CS 512 Project

GitHub: Predicting co-committers

Chia-Hao Hsieh
Hanqing Chen
Le Tang Boon

Abstract—Link Prediction is an example class of tasks under link mining where techniques are developed to “explicitly consider these links when building predictive or descriptive models of the linked data”. Other tasks include object ranking, group detection, collective classification and subgraph discovery. In this project, we present three algorithms for predicting links for GitHub network.

I. INTRODUCTION

Link Prediction is defined to be predicting future existence of links between two entities in a network given current knowledge of the network. This knowledge includes observable links or entities at the current time stamp of the network. Examples include Co-Author Relationship Prediction in Heterogeneous Bibliographic Networks[5], friendship prediction in social network and relationships between connected entities in the university context[4]. Link Prediction may also play a role in finding missing data in social networks.[3] In the area of Internet and web science, it can be used in tasks like automatic web hyper-link creation and web site hyper-link prediction. In e-commerce, one of the most prominent usages of link prediction is to build recommendation systems. It also has various applications in other scientific disciplines. For instance, in bibliography and library science, it can be used for de-duplication and record linkage; in Bioinformatics, it has been used in protein-protein interaction (PPI) prediction or to annotate the PPI graph. In security related applications, it can be used to identify hidden groups of terrorists and criminals. In many of the above applications, the graphs that we work on are not necessarily social network graphs, rather they can be Internet, information networks, biological entity networks, and so on.

As mentioned by Getoor et al., this is a binary classification problem where the link between two entities at time t+1 exists(1) or do not exist(0). Several methods have arisen for this problem. Each type investigates different aspects of the network: structural, attribute and hybrid. Structural properties of the network have been used primitively to derive proximity measures to define the closeness of two entities in order to predict whether a link will exist between them in the future.[6] Attribute properties are exploited for link prediction. For instance, Popescul et al. used logistic regression on the relational features for prediction.[7] Hybrid methods used both attribute and structural features to develop probability models for the network.

One main problem with link prediction is the sparsity of the network. This increases the difficulty for modeling the network and ultimately, evaluating the confidence for making inferences as mentioned by Rattigan.[8] A better approach to improve prediction accuracy is to make collective predictions. Markov Random Fields(MRF) define the joint probabilistic models over the network. They enable probabilistic inferences over the links to capture the correlation between links. However, due to computational costs, exact inferences are intractable and often, approximate inferences are made instead.[2]

II. RELATED WORKS

Similarity based algorithms assign a score to each corresponding pair of nodes in the network. Prediction of non existent links are based on nodes that return higher scores according to ranking. Different metrics are used for measuring similarity. Nodes are determined to be similar if their underlying attributes are similar. However, these attributes are often hidden and thus we have to look at structural similarity of the network instead. Structural similarity can be organized into “local vs. global, parameter-free vs. parameter-dependent, node-dependent vs. path-dependent, and so on. The similarity indices can also be sophisticated classified as structural equivalence and regular equivalence.”[1] Or more generally, we specify structural equivalence to be the 1st order similarity where connected nodes have higher levels of similarity. In contrast, regular equivalence is 2nd order similarity where nodes with more shared neighbors are assumed to be more similar.

III. DATASET

A. Gathering Data

We collected GitHub data from GitHub Archive [9]. GitHub Archive is a "project to record the public GitHub timeline, archive it, and make it easily accessible for further analysis". These events are aggregated into hourly archives accessible with any HTTP client represented as JSON events. Downloading the raw data into disk is impossible especially if we are to examine an entire year of
entries. Hence, we used the alternative method to query the database.

By setting up an account by Google BigQuery, we can utilize the Web UI to send query jobs and retrieve data that matches our conditions. Below is a sample query that takes 2013 archived data, applies the conditions and return the following entities: repository_name, repository_owner, actor, repository_language and event_types:

```
SELECT repository_name, repository_owner, actor, repository_language
FROM [gitcopy_2013]
WHERE (type CONTAINS 'PullRequestEvent',
  OR type CONTAINS 'FollowEvent',
  OR type CONTAINS 'PushEvent')
AND (repository_language CONTAINS "JavaScript"
  OR repository_language CONTAINS "Ruby"
  OR repository_language CONTAINS "Python"
  OR repository_language CONTAINS "Java"
  OR repository_language CONTAINS "PHP")
AND actor != "_
GROUP BY repository_name, repository_owner, actor, repository_language;
```

Due to the size of the data, we have restricted our query to the top 5 languages used in 2013 and beginning of 2014. These include JavaScript, Ruby, Python, Java and PHP. We used data from 2013 for training and data from February and March 2014 for testing.

