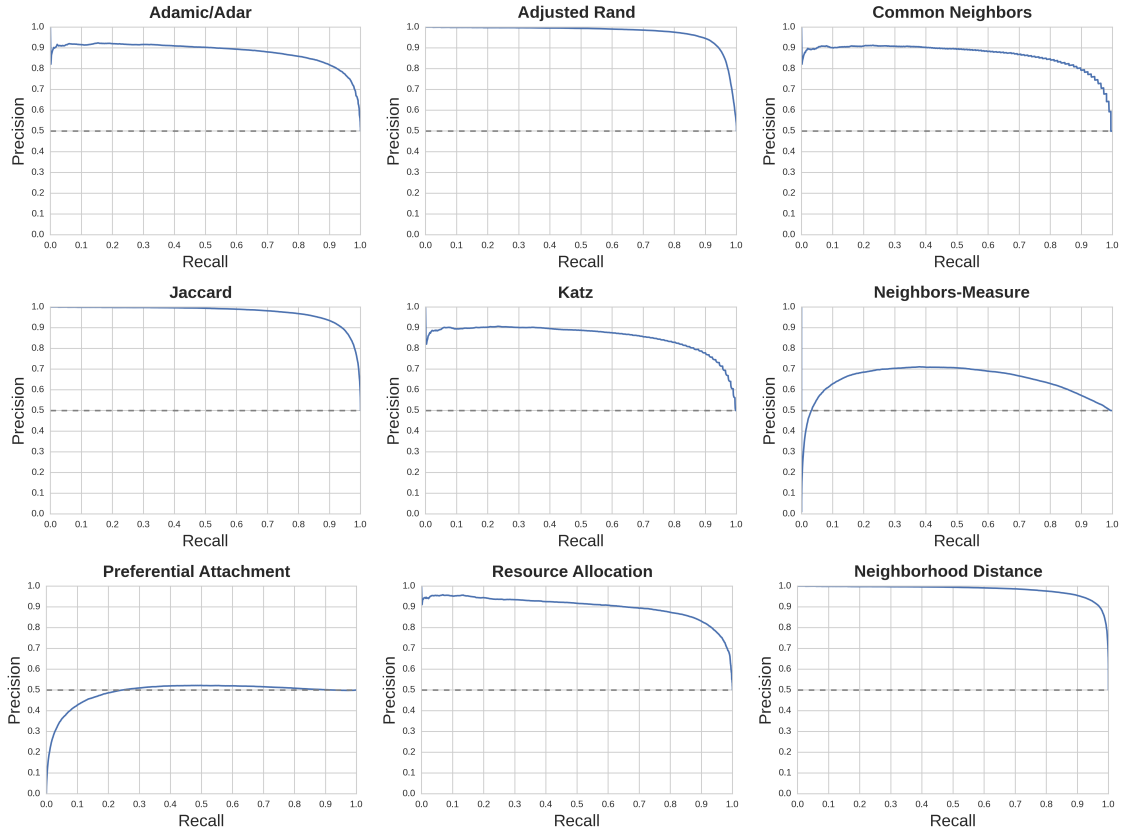

Figure A.21.: PR curves for similarity indices on Hep-Th.

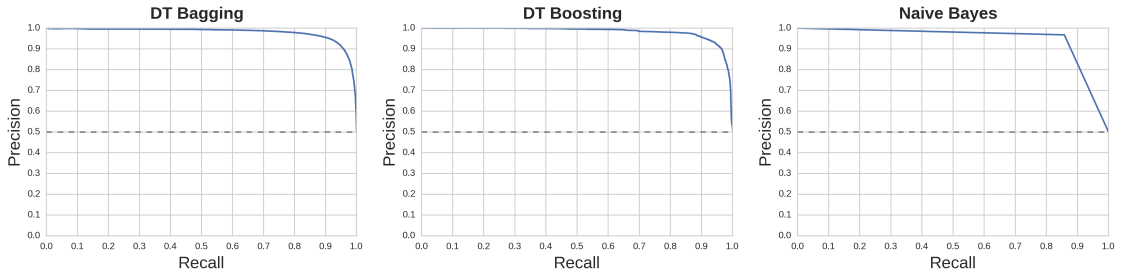

Figure A.22.: PR curves for supervised methods on Hep-Th.

