**DOKUZ EYLÜL UNIVERSITY**
**GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

# LINK PREDICTION IN SOCIAL NETWORKS

**by**
**Merve Işıl PETEN**

**August, 2021**
**İZMİR**

# LINK PREDICTION IN SOCIAL NETWORKS

**A Thesis Submitted to the**
**Graduate School of Natural and Applied Sciences of Dokuz Eylül University**
**In Partial Fulfillment of the Requirements for the Master of**
**Science in Computer Engineering**

**by**

**Merve Işıl PETEN**

**August, 2021**
**İZMİR**

**M.Sc THESIS EXAMINATION RESULT FORM**

We have read the thesis entitled **"LINK PREDICTION IN SOCIAL NETWORKS"** completed by **MERVE IŞIL PETEN** under supervision of **ASSOC. PROF. DR. ZERRİN IŞIK** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

……

Assoc. Prof. Dr. Zerrin IŞIK

Supervisor

……...……

Asst. Prof. Dr. Feriştah DALKILIÇ

(Jury Member)

……...……

Asst. Prof. Dr. Özlem ERDAŞ ÇİÇEK

(Jury Member)

Prof. Dr. Özgür ÖZÇELİK

Director

Graduate School of Natural and Applied Sciences

## ACKNOWLEDGEMENTS

There are many who helped me along the way on this journey. I want to take a moment to thank them.

First of all, I would like to thank my advisor, Assoc. Prof. Dr. Zerrin IŞIK, whose expertise was invaluable through the process. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would like to express my deepest appreciation to Dr. Ahmet Serkan Karataş, who provided me encouragement and patience. Without you believing in me, I never would have finished it.

Special thanks to my family, *Ömer, Nalan* & *Efsun*, for all of the sacrifices that you've made on my behalf and always tolerating me. I will also cannot forget my cats' irreplaceable love.

Finally, I want to thank myself for never giving up even in the darkest time.

Merve Işıl PETEN

**LINK PREDICTION IN SOCIAL NETWORKS**

## ABSTRACT

Link prediction is used to forecast link evolution over time in networks. It has been used in several areas such as bioinformatics, online recommendation systems, e-commerce sites, collaboration networks and social networks. Predicting user behavior has become crucial with the expansion of multiuser online systems. This study aims to provide an insight to performance characteristics, both in terms of effectiveness and efficiency, for several link prediction methods. Four fundamental link prediction methods (i.e., common neighborhood, Adamic-Adar, preferential attachment, and Jaccard coefficient) that have been reported in the literature, and a novel metric have been evaluated. The proposed metric makes predictions on the premise that a newly joined member tends to make connections with available nodes that are popular amongst the network. Real-life data sets obtained from the Stanford Large Network Dataset Collection. Common neighborhood, Adamic-Adar and preferential attachment metrics provided more successful results than the others in all networks. In terms of running time, preferential attachment, common neighborhood and the novel metric of this study are the fast-running ones. The highest F1-score is 0.12 in the email-Eu-core and Reddit networks achieved by the Adamic-Adar metric. This study presents and discusses the performance of several link prediction methods on temporal networks. It provides some insights for practical usage of link prediction metrics.

**Keywords:** Link prediction, structure based metrics, temporal network

# SOSYAL AĞLARDA BAĞLANTI TAHMİNİ

## ÖZ

Bağlantı tahmini, ağlarda zamana göre değişen bağlantı gelişimini tahmin etmek için kullanılır. Biyoinformatik, çevrimiçi öneri sistemleri, e-ticaret siteleri, işbirliği ağları ve sosyal ağlar gibi çeşitli alanlarda kullanılır. Çok kullanıcılı çevrimiçi sistemlerin yaygınlaşmasıyla kullanıcı davranışını tahmin etmenin önemi arttı. Bu çalışma, çeşitli bağlantı tahmin yöntemleri için hem etkinlik hem de verimlilik açısından performans özelliklerine bir fikir vermeyi amaçlamaktadır. Literatürde bildirilen dört temel bağlantı tahmin yöntemi (Common Neighbourhood, Adamic-Adar, Preferential Attachment ve Jaccard Coefficient) ve sunulan yeni bir metot kullanılmıştır. Önerilen metot, yeni katılan bir üyenin ağ arasında popüler olan mevcut üyelerle bağlantı kurma eğiliminde olduğu varsayımıyla tahminlerde bulunur. Kullanılan gerçek veri setleri Stanford Large Network Dataset kütüphanesinden alınmıştır. En başarılı sonuçlara Common neighbourhood, Adamic-Adar ve Preferential attachment metotlarıyla ulaşılmıştır. En hızlı metrikler Preferential attachment, Adamic-Adar ve yeni metriktir. En yüksek F1 değeri, Adamic-Adar metriğiyle email-Eu-core ve Reddit veri setlerinde 0,12 olarak hesaplanmıştır. Bu çalışma zamansal ağlar üzerindeki çeşitli bağlantı tahmin yöntemlerinin performansını sunmakta ve karşılaştırmaktadır. Bağlantı tahmini ölçümlerinin pratik kullanımı için bilgiler içermektedir.

**Anahtar kelimeler:** Bağlantı tahmini, yapı tabanlı metrikler, zamansal ağ

**CONTENTS**

**Page**

**LIST OF FIGURES**

**Page**

## LIST OF TABLES

**Page**

# CHAPTER ONE
# INTRODUCTION

## 1.1 Motivation

Link prediction is connected to real life problems thus it is a popular research area. It can be used for recommender systems, socialization, finding potential collaborators, and online shopping (Wang, Zu, Wu & Zhou, 2015). To find out which entities may generate new links in the near future is a very important and challenging problem (Shan, Li, Zhang, Bai & Chen, 2020). For instance, accurate identification of product sets that would appeal to individual users in a large e-commerce system is becoming a greater challenge everyday as the number of available products and the size of userbases increase steadily. Tackling with such problems has opened way for ramifications in the approaches used for link prediction and several different strategies have been devised for this purpose.

It is important to understand and use these data to correctly predict the behavior of the users. Especially for e-commerce sites and social media, the number of users and the time they spend online are two important parameters. To ensure that, making better predictions are significant. For example, when a user search for a product on Amazon it is crucial for site to suggest similar products that will attract interest of the user. If a user easily find the product he/she searches and leaves the site satisfied, it is more likely for him/her to visit this site again. This loyalty is important for Amazon to increase their profit. Another example is one of the most popular social media application Instagram, when users are able to find accounts that they want to follow or watch videos suitable for their interest they will continue to use the application. People can share photos about their lives, communicate with other people and this way they feel fulfilled. For an application to be popular delivering users what they want is critical. So, they need to understand the act of users and show matching accounts/products or brands to them. It is similar for Reddit as well, one of the data sources used in this study, when a user finds the subject he/she wants

to read about, he/she should be able to find other users comments and the application should offer other titles that would be appealing to the user. For online helping sites like Stack-overflow and Math-overflow, users post the questions and other users help them to solve that problem. They should retrieve similar questions when user searches for one.

The motivation of this study is to propose a link prediction approach that explores multiple temporal network data sources and link prediction methods to increase the accurateness of predicted links. Besides, a novel metric is proposed and compared the results after evaluating with metrics. The methods are investigated and the type of networks the methods worked better on are explained.

### 1.2 Problem Definition

Subject of link prediction takes an important place in the field of graphs and networks. Link prediction techniques consist of node-based metrics, topology-based metrics, social-theory-based metrics and machine learning based methods. Feature selection is the essential part of feature-based classification. Link prediction problem has six categories as temporal link prediction, link prediction in heterogeneous networks, link prediction with active and inactive links, link prediction in bipartite networks, link prediction for unfollow or disappearing links and link prediction scalability (Wang et al., 2015). In this study, topology based metrics have been used for temporal link prediction.

In temporal networks entities are represented by nodes and relationships between them are represented by links. "Temporal link prediction problem is defined as graph $G$, $G = (V, E)$ be a dynamic network, where $V$ is the set of vertices and each edge $(u, v) \in E$ represents a link between $u$ and $v$" (Divakaran & Mohan, 2020). These entities and relationships appear and disappear over time. Temporal link prediction is a task of predicting the links in a network that would appear in a near future at the $t+1$ time, by looking at the snapshots of the network from a period $1$ to $t$ time (Liben-Nowell & Kleinberg, 2007). Link prediction can be classified in three categories as the newly added

links, disappeared links or both (Gao, Musial, Cooper & Tsoka, 2015). This research is only focused on the newly added ones.

Link prediction techniques can be grouped as node-based, topology-based and social theory. Topology-based techniques have sub categories as neighbor-based, path-based and random walk-based. In this study, link prediction is performed using different temporal networks and neighbor-based methods. Efficiency of neighbor-based methods is analyzed in terms of both accurate prediction of temporal links and running times of methods.

### 1.3 Contribution

Contribution of this study is twofold: the performance of four naive methods that have already been reported in the literature are realized, evaluated and analyzed both in effectiveness and in efficiency. Second, a novel method to predict future link evolution that can especially be useful in domains that include, but not limited to, e-commerce and social networks is proposed.

In this study, six real-life temporal data sets are used and the topology-based methods are evaluated. Presenting a comparative analysis of different networks, a solution to run big data sets in shorter time using chunks is found, so there is no need for special computer systems for evaluation. The results are compared to obtain which methods works better on which type of network. For this purpose, the relationship between the metrics that performed well and the networks that give the results are examined and the reasons are searched.

Part of this thesis (Peten & Işık, 2021) had been accepted and presented in Global Conference on Engineering Research (GLOBCER'21).

## 1.4 Organization of the Thesis

This thesis consists of five chapters organized as follows:

In Chapter 2, detailed background information and literature review of relevant researches and approaches are provided. Thus, the history of the subject and the basic concepts are outlined and our approach to link prediction, evaluation metrics and temporal networks are correlated.

In Chapter 3, we introduce our framework in terms of what we evaluate and measure and why we choose this methods and metrics. After that, we explain our preparation process such as creation of the graphs, finding the giant component and building the dictionaries. Then, we mention about our strategy and the aim of third-party tools used while compiling the code. We finalize with the details of execution.

In Chapter 4, we present the experimental results of link prediction performance for each network and evaluate them in terms of precision, recall and F1-Score values. We also examine the compilation time of every network for each of the methods. We elect the most successful metrics and clarify the reasons behind.

In Chapter 5, we conclude our study and mention about what can be added in the future.

# CHAPTER TWO
# LITERATURE REVIEW

In this chapter, relevant studies and techniques for link prediction will be introduced. Discussions and history of the subject will be given for seminal and significant studies. Similar studies will be examined in detail both in common and different point of views.

## 2.1 Networks and Link Prediction

Large amounts of data are encountered in countless examples in the real world, ranging from e-commerce systems to online forums. These data are considered to be networks among different pieces of entities and are often represented using graphs. In theory a network can be static, however, most of the examples seen in the real-world change over time. Such changes occur by the addition or deletion of entities and relationships over time, and networks that exhibit this kind of behavior are called dynamic or temporal networks (Casteigts, 2012; Holme, 2012; Kostakos, 2009).

Since dynamic and temporal networks are in constant evolution it is very important to be able to predict how the network would change as time passes. Link prediction addresses this challenge and aims to predict the links in a changing network that would appear in its next state of period (Liben-Nowell & Kleinberg, 2007). Link prediction can take place in various scenarios to analyze and solve problems such as product recommendation, e-mail networks, co-author networks (Hasan & Zaki, 2011).

Link prediction is not an easy task due to the time-varying structure of the temporal networks. There are several approaches reported in the literature to tackle this challenge. These approaches can be classified into mainly two categories: baseline methods, which employ relatively basic heuristics, and enhancements applied to these methods. There are many methods belonging to these categories, which will be discussed in the following subsections.

This study aims to evaluate the performance of selected neighborhood-based baseline methods, and a novel baseline prediction method that will be presented in the following chapter, in terms of both effectivity and efficiency. Therefore, it aims to provide an insight to practitioners that would use the selected methods in a real-world link prediction problem.

## 2.2 Baseline Methods

The baseline methods aim to use the current topology of the network. They define a similarity index for the nodes, calculate the values for the node tuples of the defined index for the current state of the network, and use these values to predict which links may come into existence and which links may be removed from the network in its next state.

The Common Neighbor (CN) method (Yao, Wang, Pan & Yao, 2016) quantizes the similarity between two nodes by finding the number of nodes that are adjacent to both nodes. The higher the number of common neighbors the higher a similarity value to nodes get. An application of this method is the study by Yang-Tian & Zhang (2012) which aims to perform link prediction on a Facebook wall posts dataset.

The Jaccard Coefficient (JC) (Jaccard, 1901) method uses a similar approach to CN, where it normalizes the number of common neighbors by taking into account the proportion of common neighbors to the total number of neighbors the nodes have. A pair with a higher ratio of common neighbors receive a higher similarity value in this method. Niwattanakul et al. (2013) uses this method to find the similarity between keywords and improve search performance for search engines.