B. Data Cleaning

We picked 3000 top users who have the most event occurrences in the training data and then 3000 top users from the testing period. To make relevant predictions, we need to enforce the constraint that the same user group appears in both training and testing phase so that we do not penalize for making wrong predictions on users who do not exist in the testing users but appear in the training data. An analysis on the average number of co-committers show that there is a large variation in the range of 3 to 50.

IV. PERFORMANCE METRIC

Assume we have social network $G = (V,E)$, in which each edge $e = (u,v) \in E$ represents an interaction took place at a specific time $t(e)$. Multiple interactions between $u$ and $v$ as parallel edges will be recorded. Suppose there are two possible times $t < t'$, $G[t,t']$ can be denoted as the subgraph of $G$ consisting of all edges with a timestamp between $t$ and $t'$. After that, it is a concrete formulation network problem of the linking prediction. If four times are selected, $t_0, t_1, t_0, t_1$ and $t_0 < t_0 < t_1$, given an algorithm access to network $G[t_0, t_0']$, the output has to not be present in $G[t_0, t_0']$ but predicted in $G[t_1, t_1']$. $G[t_0, t_0']$ is referred to be training interval and $G[t_1, t_1']$ is test interval.

V. METHODOLOGY

A. PathSim

As covered in the class, PathSim favors peers, which are objects with strong connectivity and similar visibility with a given meta-path. We modified the code in Homework 1 to predict co-committers on Github. Instead of papers (P), actors(A) commit to repositories(R). And instead of Terms (T) or Venues(V), a repository uses some programming language (L).

The limitation of PathSim is that we have to pick one type of meta-path. PathPredict, which we also implemented, solves this problem.

B. P-PageRank

As covered in the Homework 1, Personalized PageRank (P-PageRank) is variant of PageRank. But instead of teleporting to an random node in the network, P-PageRank jumps to a set of predefined nodes, which are called preferred nodes. For searching similar co-committers, we have only one preferred node. P-PageRank try to find the $v$ in:

$$v = (1-c)Av + cu$$

where $c$ is the teleportation constant, $A$ is the adjacency matrix whose every column normalized to 1, $v$ is the pagerank scores, and $u$ is the preference vector where $u$ (queried actor) = 1 and the rest of $u$ contains only zeros.

We modified the code in Homework 1 to predict co-committers on Github.

C. PathPredict

The framework of PathPredict is based on two main parts, one is topological features in heterogeneous network, and another is co-committers prediction model. Basically, the path predict problem can be treated as a classification task, the task is to detect whether the two users have committed on the same repository or not. With the idea of classification, procedure has been transformed into feature selection and model choosings.
1) Topological Features Selection: First, we focus on the approach to systematically define the topological features in the Github community network. Topological features are also called structural features, which aim at extracting connectivity properties for pairs of objects. Topological feature-based link prediction aims at inferring the future connectivity by leveraging the current connectivity of the network. There are some frequently used topological features defined in homogeneous networks, such as the number of common neighbors, preferential attachment and so on. We firstly review several commonly used topological features in homogeneous networks, and then propose a systematic meta path-based methodology to define topological features in heterogeneous networks.