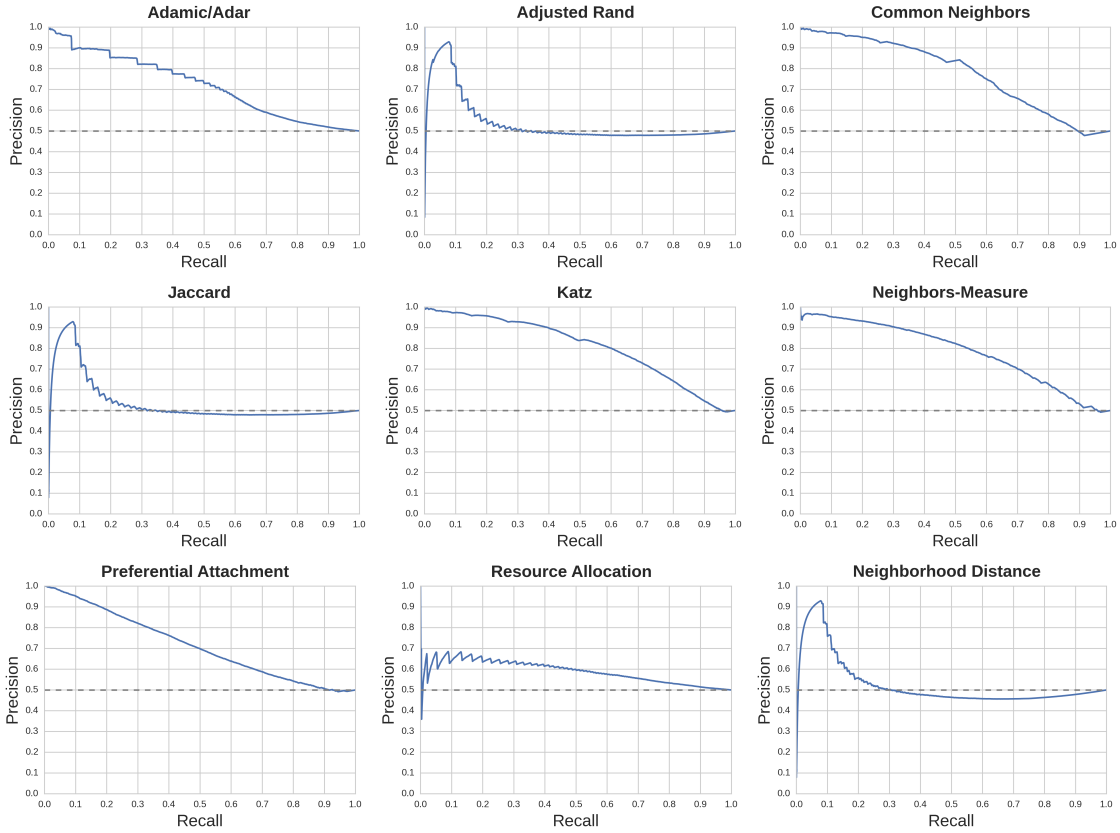

Figure A.23.: PR curves for similarity indices on DBLP.

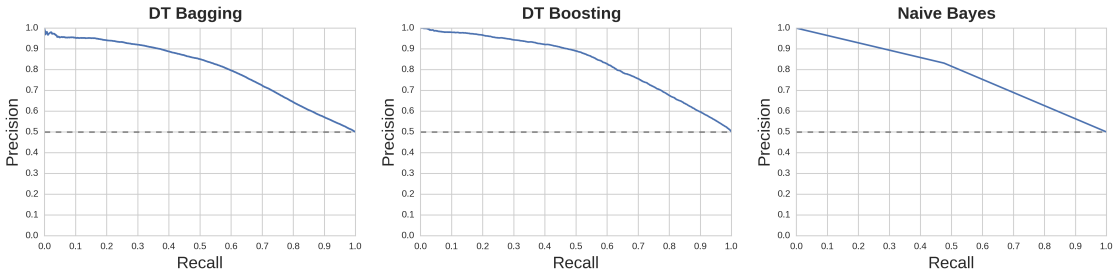

Figure A.24.: PR curves for supervised methods on DBLP.