Adamic Adar (AA) (Adamic & Adar, 2003) formulation is like the reverse sibling of the JC formulation, where lower ratios receive higher values. It is seminally used to compute web page similarities. Liben-Nowel and Kleinberg (2007) use AA, among many other methods, to predict possible future collaborations in academic publishing.

Preferential Attachment (PA) (Barabasi, 2002) is based on the premise that nodes with high number of links are more likely to acquire new links. Thus, node pairs containing nodes with high degrees receive a greater similarity value. Capocci et al. (2006) use PA to build a statistical model in order to predict the growth of Wikipedia.

Apart from the baseline methods discussed above, there are many other simple methods based on neighborhood in the network. Hub Promoted (Ravasx et al., 2002) and Hub Depressed (Zhou & Zhang, 2009) checks how overlapped two nodes neighbors are and assign a similarity value depending on the degree with higher and lower nodes, respectively. Leiht-Holme-Newman (2006) considers a ratio of number of common neighbors that exist over number of common neighbors expected to exist and assigns a higher value to those with a higher ratio.

There also exist methods that are based on path-based metrics. For instance, Katz (Katz, 1953) computes all paths between two nodes and assigns higher similarity values to the links en route that would form the shortest path. FriendLink (Papadimitriou, Symeonidis & Manolopoulos, 2012) also computes all paths, however, bound the traversal length with a certain limit. Local Path (Lu, Jin & Zhou, 2009) augments neighborhood concept by not just taking the immediate neighbors but taking into account the neighbors that have a path length of 2 and 3.

Some methods adopt a random traversal strategy, where a random walk is formed through neighbors and a probabilistic transition function is used to take the next step. Hitting Time (HT) (Fouss, Pirotte, Renders & Saerens, 2007) performs an arbitrary walk between two certain nodes to find the number of probable steps between them and assigns a higher prediction value to the links forming the path. Commute Time (Fouss et al., 2007) adds symmetry to HT by walking back. SimRank (Jeh & Widom, 2002) adopts a recursive approach and assumes two nodes are similar if they are connected to similar nodes. Rooted PageRank (Liben-Nowell & Kleinberg, 2007) uses probability to guess how likely a node is to be visited through a random walk and assigns ranks to nodes in order to find

similarity values. PropFlow (Lichtenwalter, Lussier & Chawla, 2010) is similar to the Rooted PageRank method, except that it limits the length of the path to be walked.

### 2.3 Enhanced Methods

There are various works that aim to enhance the aforementioned baseline methods. For instance, Xu & Zhang (2013) facilitates a time attribute to extend these methods and enhance the performance results. The paper proposes of active factor usage on two datasets, Citation Network and Cooperation Network collected from ArXiv. Preferential Attachment, Common Neighborhood, Jaccard Coefficient and Adamic-Adar methods are used with and without active factor. As a result, the AUC value of Preferential Attachment method increases 0.1, and becomes 0.9 with active factor while others remain same.

Tylenda et al. (2009) propose to use network history to create edge weights and increase the performance of the Adamic Adar method. A novel testing method proposed and applied on the datasets DBLP and astro-ph. Normalized Discounted Cumulative Gain and Average Normalized Rank methods are defined, even though the score of the first method is similar, Preferential Attachment is better than the second method. The best precision score achieved by the random classifier on astro-ph data set is 8.56%.

Facilitating machine learning approaches, to enhance the efficiency of baseline methods have also been reported in the literature by Ramya By et al. (2020). Twitch dataset, which is a network of gamers that consists of 7.126 nodes and 35.324 edges collected from the Stanford Large Network Dataset Collection. After random sampling applied to missing edges and edges that are already exist in the network are classified as positive and negative, 10 features used. The highest value of precision, recall, F1-Score and accuracy achieved using XGBoost algorithm, the accuracy value is 97.72%.

Wohlfarth and Ichise (2008) propose to utilize semantic and event-based features to improve the prediction success while aiming to find researchers whose collaboration

would be highly prolific. Valverde-Rebeza and de Andrade Lopes (2013) suggests facilitating features based on Twitter users' interests and behavior, thus augment the topology of the network with the community information, to predict future links in Twitter network. Pech et al. (2019) proposes to transform the link prediction problem into a linear optimization problem and use the solutions devised in this mature area. Menon and Elkan (2011) utilizes matrix factorization for link prediction. Pech et al. (2017) redefines the link prediction problem as a matrix completion problem to be able to use the solutions reported in that area.

# CHAPTER THREE
# METHOD

In this chapter, the formulas and the complexity of the link prediction methods will be given, after that data sets will be explained, followed by the preparation of the data. Finally, the evaluation details will be revealed.

## 3.1 Neighbor-Based Link Prediction Methods

This study focuses on neighbor-based metrics that take place under the topology-based techniques. Networks have been modelled as graph G = <V, E>, where V is the set of nodes and E is the set of edges. Summary of the applied methods in this study can be found below.

### *3.1.1 Common Neighborhood*

Assumption of this method is if two nodes have lots of common neighbors the probability that they will be connected in the future is also high. If m is the average degree in a network, the running time complexity of this method is O ($Nm^2$).

$$CN(x, y) = |\Gamma(x) \cap \Gamma(y)| \tag{3.1}$$

where x, y denote nodes, N denotes number of nodes in the network. Γ (x) and Γ (y) denote the neighboring nodes of x and y, respectively (Tylenda et al., 2009).

### *3.1.2 Jaccard Coefficient*

It supposes if neighbors of two nodes include lots of common neighbors, they are more likely to connect in the future. The running time complexity of this method is O ($Nm^2$).

$$JC(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|} \quad (3.2)$$

### *3.1.3 Adamic-Adar Index*

This method assigns higher importance to node pairs with fewer common nodes. The running time complexity is also O ($Nm^2$).

$$AA(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{log\ |\Gamma(z)|} \quad (3.3)$$

where z is a common neighbor of node x and y.

### *3.1.4 Preferential Attachment*

This method gives higher scores to node pairs that have high degree. The running time complexity of this metrics is O ($N^2m^2$).

$$PA(x,y) = |\Gamma(x) * \Gamma(y)| \quad (3.4)$$

### *3.1.5 Popularity Method*

Popularity method is a novel link prediction method recently proposed by Peten and Işık (2021). The motivation behind this new metric comes from daily used applications. Nowadays, most of the people sign up for social media and they make online shopping more frequently. When a new user signing up to a social media platform such as Twitter or Instagram, accounts of celebrities or popular people are directly suggested to follow for the new user. When a new user searches for something to watch, YouTube suggests the most popular videos. On the other hand, the best seller items are listed on e-commerce web sites like Amazon. As a result of such content suggestions, the chance of a new link occurrence between these two nodes in a network become higher.

The Popularity Method is conducted on the idea that a new node with fewer neighbors has a higher probability to connect nodes with many connections. The idea behind this method is similar to Preferential Attachment with an opposite manner, in Preferential Attachment both nodes should have high degree. The popularity method considers the degree values of all nodes then it sorts them; it gives a higher matching chance to lowly connected and highly connected ones by considering absolute value of their degree differences.

$$PM(x,y) = abs(|\Gamma\,(\mathrm{x}) - \Gamma\,(\mathrm{y})|) \tag{3.5}$$

### 3.2 Data sets

The study has been conducted on six different temporal networks, downloaded from Stanford Large Network Dataset Collection (Leskovec, Krevl, 2021). The first one is Reddit Hyperlink Network; it represents the connections between two users. It has 55.863 nodes and 858.490 edges. The attributes of the dataset are the source subreddit, target subreddit, post id, timestamp, link sentiment and properties. This network is directed, signed, temporal and attributed. Each post has a title and a body, therefore there are two networks, and the network with hyperlinks extracted from body of the posts has been used (Kumar, Hamilton, Leskovec & Jurafsky, 2018). Source subreddit, target subreddit and timestamp columns are used for experiments, the others are eliminated. The attribute information can be seen in Figure 3.1 below.

| | SOURCE_SUBREDDIT | TARGET_SUBREDDIT | POST_ID | TIMESTAMP | LINK_SENTIMENT | PROPERTIES |
|---|---|---|---|---|---|---|
| 2 | leagueoflegends | teamredditteams | 1u4nrps | 12/31/2013 16:39 | 1 | 345.0,298.0,0.75652173913,0.0173913043478,0.0869565217391,0.1 |
| 3 | nfl | cfb | 1u4sjvs | 12/31/2013 17:37 | 1 | 1124.0,949.0,0.772241992883,0.0017793594306,0.0578291814947, |
| 4 | theredlion | soccer | 1u4qkd | 12/31/2013 18:18 | -1 | 101.0,98.0,0.742574257426,0.019801980198,0.049504950495,0.059 |
| 5 | dogemarket | dogecoin | 1u4w7bs | 12/31/2013 18:35 | 1 | 1328.0,1110.0,0.768825301205,0.0143072289157,0.0753012048193 |
| 6 | gfycat | india | 1u5df2s | 12/31/2013 22:27 | 1 | 2849.0,2467.0,0.704106704107,0.00737100737101,0.03861003861, |
| 7 | posthardcore | bestof2013 | 1u5ccus | 12/31/2013 23:16 | 1 | 3927.0,3488.0,0.719887955182,0.0229182582124,0.0825057295646 |

Figure 3.1 The structure and attributes of the Reddit data set

The second data set is Stack Overflow temporal network; it consists of the answers, questions and comments on the stack exchange web site. This is the biggest data set with 2.601.977 nodes and 63.497.050 temporal edges (Paranjape, Benson & Leskovec, 2017). Dataset has the source user, target user and timestamp information. Since, this data is too big, it is separated looking at the years as 2008 and 2009.

The third data set is Math Overflow temporal network, like Stack Overflow it includes the interactions on the stack exchange web site Math Overflow. It has 24.818 nodes and 506.550 temporal edges (Paranjape et al., 2017). Some of train and test sets can be seen in Figure 3.2 below.

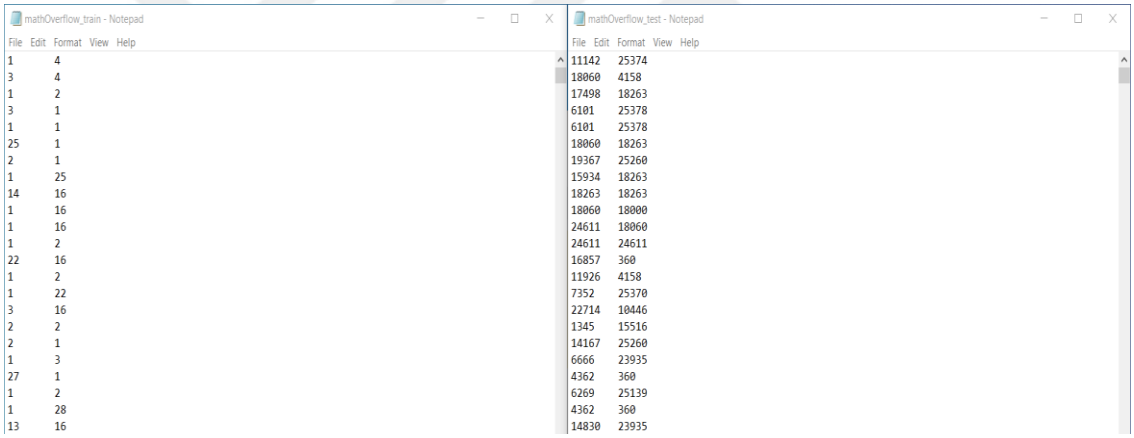


Figure 3.2 Example of seperated data as train and test set for Math Overflow Network

The fourth data set is another stack exchange interaction network named Super User. The dataset has 94.548 nodes and 479.067 temporal edges.

The fifth data set is named email-Eu-core temporal network, an email data from a large research institution. A directed edge is created to save the information about sender, receiver, and the receipt time of the e-mail. For each recipient of the email, a separate edge is created. This data has 986 nodes and 332.334 temporal edges (Yin, Benson, Leskovec & Gleich, 2017).

The sixth data set is Bitcoin Alpha trust weighted signed network, each member rate other members in a scale of -10 to 10 for trust. The network includes 3.783 nodes and 24.186 edges. The attributes of the network are source id, target id, rating, and time (Kumar, Spezzano, Subrahmanian & Faloutsos, 2016).

All the datasets are divided half as a train set and half as a test set, the node and edge numbers are given in the Table 3.1. For all data sets source, target and timestamp attributes are used. Timestamps are converted to date before separation of the data to clear the beginning and ending range of the train and test sets. An example of conversion code block is given in Figure 3.3.

Table 3.1 Total number of nodes and edges in train and test sets

| **Dataset** | **Train set** | | **Test set** | |
|---|---|---|---|---|
| | # of Nodes | # of Edges | # of Nodes | # of Edges |
| Reddit | 14.584 | 60.030 | 14.584 | 59.475 |
| Stack Overflow | 7.263 | 68.585 | 7.263 | 38.978 |
| Math Overflow | 4.444 | 76.521 | 4.444 | 36.983 |
| Super User | 10.903 | 52.860 | 10.903 | 23.055 |
| Email-Eu-core | 793 | 10.705 | 793 | 10.589 |
| Bitcoin Alpha | 582 | 582 | 2839 | 1561 |

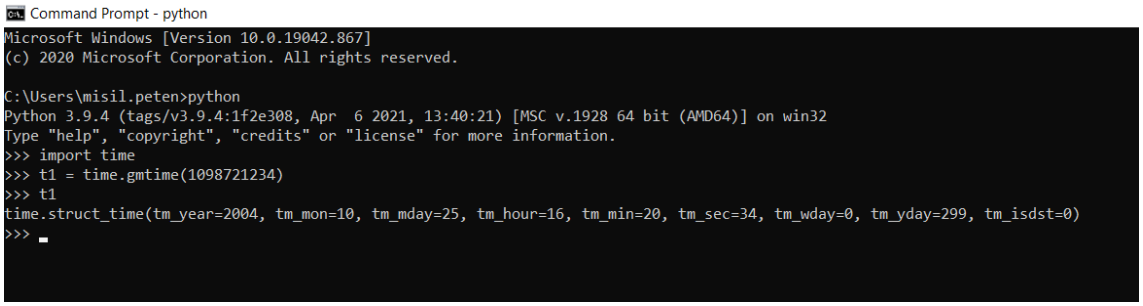


Figure 3.3 The code used of conversion of timestamp to date

### 3.3 Data Processing

Temporal link prediction is a time related activity; therefore, datasets with timestamps have been used, the entire data was separated as training and test sets. A small example for construction steps of training and test sets is given in the Fig. 3.4 and Fig. 3.5 The metrics predict the potential links that will be added from time ***t*** to a given future time ***t'*** (Tylenda et al., 2009). Based on this time interval operation, Fig.3.4 shows the separation of initial training set (on the left) and test set (on the right). The final training set is constructed by focusing the giant component, which is the largest cluster of connected nodes. The giant component can be seen in Figure 3.5 (on the left). The nodes that are in the test set but not in the train set have been eliminated and then final test set is obtained (the right panel of Fig. 3.5).

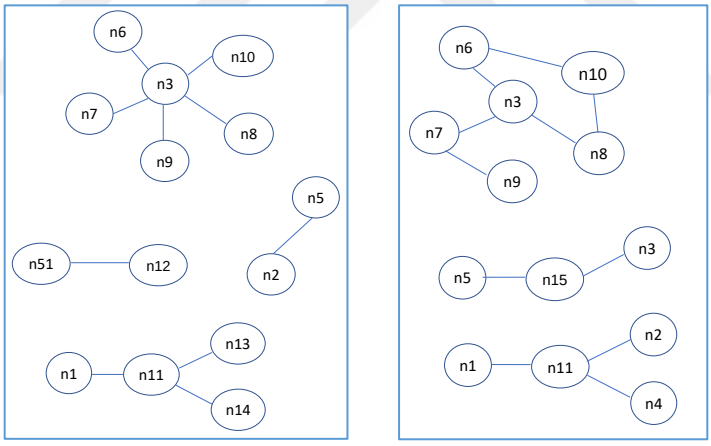


Figure 3.4 Initial training set (on the left) and test set (on the right)

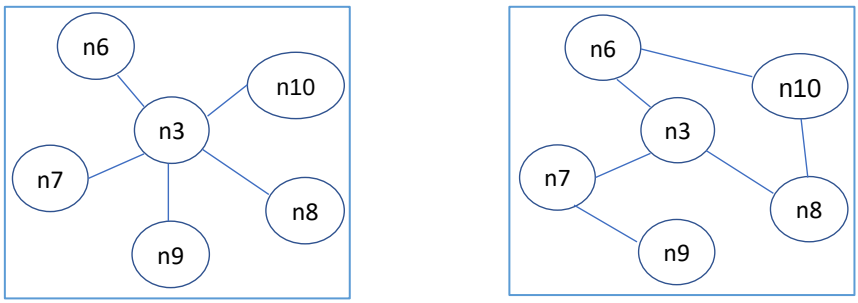


Figure 3.5 The final training set with giant component (on the left) and final test set (on the right)

Train and test set networks are created as graphs with the help of Network X Python Library. Positive predictions and the existing links in the test set are stored in dictionaries.

### 3.4 Evaluation Metrics

For every dataset, a ***k*** threshold value is set, which is 10% of all of edges covered in the test set. Predictions are labeled as positive and negative; link prediction is regarded as a binary classification problem, the class label is specified by the existence of links (Leskovec et al., 2021). If two nodes have a connection in the final test set, we label them as positive, if there is no connection occurred between two nodes the edge label will be negative. If there is a link between two nodes in the training set (giant component) and the connection still remains in the final test set, the edge label will be positive, and negative otherwise. True positive reflects the number of node pairs that links are correctly identified as positive. False positive shows the number of node pairs that are incorrectly classified as positive. False negative shows the number of node pairs that links are incorrectly recognized as negative. True negative reflects the number of node pairs that are negative and truly predicted as negative (Shan, Li, Zhang & Chen, 2020).

Precision, Recall and F1-Score have been used to measure the performance of each neighbor-based metric.

Precision shows us how many selected items are relevant.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \tag{3.6}$$

Recall shows us how many relevant items are selected.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \tag{3.7}$$

F1- Score is a combination of Precision and recall values and it show us the accuracy.

$$\text{F1. Score} = \frac{2 * (\text{Precision} * \text{Recall})}{\text{Precision + Recall}} \tag{3.8}$$

## 3.5 Evaluation

### *3.5.1 Strategy*

The experiments that have been conducted to evaluate the selected methods consist of three major stages:

- Input data preparation
- Method execution
- Result report generation

Input data preparation is the stage where the graph that represents the network is created from the input data sets. The method execution phase consists of running the method realization codes that are fed with the graph, which has been prepared in the previous phase. Finally, result report generation includes saving the evaluation metrics that have been recorded during and after the method execution phase. This stage, which incorporates the steps to generate the result reports, calculates the evaluation metrics that have been discussed in Subsection 3.4 using the data recorded during the execution of the

prediction methods, formats the output, creates visualized presentations, and saves the result in a separate output file.

Processing the graphs that have been created using real-world data sets pose a significant challenge due to the large amount of data that has been stored in the data sets. As presented in Subsection 3.2 the graphs include thousands of nodes and tens of thousands of edges. Therefore, trying to store all possible node pairs, which are candidates to be connected in the next state of the graph, grow exponentially and result in memory insufficiency very quickly. In order to overcome this problem, all candidate pairs have not been created at once but rather been processed using chunks of data, which enables freeing the memory portion used for a chunk after the chunk has been processed. By this approach the candidates have been processed chunk by chunk, growing the processed portion constantly until all is processed, obtaining the results for all possibilities without facing a memory problem.

### *3.5.2 Implementation*

Experiments have been implemented in the Python programming language using PyCharm version 2019.3.3 Community edition (PyCharm, The Python IDE for Professional Developers).

Since we are dealing with large graphs, it is essential to perform to computations with optimized methods. Also, another concern would be ensuring that the methods work effectively and efficiently. It would require a great deal of time and effort to develop such methods, therefore, we have preferred to use third party libraries in order to avoid such costs. We have chosen to use the NetworkX Python Library and Pandas Python Data Analysis Library (Network X, Network Analysis in Python; Pandas, Python Data Analysis Library). Both of the libraries are well known and open source.

Network x Python library is a useful helper package especially for complex networks. The version used is NetworkX 2.4. It is used for reading networks, creating graphs for train and test sets and for the evaluation of five methods explained in 3.1.

Pandas Python Data Analysis Library is beneficial for data structures and data analysis. The version 1.2.4 is used for storing the results of the methods and writing the results to excel.

Code file consists of mainly three segments. The first portion is responsible for reading the input datasets and creating the corresponding graphs. The NetworkX library methods have been utilized for this purpose as can be seen in Fig 3.6. Datasets for the train and test graphs are processed separately.

```
print("Reading input networks...", end=' ', flush=True)

# Training Network
trainingNetwork = nx.read_edgelist("stackoverflow_train.txt", create_using=nx.Graph())
conCompsInTraining = sorted(nx.connected_components(trainingNetwork), key=len, reverse=True)
trainingSet = trainingNetwork.subgraph(conCompsInTraining[0]).copy()

# Test Network
testNetwork = nx.read_edgelist("stackoverflow_test.txt", create_using=nx.Graph())
testSet = testNetwork.copy()
```

Fig 3.6 Reading the inputs and separation of the graph as training-test sets

Once the graphs are created, the isolated nodes are removed from the graphs in order to obtain the giant component. Since the main objective of the experiments are checking the success of the prediction methods on determining the addition of new links among existing nodes it is necessary to ensure that the train and test graphs contains the same set of nodes. To achieve this condition nodes that are exclusive to a single graph are removed from the graphs. The corresponding code part is given in Figure 3.7. This step concludes the first segment.

```
# Eliminate nodes that are not common to both networks
trainingSet.remove_nodes_from(n for n in trainingNetwork if n not in testSet)
testSet.remove_nodes_from(n for n in testNetwork if n not in trainingSet)

# Islands may come into existence in the trainingSet after the above step (e.g., when a node that exists in the
# testSet but not in the trainingSet -i.e., a node that had been added over time- has been removed from the
# trainingSet). Such cases destroy the assumption of giant component being connected, thus, it's necessary to make
# sure that no such case arises. If such a case has been detected, then the trainingSet and the testSet must be
# re-computed to obey this requirement.
conCompsInTraining = sorted(nx.connected_components(trainingSet), key=len, reverse=True)
giantComponent = trainingSet.subgraph(conCompsInTraining[0]).copy()

if len(trainingSet.nodes) != len(giantComponent.nodes):
    trainingSet = giantComponent
    testSet.remove_nodes_from(n for n in testNetwork if n not in trainingSet)

print("Done!", flush=True)

print("TRAIN SET: Nodes: " + str(len(trainingSet.nodes)) + ", Edges: " + str(len(trainingSet.edges)), flush=True)
print("TEST SET : Nodes: " + str(len(testSet.nodes)) + ", Edges: " + str(len(testSet.edges)), flush=True)
```

Figure 3.7 Finding the giant component

Second segment aims to realize the experimentation framework for the methods that will be evaluated. Here the implementation for the framework for the Common Neighborhood evaluation will be discussed, implementation of the remainder frameworks is similar in nature and the reader can refer to Appendix 1 for the full implementation code.

To determine the predictions of the common neighborhood method first number of common neighbors must be computed for each possible pair that can be extracted from the graph. A nested loop structure has been used to form these possible pairs (Fig. 3.8).

```
for i in range(len(setOfNodes)):
    for j in range(i+1, len(setOfNodes)):
        chunkSetOfPossibleLinks.append((setOfNodes[i], setOfNodes[j]))
```

Figure 3.8 Generation of the possible node pairs

If a graph has *n* nodes then the number of possible node pairs, which is equivalent to choosing all subsets of the set of nodes where each subset cardinality is equal to exactly two, can be calculated with the following formula:

$$\binom{n}{2} = \frac{n!}{2!\,(n-2)!} \tag{3.9}$$

Therefore, the number of potential pairs will be very large since it will be in the order of a factorial function. Due to this, it is not feasible to create all pairs at once and process them altogether. In order to achieve a feasible solution node pairs are created and processed chunk by chunk, where each chunk consists of 1000 pairs (Fig. 3.9).

```
for i in range(len(setOfNodes)):
    for j in range(i+1, len(setOfNodes)):
        chunkSetOfPossibleLinks.append((setOfNodes[i], setOfNodes[j]))
        if len(chunkSetOfPossibleLinks) == chunkSize:
            chunkNumberOfCommons = [(e[0], e[1], len(list(nx.common_neighbors(trainingSet, e[0], e[1])))) for e in
                                    chunkSetOfPossibleLinks]
```

Figure 3.9 Dividing data into chunks

After a chunk is created and filled with pairs, the number of common neighbors for the nodes in the pairs are computed for each pair in the chunk. Next, these calculated values are checked against the best values that have been encountered so far. Best candidates among the combined list of values are chosen and the remaining values are discarded (Fig. 3.10). With this approach only the best candidates (i.e., the pairs that will be included in the predictions of the evaluated method) are kept in the memory, which uses the memory efficiently and prevents memory insufficiency problems. With each step the portion of the processed pairs grow and results reflect the best candidates among the cumulative graph portion processed.

```
# Combine the top nodes with maximum common neighbors found so far with the ones
# computed for the current chunk
combined = topNumberOfCommons + chunkNumberOfCommons

# Choose the new top nodes with maximum common neighbors
sortedCombined = sorted(combined, key=lambda x: x[2], reverse=True)
topNumberOfCommons = sortedCombined[0:chunkSize]

# Empty the set of possible links so that a fresh start can be done for the next chunk
chunkSetOfPossibleLinks = []
```

Figure 3.10 Choosing the best candidates

Final code segment includes the computation of the metrics that will be used in the evaluation of the methods and generating a combined report of all metrics and visualization data for each of the methods evaluated. To achieve this goal, metrics are calculated right after a method is executed. Three types of metrics (i.e., Precision, Recall, F1-Score) have been computed, as mentioned in Subsection 3.4. In order to reach a more accurate view these metrics have been calculated for a series of number of picks ranging from 1 to a predefined value, and all of these metrics are included in the result report (Fig. 3.11).

```
for k in range(1, maxNoOfPicks+1):
    # Consider only the top-k predictions
    pickedPredictions = topNumberOfCommons[0:k]

    # Reformat for the comparisons
    positivePredictions = []
    for i in range(len(pickedPredictions)):
        data = str(pickedPredictions[i][0]) + str(separators[0]) + str(pickedPredictions[i][1])
        positivePredictions.append(data)

    # Compute the accuracy for the predictions (for top numberOfTopLinks links)
    results = computeAccuracy(positivePredictions, k, actualPositives)

    cnPrecisionValues.append(results[0])
    cnRecallValues.append(results[1])
    cnF1ScoreValues.append(results[2])
    cnTPValues.append(results[3])
    cnFPValues.append(results[4])
    cnFNValues.append(results[5])
```

Figure 3.11 Calculating the metrics

Metric calculations are performed by code encapsulated as a function, which is presented in Figure 3.12.

Final segment is devoted to a report generation. Data gathered from the evaluation of methods are combined and presented in a user-friendly way with the aid of visual representations. The collected data is presented in an Excel file where columns denote the methods evaluated and rows include the performance values for the respective methods. A sample from the aforementioned file is given in Figure 3.13.

```
# ##############################################################################################
# The function that will compute the accuracy metrics for the given prediction lists
# ##############################################################################################

# posPredictionsDict : dictionary that includes the 'positive' predictions made by the method that will be evaluated
# topK               : number of predictions that will be considered while measuring the accuracy
# actualPosDict      : dictionary that includes the existing links in the test set

def computeAccuracy(posPredictions, topK, actualPos):
    tp = 0
    fp = 0
    fn = 0

    # first check positive (true or false) predictions
    for i in range(topK):
        # if the method has predicted a link to be positive and the test set actually includes that link
        if posPredictions[i] in actualPos:
            tp += 1
        # if the method has predicted a link to be positive, however, the test set does NOT actually include that link
        else:
            fp += 1

    # next check false negative predictions
    fn = (len(actualPos) / 2) - tp

    accuracy = []

    # Calculate precision
    precision = tp / (tp + fp)
    accuracy.append(precision)

    # Calculate recall
    recall = tp / (tp + fn)
    accuracy.append(recall)

    # Calculate F1 Score
    f1score = 2 * (precision * recall) / (precision + recall) if (precision + recall) != 0 else 0
    accuracy.append(f1score)

    accuracy.append(tp)
    accuracy.append(fp)
    accuracy.append(fn)

    return accuracy
```

Figure 3.12 Function that calculates the evaluation metrics

| | CN | JC | AA | PA | PM |
|---|---|---|---|---|---|
| **161** | 0,900621 | 0,012422 | 0,906832 | 0,919255 | 0,012422 |
| **162** | 0,901235 | 0,012346 | 0,907407 | 0,919753 | 0,012346 |
| **163** | 0,90184 | 0,01227 | 0,907975 | 0,920245 | 0,01227 |
| **164** | 0,902439 | 0,012195 | 0,902439 | 0,920732 | 0,012195 |
| **165** | 0,90303 | 0,012121 | 0,90303 | 0,921212 | 0,012121 |
| **166** | 0,903614 | 0,012048 | 0,903614 | 0,915663 | 0,012048 |
| **167** | 0,904192 | 0,011976 | 0,904192 | 0,916168 | 0,011976 |
| **168** | 0,89881 | 0,011905 | 0,904762 | 0,916667 | 0,011905 |
| **169** | 0,899408 | 0,011834 | 0,905325 | 0,91716 | 0,011834 |
| **170** | 0,9 | 0,011765 | 0,905882 | 0,911765 | 0,011765 |
| **171** | 0,900585 | 0,011696 | 0,906433 | 0,912281 | 0,011696 |
| **172** | 0,901163 | 0,011628 | 0,906977 | 0,912791 | 0,011628 |
| **173** | 0,901734 | 0,011561 | 0,907514 | 0,913295 | 0,011561 |
| **174** | 0,902299 | 0,011494 | 0,908046 | 0,913793 | 0,011494 |
| **175** | 0,902857 | 0,011429 | 0,908571 | 0,914286 | 0,011429 |

Figure 3.13 Example Precision values for the Reddit dataset

Result presentation is also supported with visual representations, two example charts are presented in Figure 3.14 and Figure 3.15.

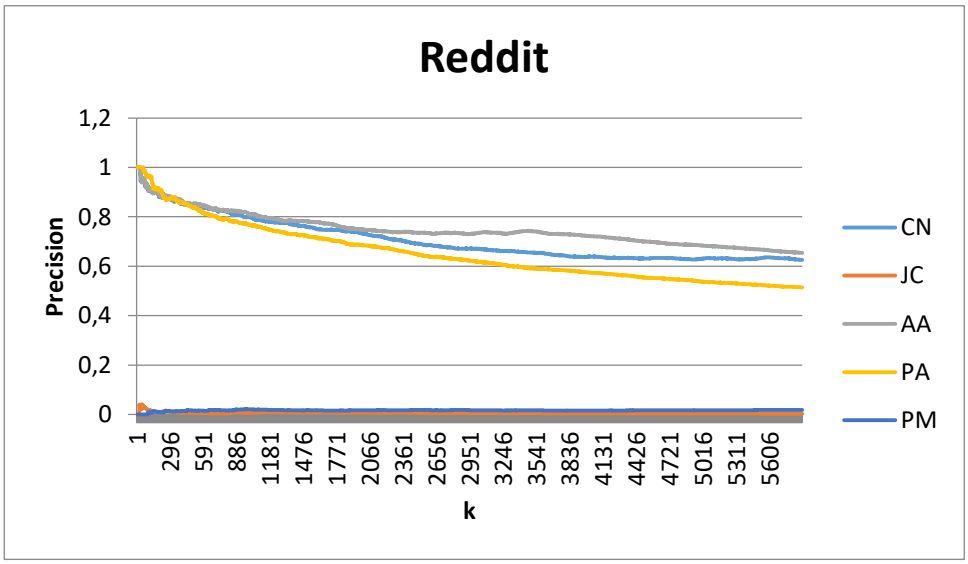


Figure 3.14 Visual chart of the precision results for the Reddit data set

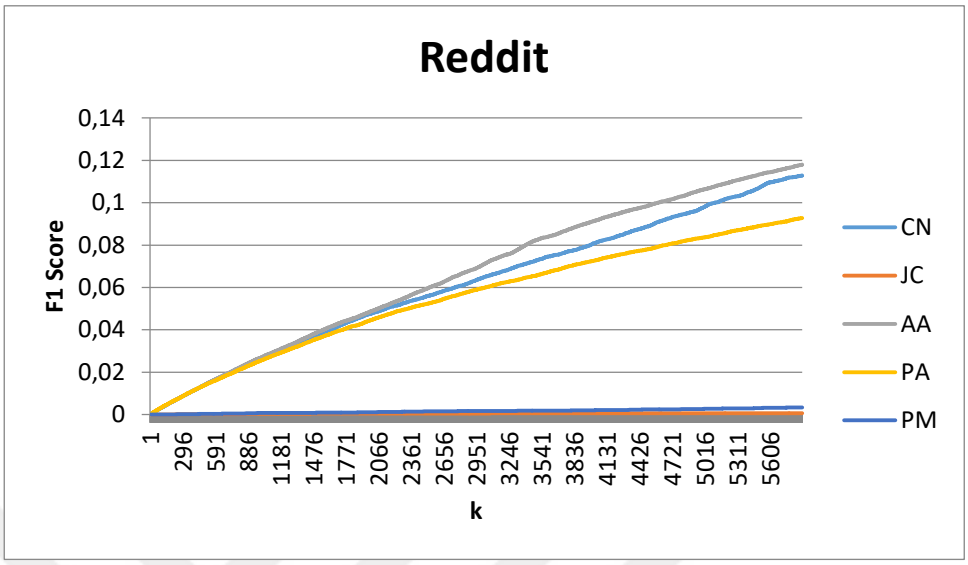


Figure 3.15 Visual chart of the F1-Score results for the Reddit data set

### *3.5.3 Execution Setup*

All the experiments conducted on a computer with CPU Intel® Core™ i7-8750H, 16GB RAM and Windows 10 Home operating system. Datasets are divided 50% - 50% for training and test sets. The Python program is executed from the command line and program is use its messages to the command window (Fig 3.16).

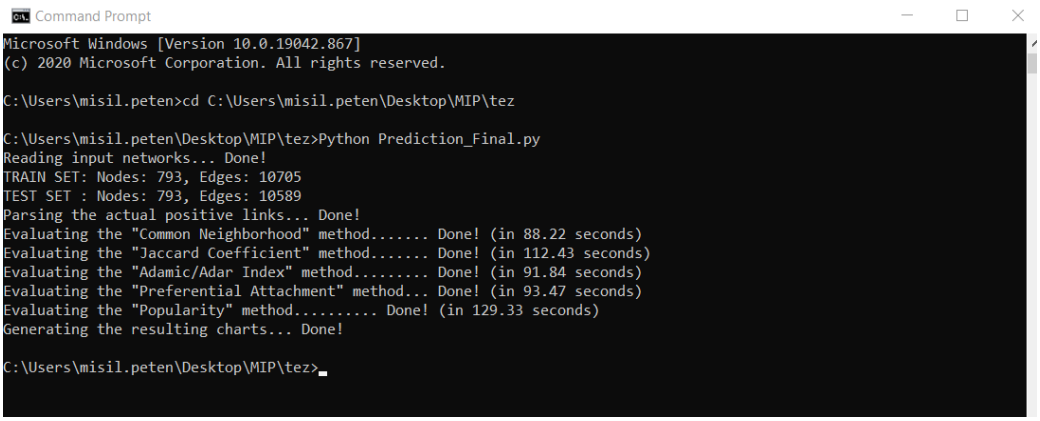


Figure 3.16 An example run

# CHAPTER FOUR
# RESULTS AND DISCUSSION

In this section, first results of the measured metrics will be given as graphs and tables. All methods and results will be compared to each other and best performance metrics and methods will be revealed. Then, the reasons behind this success will be discussed.

## 4.1 Results

During the experiments three evaluation metrics have been measured: Precision, Recall, and F1-Score. As discussed in Subsection 3.4, these metrics have been computed for all values in the range 1 to ***k***, thus, k-many metrics have been measured for each data set. Figures 4.1, 4.2, and 4.3 depict all computed values for the metrics precision, recall, and F1-score, respectively.

The charts indicate that all of the methods have achieved their best performances for the precision metrics. It can be seen that three methods (i.e., PA, AA, and CN) stand out for all of the datasets. JC and PM generally got lower results, however, JC is the third best method for email-eu-core dataset.

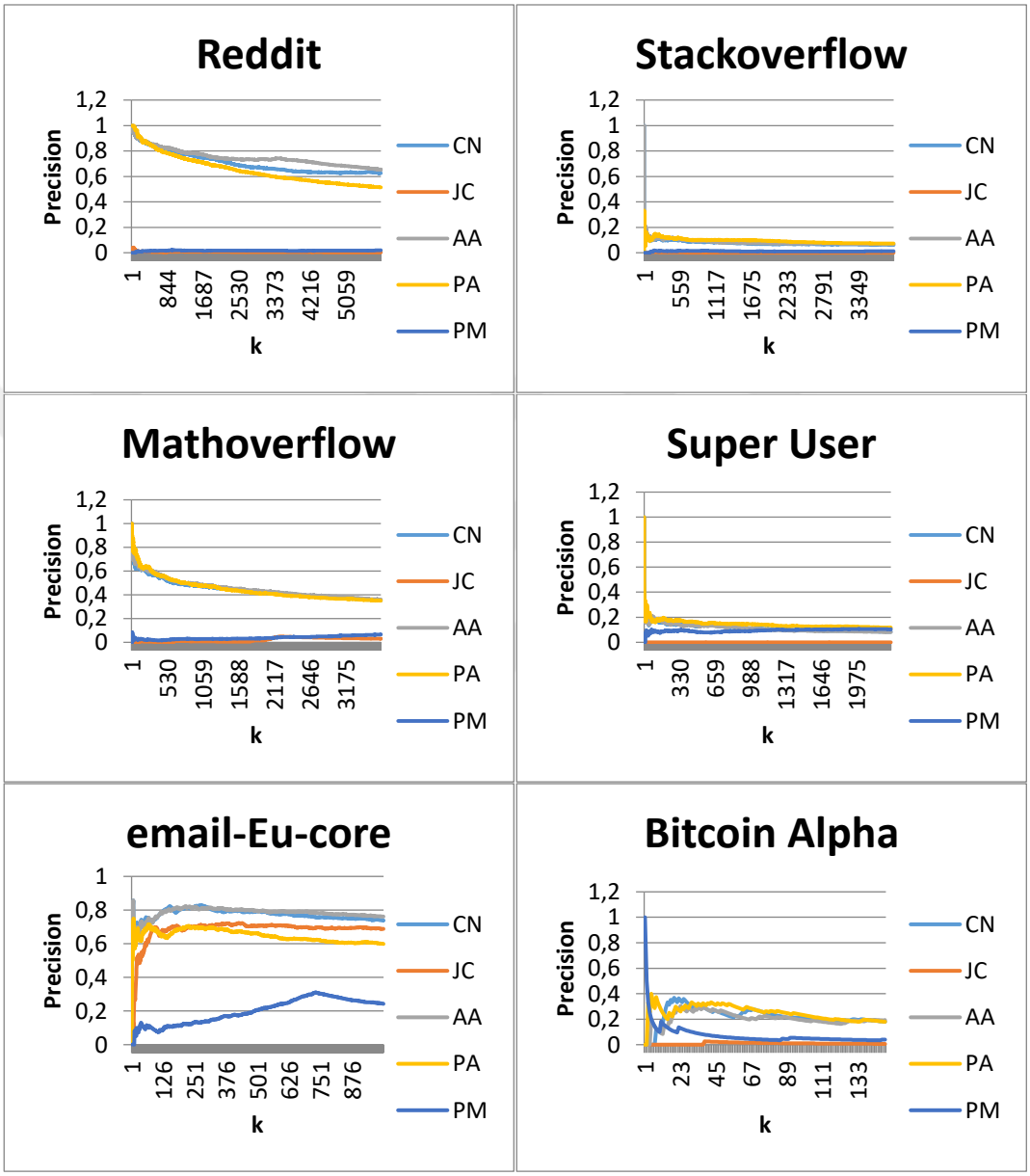


Figure 4.1 The results evaluated by Precision

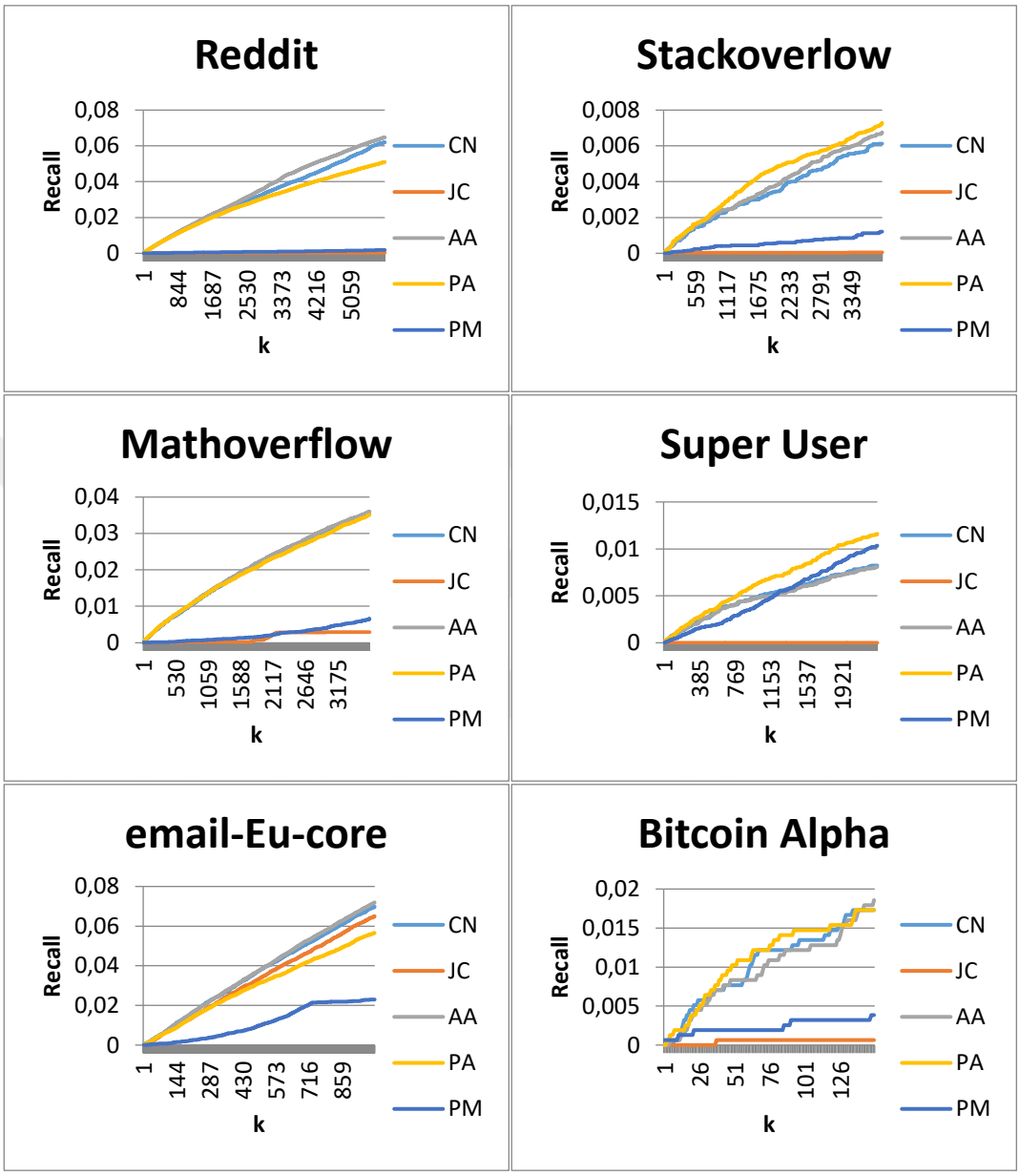


Figure 4.2 The results evaluated by Recall

Recall metrics values indicate that methods can be categorized into three groups with respect to their performances. The first group consists of the methods AA, CN, and PA which usually outperform the other methods. Second group includes only the PM method which performs generally worse than the methods in the first group, however, better than JC. Worst performing method is JC, however, it must be noted that its performance

fluctuates significantly, for instance it is the third best performing method in the Email-eu-core data set. However by far the worst method in the Stack overflow.

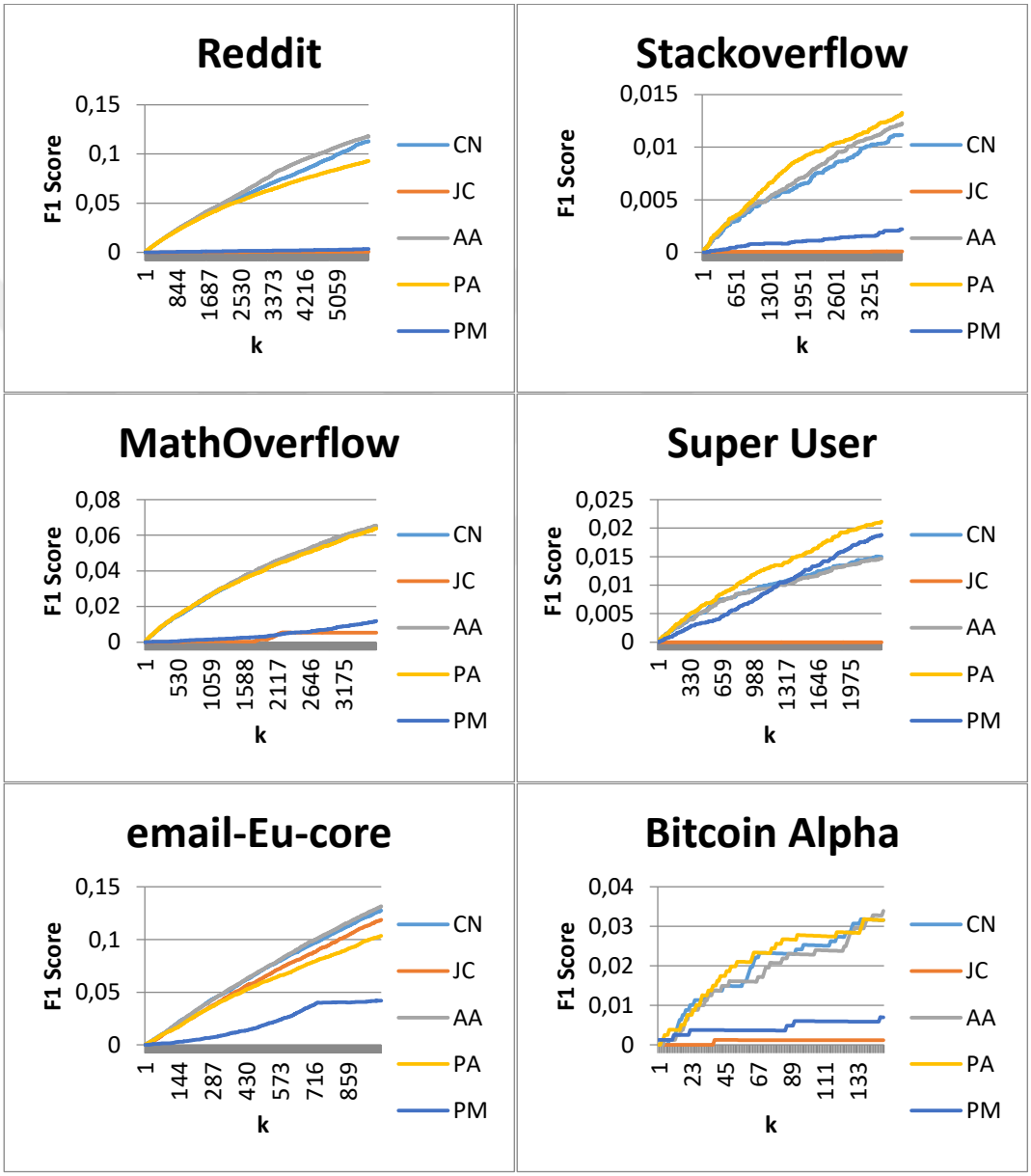


Figure 4.3 The results evaluated by F1-score

The charts in Fig 4.3 indicate that CN, PA, and AA methods depict similar performances, where they usually outperform the JC and PM. The best performing

method varies from dataset to dataset, hence, a specific method cannot be named. Differences between the performances of the methods also vary from dataset to dataset. For instance, the difference among the performances in the Reddit dataset is more significant than the difference in the e-mail-eu core dataset.

Results provide a detailed view of how prediction methods perform on these datasets and enable deriving several simple metrics (e.g., precision), as well as more complex metrics (e.g., Area under curve) for evaluation. In the scope of this study all three chosen metrics require recordings at designated points. To achieve a more accurate evaluation, the values at the points where ***k*** has its maximum value, hence converges to a larger coverage, have been chosen. These values are presented in the Tables 4.1, 4.2, and 4.3, for the evaluation metrics precision, recall, and F1-Score, respectively.

Table 4.1 Precision values at the points where *k* reaches the maximum value

| | **Common Neighborhood** | **Jaccard Coefficient** | **Adamic/Adar Index** | **Preferential Attachment** | **Popularity Method** |
|---|---|---|---|---|---|
| **Reddit** | 0.6252 | 0.003 | 0.0654 | 0.5138 | 0.0186 |
| **Stack Overflow** | 0.0612 | 0.0051 | 0.0674 | 0.0728 | 0.0121 |
| **Math Overflow** | 0.3521 | 0.0289 | 0.3601 | 0.3509 | 0.0656 |
| **Super User** | 0.8260 | 0 | 0.8087 | 0.1165 | 0.1039 |
| **Email-Eu-core** | 0.7391 | 0.6881 | 0.7620 | 0.5997 | 0.2431 |
| **Bitcoin Alpha** | 0.1813 | 0.0066 | 0.1933 | 0.1812 | 0.0403 |

Some methods achieve full performance on several datasets, for instance, CN and AA for Reddit and Math Overflow. PM is generally outperformed by other methods except for the Bitcoin Alpha dataset. PA, CN, and AA generally perform better than JC.

Table 4.2 Recall values at the points where *k* reaches the maximum value

| | Common Neighborhood | Jaccard Coefficient | Adamic/Adar Index | Preferential Attachment | Popularity Method |
|---|---|---|---|---|---|
| **Reddit** | 0.0620 | 0.0003 | 0.0648 | 0.0509 | 0.0018 |
| **Stack Overflow** | 0.0061 | 0.0001 | 0.0067 | 0.0073 | 0.0012 |
| **Math Overflow** | 0.0352 | 0.0029 | 0.0360 | 0.0351 | 0.0066 |
| **Super User** | 0.0082 | 0 | 0.0081 | 0.0116 | 0.0104 |
| **Email-Eu-core** | 0.0697 | 0.0649 | 0.0719 | 0.0566 | 0.0229 |
| **Bitcoin Alpha** | 0.0173 | 0.0006 | 0.0185 | 0.0172 | 0.0038 |

Results indicate that performances of the methods for the Recall metric show a resemblance to the Precision metric. For instance CN, AA, and PA generally outperform JC. However, results for the Recall metric are significantly lower than the Precision metric. For instance no method could achieve the maximum performance for any of the datasets.

Table 4.3 F1-Score values at the points where *k* reaches the maximum value

| | Common Neighborhood | Jaccard Coefficient | Adamic/Adar Index | Preferential Attachment | Popularity Method |
|---|---|---|---|---|---|
| **Reddit** | 0.1129 | 0.0006 | 0.1180 | 0.0927 | 0.0034 |
| **Stack Overflow** | 0.0111 | 0.0001 | 0.0123 | 0.0132 | 0.0022 |
| **Math Overflow** | 0.0640 | 0.0053 | 0.0655 | 0.0638 | 0.0119 |
| **Super User** | 0.0149 | 0 | 0.0146 | 0.0211 | 0.0188 |
| **Email-Eu-core** | 0.1275 | 0.1187 | 0.1315 | 0.1033 | 0.0419 |
| **Bitcoin Alpha** | 0.0316 | 0.0012 | 0.0339 | 0.0315 | 0.0070 |

Performances of the methods are closer to each other for F1-Score values. The results of F1-Score are higher than recall metric and lower than precision metric. AA is the most successful method followed by JC on dataset Super User.

Another analysis was performed for the total completion (i.e., running) time of methods. As it can be seen in Table 4.4, the Popularity method is the fastest one for Bitcoin Alpha and Super user datasets; Common Neighborhood is the fastest method for Math overflow and email-Eu-core datasets; Preferential Attachment is the fastest for Reddit and Stack overflow datasets.

Table 4.4 Total running time (in seconds) of the methods

| | **Common Neighborhood** | **Jaccard Coefficient** | **Adamic/Adar Index** | **Preferential Attachment** | **Popularity Method** |
|---|---|---|---|---|---|
| **Reddit** | 16052.35 | 33244.03 | 15400.69 | 15349.98 | 30809.00 |
| **Stack Overflow** | 9055.46 | 10591.01 | 9096.49 | 8452.78 | 9121.55 |
| **Math Overflow** | 5904.55 | 8412.74 | 6359.45 | 8724.07 | 8924.60 |
| **Super User** | 2211.78 | 2675.75 | 2226.40 | 1523.56 | 1484.36 |
| **Email-Eu-core** | 99.35 | 123.96 | 100.77 | 102.79 | 141.72 |
| **Bitcoin Alpha** | 2.58 | 3.25 | 2.83 | 0.83 | 0.53 |

## 4.5 Discussion

An analysis of the precision metric reveals that the AA method outperforms the JC method in all of social networks that can be classified as untargeted social networks (i.e., datasets Reddit through Super User). Note that the Reddit, Stack overflow, math overflow, and super user data sets are regarded as untargeted since the posts in these networks do not specifically target an individual, whereas email-eu core dataset is regarded to be targeted since an email is sent with an explicit set of specifically targeted users. Recall that node pairs with a higher normalized value receive a higher similarity index in JC, hence, are more likely to be picked in the predictions. However, a higher ratio of number of existing common neighbors to all existing neighbors can indicate a sense of satisfaction, since it means the user has already posted several times in the topics

the user is interested in and it is unlikely that the same user would post in the same topics again and again.

On the other hand, similar node pairs in AA (i.e., pairs with a higher normalized value) receive a lower similarity index, thus, are less likely to be picked. Rather, node pairs with a lower ratio are more likely to be picked. Since a lower ratio means the user has not posted many times on these topics yet, it still has room for growth (i.e., the user can grow interest in these topics throughout time). In these cases, AA bets on the probability that users will grow and shift their interests in different topics, whereas JC assumes the users will keep posting on the domain they already have experience.

Results indicate that user behavior fits AA's ideas more than JC's ideas and tend to expend their territories to new regions. However, when the results for the eu-email-core dataset are checked it can be observed that there is no significant difference in the performances of these two methods. It can be explained with the following circumstances. Members of a large institution are less dynamic than users of an online forum, and timespan of projects may range over long periods of time. Therefore, nodes connected to similar edges are likely to be linked (i.e., people working on the same projects or departments send emails to each other) again and again, so it is expected that JC method would perform well. However, some projects can end and some people can move to different projects or divisions. Such cases allow new links to be formed among people that did not have much in common before. Therefore, it is also expected that AA method would perform as well.

When the results for the recall metric are analyzed, the most striking feature is the notably low results achieved by the JC method. JC performed very low, even close to 0 for all the datasets except for the email-Eu-core dataset. This situation indicates that JC does not perform well when the false negative predictions are included in the metric calculations (because JC makes lots of false negative predictions.).

Methods that have performed relatively well according to the recall metric obtained better results as ***k*** grew. However, remaining methods' performance do not significantly improve with larger ***k*** values. Hence, it would be a better choice to choose CN, AA, or PA with larger designated ***k*** values whenever the recall metric will be used (i.e., false negatives will figure in metric calculations) to evaluate the performance of prediction of the next state of a network.

It is not surprising to observe similar results when the F1-Score metric is taken into consideration. This is largely due to the fact that the recall metric score figures in the calculations for the F1-Score metric, hence, has a direct impact on the performance of the methods with the F1-Score metric as well. The methods CN, AA, and PA generally obtain better scores in the F1-Score metric. Also, larger ***k*** values provide better results.

Datasets that have been used in the experiments vary in size. There are large datasets that include hundreds of thousands of transactions (e.g., the reddit dataset) and form graphs that have thousands of nodes and tens of thousands of edges, as well as relatively smaller datasets that include only thousands of transactions (e.g., the bitcoin dataset) and form smaller graphs. Results indicate that methods exhibit generally consistent behavior in terms of evaluation metrics. Thus, it can be stated that all methods are scalable (i.e., provide similar performances in both large and small networks).

It is obvious that regardless of the metric chosen, results on the datasets Reddit and email-Eu-core are better than the other datasets. These networks differ in size, however, the nodes in their graphs tend to form new links, since messages in these networks tend to get quick responses. For instance, when a reddit user posts something, the user specifies a certain group, by choosing a relevant topic, to specify who will read and answer. Other users who have already following these topics have a higher probability to answer this post due to the experience they gained. The email-Eu-core network represents email transactions and for each recipient of the email a separate connection is created, hence the probability of getting an answer in a short time is high. When all the metrics are taken

into account one can observe that the CN and AA methods are quite successful in the Reddit and email-Eu-core networks. These methods focus on the number of common neighbors between two given nodes. So, their better performance on these networks show that the future connections are accumulated on specific nodes which have triangle relationship with each other.

Finally, when the running time performances of the methods are analyzed it can be seen that different methods finish first for different datasets. For instance, CN method has the fastest score for the Math overflow dataset, whereas the PM beats all the other methods for the Bitcoin-Alpha dataset. It must be noted that the chunk strategy that has been discussed in Section 3.5 decreases the demand for memory, however, affects runtime performance negatively. Nevertheless, no dramatic performance differences between the methods have been observed. Only one exception exists for this statement, which is the experiment conducted on the reddit dataset where the methods JC and PM perform twice as bad as the remaining methods. However, since majority of the results do not show such differences no method can be stated as the fastest.

# CHAPTER FIVE
# CONCLUSION AND FUTURE WORK

This study addresses the link prediction problem on temporal networks and aims to provide an insight to practitioners on the performance of four existing link prediction methods (i.e., common neighborhood, Adamic Adar, preferential attachment, and Jaccard coefficient) and a novel one (i.e., the popularity method). A number of experiments have been conducted to measure the performance of aforementioned methods using three well-known evaluation metrics, which are precision, recall, and F1-score. Six datasets, which contain real-world data, of varying sizes have been used in the experiments.

Experiments have been designed to measure the effectivity and efficiency of the prediction methods and have been implemented in the Python programming language. Third party tools such as Network X library is used to decrease development time while increasing efficiency and reliability.

Analysis indicate that three methods, CN, AA, and PA, stand out in terms of performance evaluations. Regardless of the nature of the dataset used (e.g., four of them are derived from social networks, one from a financial network, and one from an email network) link prediction methods under evaluation showed similar performances. The novel link prediction method, the popularity method, recorded promising results, however, it is evident that there is still room for improvement since it could not outperform the three methods named above. As for scalability, all of the methods satisfied expectations since they did not exhibit any unpredictable and off the chart behavior on datasets of varying sizes. Runtime performances of all methods, regardless of how well they performed, did not depict significant differences.

All of the methods evaluated in this study fall into the category of topology-based methods as they all rely on node-based information to do their predictions. However, there are various other heuristics such as using path-based information, adopting a

random traversal strategy, or facilitating enhanced tactics like using machine learning algorithms to improve performance. Thus, this study can be extended by adding methods, chosen from the aforementioned categories, to the experiments in order to obtain a larger view on performance statistics of link prediction methods. Such an expansion would provide a better view to the practitioners, hence, be more beneficial.

As discussed before, the datasets that have been used in the experiments contain real-world data and belong to different categories such as social networking and finance. Repeating the experiments designed in this study on other types of datasets (e.g., datasets created from e-commerce or co-authorship networks) would provide a better picture on the performances of methods and can even reveal new information on the characteristics of methods. Thus, it is desirable to extend the range of datasets used in the experiments to include a larger number of input datasets belonging to different domains as future work.

## REFERENCES


Adamic, L.A., & Adar, E. (2003). Friends and neighbors on the web. *Social Networks, 25* (3), 211–230.

Barabasi, A. L., & Jeong, H., & Neda, Z., & Ravasz, E., & Schubert, A., & Vicsek, T. (2002). Evolution of the social network of scientific collaborations. *Physica A, 311* (3), 590–614.

By, R., & Varma, N. S., & Indra, R. (2020). Recommendations in social network using link prediction technique. *2020 International Conference on Smart Electronics and Communication (ICOSEC),* 782-786.

Casteigts, A., & Flocchini, P., & Quattrociocchi, W., & Santoro, N. (2012). Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems, 27* (5), 387–408.

Capocci, A., & Servedio, V. D. P., & Colaiori, F., & Buriol, L. S., & Donato, D., & Leonardi, S., and Caldarelli, G. (2006). Preferential attachment in the growth of social networks: The internet encyclopedia Wikipedia. *Physical Review E, 74* (3).

Divakaran, A., & Mohan, A. (2020). Temporal link prediction: a survey. *New Generation Computing, 38,* 213-258.

Fouss, F., & Pirotte, A., & Renders, J. M., & Saerens, M. (2007). Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering, 19* (3), 355–369.

Gao, F., & Musial, K., & Cooper, C., & Tsoka, S. (2015). Link prediction methods and their accuracy for different social networks and network metrics. *Scientific Programming, 2015* (3), 1-13.

Hasani, M., & Zaki, M.J. (2011). *Social Network Data Analytics*. Boston: Springer Science and Business Media.

Holme, P., & Saramäki, J. (2012). Temporal networks. *Physics Reports, 519* (3), 97–125.

Işık, Z., & Peten, M. (2021). Evaluation of link prediction methods on temporal networks. *Proceedings of Global Conference on Engineering Research (GLOBCER'21),* 412- 421.

Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Societe Vaudoise des Sciences Naturelles, 37* (147), 547–579.

Jeh, G., & Widom, J. (2002). SimRank: a measure of structural-context similarity. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'02),* 538–543.

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika, 18,* 39–43.

Kumar, S., & Hamilton, W.L., & Leskovec, J., & Jurafsky, D. (2018). Community interaction and conflict on the web. *International World Wide Web Conference Committee,* 933-943.

Kumar, S., & Spezzano, F., & Subrahmanian, V.S., & Faloutsos, C. (2016). Edge weight prediction in weighted signed networks. *IEEE 16th International Conference on Data Mining (ICDM),* 221-230.

Kostakos, V. (2009). Temporal graphs. *Physica A: Statistical Mechanics and Its Applications, 388* (6), 1007–1023.

Leicht, E. A., & Holme, P., & Newman, M. E. J. (2006). Vertex similarity in networks. *Physical Review E, 73* (2), 026120.

Leskovec J., & Krevl. A. (2021). *SNAP Datasets: Stanford Large Network Dataset Collection*. Retrieved March 15, 2020, from http://snap.stanford.edu/data.

Liben-Nowell, D., & Kleinberg, J. (2007). The link prediction problem for social networks. *Journal of the American Society for Information Science and Technology, 58* (7), 1019- 1031.

Lichtenwalter, R. N., & Lussier, J. T., & Chawla, N. V. (2010). New perspectives and methods in link prediction. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 243–252.

Lu, L., & Jin, C. H., & Zhou, T. (2009). Similarity index based on local paths for link prediction of complex networks. *Physical Review E, 80,* 046122.

Menon, A. K., & Elkan, C. (2011). Link prediction via matrix factorization. *New Generation Computing 123 ECML PKDD 2011, Lecture Notes in Computer Science, 6912*, 437–452.

*Network X, NetworkAnalysis in Python*. Retrieved February 12, 2019, from, https://networkx.org/.

Niwattanakul, S. & Jatsada S. & Ekkachai N. & Wanapu S. (2013). Using of Jaccard Coefficient for keywords similarity. *Proceedings of the International Conference on Internet Computing and Web Services (ICICWS'13)*, 1.

*Pandas, Python data analysis library*. Retrieved February 12, 2019, from, https://pandas.pydata.org/.

Papadimitriou, A., & Symeonidis, P., & Manolopoulos, Y. (2012). Fast and accurate link prediction in social networking systems. *Journal of Systems and Software, 85* (9), 2119–2132.

Paranjape, A., & Benson, A. R., & Leskovec, J. (2017). Motifs in temporal networks. *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 601-610.

Pech, R., & Hao, D., & Lee, Y.L. &, Yuan Y., & Zhou, T. (2019). Link prediction via linear optimization. *Physica A: Statistical Mechanics and its Applications, 528*, 121319.

Pech, R., & Hao, D., & Pan, L., & Cheng, H., & Zhou, T. (2017). Link prediction via matrix completion. *Europhysics Letters, 117* (3), 38002.

*PyCharm, The Python IDE for professional developers.* Retrieved February 12, 2019, from, https://www.jetbrains.com/pycharm/.

Ravasz, E., & Somera, A. L., & Mongru, D. A., & Oltvai, Z. N., & Barabasi, A. L. (2002). Hierarchical organization of modularity in metabolic networks. *Science, 297* (5586), 1551–1555.

Shan, N., & Li, L., & Zhang, Y., & Bai, S., & Chen, X. (2020). Supervised link prediction in multiplex networks. *Knowledge-Based Systems, 203,* 106168.

Tylenda, T., & Angelova, R., & Bedathur, S. (2009). Towards time-aware link prediction in evolving social networks. *Proceedings of the 3rd ACM Workshop on Social Network Mining and Analysis (SNA-KDD '09),* 1-10.

Valverde-Rebaza, J., & de Andrade Lopes, A. (2013). Exploiting behaviors of communities of twitter users for link prediction. *Social Network Analysis and Mining*, *3* (4), 1063–1074.

Wang, P., & Xu B., & Wu Y. (2015) Link prediction in social networks: the state-of-the-art. *China Information Sciences*, *58,* 1-38.

Wohlfarth, T., Ichise, R. (2008). Semantic and event-based approach for link prediction. *Lecture Notes in Computer Science*, *5345*, 50–61.

Xu, H., & Zhang, L. (2013). Application of link prediction in temporal networks. *Advanced Materials Research, 756-759,* 2231-2236.

Yang, X., & Tian, Z., & Cui, H., & Zhang, Z. (2012). Link prediction on evolving network using tensor-based node similarity. *2012 IEEE 2nd International Conference on Cloud Computing and Intelligent Systems (CCIS),* 154–158.

Yao, L., & Wang, L., & Pan, L., & Yao, K. (2016). Link prediction based on common-neighbors for dynamic social network. *Procedia Computer Science*, *83*, 82–89.

Yin, H., & Benson, A. R., & Leskovec, J., & Gleich, D. F. (2017). Local Higher-order Graph Clustering. *In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 555-564.

Zhou, T., & Lu, L., & Zhang, Y. C. (2009). Predicting missing links via local information. *European Physical Journal B, 71*, 623–630.

## APPENDICES

## APPENDIX 1: FULL IMPLEMENTATION CODE

```
import pandas as pd
from networkx import nx
#import matplotlib.pyplot as plt
#import operator
import time

# The function that will compute the accuracy metrics for the given
prediction lists#
# posPredictionsDict : dictionary that includes the 'positive' predictions
made by the method that will be evaluated
# topK               : number of predictions that will be considered while
measuring the accuracy
# actualPosDict      : dictionary that includes the existing links in the
test set

def computeAccuracy(posPredictions, topK, actualPos):
    tp = 0
    fp = 0
    fn = 0

    # first check positive (true or false) predictions
    for i in range(topK):
        # if the method has predicted a link to be positive and the test
set actually includes that link#
        if posPredictions[i] in actualPos:
            tp += 1
        # if the method has predicted a link to be positive, however, the
test set does NOT actually include that link#
        else:
            fp += 1

    # next check false negative predictions#
    fn = (len(actualPos) / 2) - tp

    accuracy = []

    # Calculate precision
    precision = tp / (tp + fp)
    accuracy.append(precision)

    # Calculate recall
    recall = tp / (tp + fn)
    accuracy.append(recall)

    # Calculate F1 Score
    f1score = 2 * (precision * recall) / (precision + recall) if
(precision + recall) != 0 else 0
    accuracy.append(f1score)
    accuracy.append(tp)
    accuracy.append(fp)
```

```
    accuracy.append(fn)

    return accuracy

# The function that will plot the charts in an excel sheet for the given
computed values

def plotChart(cnVals, jcVals, aaVals, paVals, pmvals, writer, sheetName):
    computedValues = {'CN': cnVals, 'JC': jcVals, 'AA': aaVals, 'PA':
paVals, "PM":pmvals}
    df = pd.DataFrame(computedValues)
    df = df.iloc[1:, ]
    df.to_excel(writer, sheet_name=sheetName)

    workbook = writer.book
    worksheet = writer.sheets[sheetName]
    chart = workbook.add_chart({'type': 'line'})

    rowCount = len(cnVals)
    chart.add_series({'values': '=' + sheetName + '!$B$3:$B$' +
str(rowCount), 'name': 'Com. Neig.'})
    chart.add_series({'values': '=' + sheetName + '!$C$3:$C$' +
str(rowCount), 'name': 'Jac. Coef.'})
    chart.add_series({'values': '=' + sheetName + '!$D$3:$D$' +
str(rowCount), 'name': 'Adamic/Adar'})
    chart.add_series({'values': '=' + sheetName + '!$E$3:$E$' +
str(rowCount), 'name': 'Pref. Att.'})
    chart.add_series({'values': '=' + sheetName + '!$F$3:$F$' +
str(rowCount), 'name': 'Pop. Met.'})

    worksheet.insert_chart('J2', chart)

def plotDetailedChart(precisionVals, tpVals, fpVals, fnVals, header,
writer, sheetName):
    computedValues = {header: precisionVals, 'TP': tpVals, 'FP': fpVals,
'FN': fnVals}
    df = pd.DataFrame(computedValues)
    df = df.iloc[1:, ]
    df.to_excel(writer, sheet_name=sheetName)

    workbook = writer.book
    worksheet = writer.sheets[sheetName]
    chart = workbook.add_chart({'type': 'line'})

    rowCount = len(precisionVals)
    chart.add_series({'values': '=' + sheetName + '!$B$3:$B$' +
str(rowCount), 'name': 'Precision'})

    worksheet.insert_chart('H2', chart)

# Computation and evaluation of the predictions according to the "Common
Neighborhood" method#

def commonNeighborhood():
    print("Evaluating the \"Common Neighborhood\" method..... ", end='',
flush=True)
```

```
    startingTime = time.time()

    # First compute the common neighborhood scores
    i = 0
    j = 0
    chunkSetOfPossibleLinks = []
    topNumberOfCommons = []
    chunkCounter = 0
    for i in range(len(setOfNodes)):
        for j in range(i+1, len(setOfNodes)):
            chunkSetOfPossibleLinks.append((setOfNodes[i], setOfNodes[j]))
            if len(chunkSetOfPossibleLinks) == chunkSize:
                chunkNumberOfCommons = [(e[0], e[1],
len(list(nx.common_neighbors(trainingSet, e[0], e[1])))) for e in
chunkSetOfPossibleLinks]

# Combine the top nodes with maximum common neighbors found so far with
the ones computed for the current chunk
                combined = topNumberOfCommons + chunkNumberOfCommons

# Choose the new top nodes with maximum common neighbors
                sortedCombined = sorted(combined, key=lambda x: x[2],
reverse=True)
                topNumberOfCommons = sortedCombined[0:chunkSize]

# Empty the set of possible links so that a fresh start can be done for
the next chunk
                chunkSetOfPossibleLinks = []

                chunkCounter += 1

# If there's a remaining chunk, smaller than the predetermined size,
process it as well
    if len(chunkSetOfPossibleLinks) > 0:
        chunkNumberOfCommons = [(e[0], e[1],
len(list(nx.common_neighbors(trainingSet, e[0], e[1])))) for e in
chunkSetOfPossibleLinks]

        combined = topNumberOfCommons + chunkNumberOfCommons
        sortedCombined = sorted(combined, key=lambda x: x[2],
reverse=True)
        topNumberOfCommons = sortedCombined[0:chunkSize]

        chunkCounter += 1

# Now that the top common neighborhood scores are computed, evaluate the
method for different number of picks #
    cnPrecisionValues.append(0)
    cnRecallValues.append(0)
    cnF1ScoreValues.append(0)
    cnTPValues.append(0)
    cnFPValues.append(0)
    cnFNValues.append(0)
    for k in range(1, maxNoOfPicks+1):
        # Consider only the top-k predictions
        pickedPredictions = topNumberOfCommons[0:k]
```

```
        # Reformat for the comparisons
        positivePredictions = []
        for i in range(len(pickedPredictions)):
            data = str(pickedPredictions[i][0]) + str(separators[0]) +
str(pickedPredictions[i][1])
            positivePredictions.append(data)

# Compute the accuracy for the predictions (for top numberOfTopLinks
links)
        results = computeAccuracy(positivePredictions, k, actualPositives)

        cnPrecisionValues.append(results[0])
        cnRecallValues.append(results[1])
        cnF1ScoreValues.append(results[2])
        cnTPValues.append(results[3])
        cnFPValues.append(results[4])
        cnFNValues.append(results[5])

    endingTime = time.time()
    elapsedTime = endingTime - startingTime

    print("Done! (in %3.2f seconds)" % (elapsedTime), flush=True)

# Computation and evaluation of the predictions according to the "Jaccard
Coefficient" method

def jaccardCoefficient():
    print("Evaluating the \"Jaccard Coefficient\" method..... ", end='',
flush=True)
    startingTime = time.time()

    # First compute the similarity coefficient scores
    i = 0
    j = 0
    chunkSetOfPossibleLinks = []
    topSimilarityCoefficients = []
    chunkCounter = 0
    for i in range(len(setOfNodes)):
        for j in range(i+1, len(setOfNodes)):
            chunkSetOfPossibleLinks.append((setOfNodes[i], setOfNodes[j]))
            if len(chunkSetOfPossibleLinks) == chunkSize:
                chunkSimilarityCoefficients =
list(nx.jaccard_coefficient(trainingSet, chunkSetOfPossibleLinks))

# Combine the top nodes with maximum similarity coefficients found so far
with the ones computed for the current chunk
                combined = topSimilarityCoefficients +
chunkSimilarityCoefficients

# Choose the new top nodes with maximum similarity coefficients
                sortedCombined = sorted(combined, key=lambda x: x[2],
reverse=True)
                topSimilarityCoefficients = sortedCombined[0:chunkSize]

# Empty the set of possible links so that a fresh start can be done for
the next chunk
                chunkSetOfPossibleLinks = []
```

```
                chunkCounter += 1

# If there's a remaining chunk, smaller than the predetermined size,
process it as well
    if len(chunkSetOfPossibleLinks) > 0:
        chunkSimilarityCoefficients =
list(nx.jaccard_coefficient(trainingSet, chunkSetOfPossibleLinks))

        combined = topSimilarityCoefficients + chunkSimilarityCoefficients
        sortedCombined = sorted(combined, key=lambda x: x[2],
reverse=True)
        topSimilarityCoefficients = sortedCombined[0:chunkSize]

        chunkCounter += 1

    # Now that the top similarity coefficient scores are computed,
evaluate the method for different number of picks
    jcPrecisionValues.append(0)
    jcRecallValues.append(0)
    jcF1ScoreValues.append(0)
    jcTPValues.append(0)
    jcFPValues.append(0)
    jcFNValues.append(0)

    for k in range(1, maxNoOfPicks+1):
        # Consider only the top-k predictions
        pickedPredictions = topSimilarityCoefficients[0:k]

        # Reformat for the comparisons
        positivePredictions = []
        for i in range(len(pickedPredictions)):
            data = str(pickedPredictions[i][0]) + str(separators[0]) +
str(pickedPredictions[i][1])
            positivePredictions.append(data)
# Compute the accuracy for the predictions (for top numberOfTopLinks
links)
        results = computeAccuracy(positivePredictions, k, actualPositives)

        jcPrecisionValues.append(results[0])
        jcRecallValues.append(results[1])
        jcF1ScoreValues.append(results[2])
        jcTPValues.append(results[3])
        jcFPValues.append(results[4])
        jcFNValues.append(results[5])

    endingTime = time.time()
    elapsedTime = endingTime - startingTime

    print("Done! (in %3.2f seconds)" % (elapsedTime), flush=True)

# Computation and evaluation of the predictions according to the
"Adamic/Adar Index" method
def adamicAdarIndex():
    print("Evaluating the \"Adamic/Adar Index\" method..... ", end='',
flush=True)
```

```
    startingTime = time.time()

    # First compute the Adamic/Adar scores
    i = 0
    j = 0
    chunkSetOfPossibleLinks = []
    topAAScores = []
    chunkCounter = 0
    for i in range(len(setOfNodes)):
        for j in range(i+1, len(setOfNodes)):
            chunkSetOfPossibleLinks.append((setOfNodes[i], setOfNodes[j]))
            if len(chunkSetOfPossibleLinks) == chunkSize:
                chunkAAScores = list(nx.adamic_adar_index(trainingSet,
chunkSetOfPossibleLinks))

# Combine the top nodes with maximum AA scores found so far with the
onescomputed for the current chunk
                combined = topAAScores + chunkAAScores

# Choose the new top nodes with maximum AA scores
                sortedCombined = sorted(combined, key=lambda x: x[2],
reverse=True)
                topAAScores = sortedCombined[0:chunkSize]

                # Empty the set of possible links so that a fresh start
can be done for the next chunk
                chunkSetOfPossibleLinks = []

                chunkCounter += 1

# If there's a remaining chunk, smaller than the predetermined size,
process it as well
    if len(chunkSetOfPossibleLinks) > 0:
        chunkAAScores = list(nx.adamic_adar_index(trainingSet,
chunkSetOfPossibleLinks))

        combined = topAAScores + chunkAAScores
        sortedCombined = sorted(combined, key=lambda x: x[2],
reverse=True)
        topAAScores = sortedCombined[0:chunkSize]

        chunkCounter += 1

# Now that the top Adamic/Adar scores are computed, evaluate the method
for different number of picks
    aaPrecisionValues.append(0)
    aaRecallValues.append(0)
    aaF1ScoreValues.append(0)
    aaTPValues.append(0)
    aaFPValues.append(0)
    aaFNValues.append(0)

    for k in range(1, maxNoOfPicks+1):
        # Consider only the top-k predictions
        pickedPredictions = topAAScores[0:k]

        # Reformat for the comparisons
```

```
        positivePredictions = []
        for i in range(len(pickedPredictions)):
            data = str(pickedPredictions[i][0]) + str(separators[0]) +
str(pickedPredictions[i][1])
            positivePredictions.append(data)

# Compute the accuracy for the predictions (for top numberOfTopLinks
links)
        results = computeAccuracy(positivePredictions, k, actualPositives)

        aaPrecisionValues.append(results[0])
        aaRecallValues.append(results[1])
        aaF1ScoreValues.append(results[2])
        aaTPValues.append(results[3])
        aaFPValues.append(results[4])
        aaFNValues.append(results[5])

    endingTime = time.time()
    elapsedTime = endingTime - startingTime

    print("Done! (in %3.2f seconds)" % (elapsedTime), flush=True)

# Computation and evaluation of the predictions according to the
"Preferential Attachment" method
def preferentialAttachment():
    print("Evaluating the \"Preferential Attachment\" method... ", end='',
flush=True)

    startingTime = time.time()

    # First compute the degree scores
    i = 0
    j = 0
    chunkSetOfPossibleLinks = []
    topDegreeScores = []
    chunkCounter = 0
    for i in range(len(setOfNodes)):
        for j in range(i+1, len(setOfNodes)):
            chunkSetOfPossibleLinks.append((setOfNodes[i], setOfNodes[j]))
            if len(chunkSetOfPossibleLinks) == chunkSize:
                chunkDegreeScores =
list(nx.preferential_attachment(trainingSet, chunkSetOfPossibleLinks))

# Combine the top nodes with maximum degree scores found so far with the
ones
# computed for the current chunk
                combined = topDegreeScores + chunkDegreeScores

# Choose the new top nodes with maximum degree scores
                sortedCombined = sorted(combined, key=lambda x: x[2],
reverse=True)
                topDegreeScores = sortedCombined[0:chunkSize]

# Empty the set of possible links so that a fresh start can be done for
the next chunk
                chunkSetOfPossibleLinks = []
```

```
                chunkCounter += 1

# If there's a remaining chunk, smaller than the predetermined size,
process it as well
    if len(chunkSetOfPossibleLinks) > 0:
        chunkDegreeScores = list(nx.preferential_attachment(trainingSet,
chunkSetOfPossibleLinks))

        combined = topDegreeScores + chunkDegreeScores
        sortedCombined = sorted(combined, key=lambda x: x[2],
reverse=True)
        topDegreeScores = sortedCombined[0:chunkSize]

        chunkCounter += 1

# Now that the top degree scores are computed, evaluate the method for
different number of picks
    paPrecisionValues.append(0)
    paRecallValues.append(0)
    paF1ScoreValues.append(0)
    paTPValues.append(0)
    paFPValues.append(0)
    paFNValues.append(0)

    for k in range(1, maxNoOfPicks+1):
        # Consider only the top-k predictions
        pickedPredictions = topDegreeScores[0:k]

        # Reformat for the comparisons
        positivePredictions = []
        for i in range(len(pickedPredictions)):
            data = str(pickedPredictions[i][0]) + str(separators[0]) +
str(pickedPredictions[i][1])
            positivePredictions.append(data)

# Compute the accuracy for the predictions (for top numberOfTopLinks
links)
        results = computeAccuracy(positivePredictions, k, actualPositives)

        paPrecisionValues.append(results[0])
        paRecallValues.append(results[1])
        paF1ScoreValues.append(results[2])
        paTPValues.append(results[3])
        paFPValues.append(results[4])
        paFNValues.append(results[5])

    endingTime = time.time()
    elapsedTime = endingTime - startingTime

    print("Done! (in %3.2f seconds)" % (elapsedTime), flush=True)

# Computation and evaluation of the predictions according to the
"Popularity" method

def popularityMethod():
    print("Evaluating the \"Popularity\" method..... ", end='',
flush=True)
```

```
    startingTime = time.time()

    linkCount = []
    for n in setOfNodes:
        count = trainingSet.degree[n]
        linkCount.append((n, count))

    sortedLinkCount = sorted(linkCount, key=lambda x: x[1], reverse=True)

    length = len(sortedLinkCount)
    combined = []
    for i in range(length-1):
        n1 = sortedLinkCount[i][0]
        c1 = sortedLinkCount[i][1]

        for j in range(i+1, length-1):
            n2 = sortedLinkCount[j][0]
            c2 = sortedLinkCount[j][1]
            diff = abs(c1 - c2)
            combined.append((n1, n2, diff))

    sortedPMScores = sorted(combined, key=lambda x: x[2], reverse=True)
    topPMScores = sortedPMScores[0:chunkSize]

# Now that the top P.M. scores are computed, evaluate the method for
different number of picks

    pmPrecisionValues.append(0)
    pmRecallValues.append(0)
    pmF1ScoreValues.append(0)
    pmTPValues.append(0)
    pmFPValues.append(0)
    pmFNValues.append(0)
    for k in range(1, maxNoOfPicks+1):
        # Consider only the top-k predictions
        pickedPredictions = topPMScores[0:k]

        # Reformat for the comparisons
        positivePredictions = []
        for i in range(len(pickedPredictions)):
            data = str(pickedPredictions[i][0]) + str(separators[0]) +
str(pickedPredictions[i][1])
            positivePredictions.append(data)

# Compute the accuracy for the predictions (for top numberOfTopLinks
links)
        results = computeAccuracy(positivePredictions, k, actualPositives)

        pmPrecisionValues.append(results[0])
        pmRecallValues.append(results[1])
        pmF1ScoreValues.append(results[2])
        pmTPValues.append(results[3])
        pmFPValues.append(results[4])
        pmFNValues.append(results[5])

    endingTime = time.time()
```

```
    elapsedTime = endingTime - startingTime

    print("Done! (in %3.2f seconds)" % (elapsedTime), flush=True)

# EXECUTION
# Reading the input networks

print("Reading input networks...", end=' ', flush=True)

# Training Network
trainingNetwork = nx.read_edgelist("email-Eu_train.txt",
create_using=nx.Graph())
conCompsInTraining = sorted(nx.connected_components(trainingNetwork),
key=len, reverse=True)
trainingSet = trainingNetwork.subgraph(conCompsInTraining[0]).copy()

# Test Network
testNetwork = nx.read_edgelist("email-Eu_test.txt",
create_using=nx.Graph())
testSet = testNetwork.copy()

# Eliminate nodes that are not common to both networks
trainingSet.remove_nodes_from(n for n in trainingNetwork if n not in
testSet)
testSet.remove_nodes_from(n for n in testNetwork if n not in trainingSet)

# Islands may come into existence in the trainingSet after the above step
(e.g., when a node that exists in the testSet but not in the trainingSet -
i.e., a node that had been added over time- has been removed from the
trainingSet). Such cases destroy the assumption of giant component being
connected, thus, it's necessary to make sure that no such case arises. If
such a case has been detected, then the trainingSet and the testSet must
be re-computed to obey this requirement.

conCompsInTraining = sorted(nx.connected_components(trainingSet), key=len,
reverse=True)
giantComponent = trainingSet.subgraph(conCompsInTraining[0]).copy()

if len(trainingSet.nodes) != len(giantComponent.nodes):
    trainingSet = giantComponent
    testSet.remove_nodes_from(n for n in testNetwork if n not in
trainingSet)

print("Done!", flush=True)

print("TRAIN SET: Nodes: " + str(len(trainingSet.nodes)) + ", Edges: " +
str(len(trainingSet.edges)), flush=True)
print("TEST SET : Nodes: " + str(len(testSet.nodes)) + ", Edges: " +
str(len(testSet.edges)), flush=True)

# Computation of the metrics for the training and the test sets

# the set of nodes
setOfNodes = list(trainingSet.nodes())

separators = ['#']
```

```
# change these values as you wish
chunkSize = 17000
maxNoOfPicks = 1000

print("Parsing the actual positive links...", end=' ', flush=True)
# Dictionary that reflects the links in the test set
existingLinksInTestSet = list(testSet.edges)

actualPositives = []
for i in range(len(existingLinksInTestSet)):
    data = str(existingLinksInTestSet[i][0]) + str(separators[0]) +
str(existingLinksInTestSet[i][1])
    actualPositives.append(data)
    data = str(existingLinksInTestSet[i][1]) + str(separators[0]) +
str(existingLinksInTestSet[i][0])
    actualPositives.append(data)

print("Done!", flush=True)

# Running the Methods

cnPrecisionValues = []
cnRecallValues = []
cnF1ScoreValues = []
cnTPValues = []
cnFPValues = []
cnFNValues = []

jcPrecisionValues = []
jcRecallValues = []
jcF1ScoreValues = []
jcTPValues = []
jcFPValues = []
jcFNValues = []

aaPrecisionValues = []
aaRecallValues = []
aaF1ScoreValues = []
aaTPValues = []
aaFPValues = []
aaFNValues = []

paPrecisionValues = []
paRecallValues = []
paF1ScoreValues = []
paTPValues = []
paFPValues = []
paFNValues = []

pmPrecisionValues = []
pmRecallValues = []
pmF1ScoreValues = []
pmTPValues = []
pmFPValues = []
pmFNValues = []

commonNeighborhood()
```

```
jaccardCoefficient()
adamicAdarIndex()
preferentialAttachment()
popularityMethod()

# Generating the resulting charts

print("Generating the resulting charts... ", end='', flush=True)

fileName = "eu-core" + ".xlsx"
writer = pd.ExcelWriter(fileName, engine='xlsxwriter')

plotChart(cnRecallValues,    jcRecallValues,    aaRecallValues,
paRecallValues, pmRecallValues, writer, "Recall")
plotChart(cnPrecisionValues, jcPrecisionValues, aaPrecisionValues,
paPrecisionValues, pmPrecisionValues, writer, "Precision")
plotChart(cnF1ScoreValues,   jcF1ScoreValues,   aaF1ScoreValues,
paF1ScoreValues, pmF1ScoreValues, writer, "F1.score")

plotDetailedChart(cnPrecisionValues, cnTPValues, cnFPValues, cnFNValues,
"CN-Precision", writer, "CN-Precision")
plotDetailedChart(jcPrecisionValues, jcTPValues, jcFPValues, jcFNValues,
"JC-Precision", writer, "JC-Precision")
plotDetailedChart(aaPrecisionValues, aaTPValues, aaFPValues, aaFNValues,
"AA-Precision", writer, "AA-Precision")
plotDetailedChart(paPrecisionValues, paTPValues, paFPValues, paFNValues,
"PA-Precision", writer, "PA-Precision")
plotDetailedChart(pmPrecisionValues, pmTPValues, pmFPValues, pmFNValues,
"PM-Precision", writer, "PM-Precision")

writer.save()

print("Done!", flush=True)
```