Existing Topological Features

Now we list several frequently used topological features in homogeneous network, including Common Neighbors, Jaccard Coefficients, Adamic/Adar, Hitting Time and Rooted Pagerank.

a. Common Neighbors:

For two nodes, x and y, the size of their common neighbors is dened as $|\Gamma(x) \cap \Gamma(y)|$. The idea of using the size of common neighbors is just an attestation to the network transitivity property. In simple words, it means that in social networks if vertex x is connected to vertex z and vertex y is connected to vertex z, then there is a heightened probability that vertex x will also be connected to vertex y. So, as the number of common neighbors grows higher, the chance that x and y will have a link between them increases.

b. Jaccard Coefficients:

The common neighbors metric is not normalized, so one can use the Jaccard Coefficient, which normalizes the size of common neighbors as below:

$$\text{Jaccard} = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

Conceptually, it denes the probability that a common neighbor of a pair of vertices x and y would be selected if the selection is made randomly from the union of the neighbor-sets of x and y. So, for high number of common neighbors, the score would be higher.

c. Adamic/Adar:

Adamic and Adar [2] proposed this score as a metric of similarity between two web pages. For a set of features z, it is dened as below.

$$\sum_{z: \text{features shared by } x, y} \frac{1}{\log(\text{frequency}(z))}$$

For link prediction, customized this metric as below, where the common neighbors are considered as features.

$$\text{adamic/adar}(x, y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log(|\Gamma(z)|)}$$

In this way, Adamic/Adar weighs the common neighbors with smaller degree more heavily. From the reported results of the existing works on link prediction, Adamic/Adar works better than the previous two metrics.

d. Hitting Time:

The concept of hitting time comes from random walks on a graph. For two vertices, x and y in a graph, the hitting time, $H_{x,y}$ denotes the expected number of steps required for a random walk starting at x to reach y. Shorter hitting time denotes that the nodes are similar to each other, so they have a higher chance of linking in the future. Since this metric is not symmetric, for undirected graphs the commute time $C_{x,y} = H_{x,y} + H_{y,x}$, can be used. The benet of this metric is that it is easy to compute by performing some trial random walks. On the downside, its value can have high variance; hence, prediction by this feature can be poor. For instance, the hitting time between x and y can be affected by a vertex z, which is far away from x and y; for instance, if z has high stationary probability, then it could be hard for a random walk to escape from the neighborhood of z. To protect against this problem we can use random walks with restart, where we periodically reset the random walk by returning to x with a xed probability $\alpha$ in each step. Due to the scale free nature of a social network some of the vertices may have very high stationary probability $\pi$ in a random walk; to safeguard against it, the hitting time can be normalized by multiplying it with the stationary probability of the respective node, as shown below:

$$\text{normalized hitting time}(x, y) = H_{x,y}\pi_y + H_{y,x}\pi_x$$

e. Rooted Pagerank:

Since pagerank is an attribute of a single vertex, it requires to be modied so that it can represent a similarity between a pair of vertices x and y. The original denition of pagerank denotes the importance of a vertex under two assumptions: for some xed probability $\alpha$, a surfer at a web-page jumps to a random web-page with probability $\alpha$ and follows a linked hyperlink with probability $1 - \alpha$. Under this random walk, the importance of an webpage $v$ is the expected sum of the importance of all the web-pages u that link to v. In random walk terminology, one can replace the term importance by the term stationary distribution. For link prediction, the random walk assumption of the original pagerank can be altered as below: similarity score between two vertices x and y can be measured as the stationary probability of y in a random walk that returns to x with probability $1 - \beta$ in
each step, moving to a random neighbor with probability \( \beta \). This metric is asymmetric and can be made symmetric by summing with the counterpart where the role of \( x \) and \( y \) are reversed. In [36], it is named as rooted pagerank. The rooted pagerank between all node pairs (represented as RPR) can be derived as follows. Let \( D \) be a diagonal degree matrix with \( D[i, i] = \sum A[i, j] \). Let, \( N = D^{-1} A \) be the adjacency matrix with row sums normalized to 1. Then, \( RPR = (1 - \beta)(I - \beta N)^{-1} \)

Meta Path-based Topological Features

In order to design the topological features in the heterogeneous networks, firstly we define the relationship between two objects using meta paths, and then define measures on the specific topology.

Meta Path-based Topology. As introduced in previous part, a meta path is a path defined over the network schema, and denotes a composition relation over the heterogeneous networks. By checking the existing topological features defined in homogeneous networks, we can find that both the neighbor set-based features and path-based features can be generalized in heterogeneous information networks, by considering paths following different meta paths. For example, if we treat each type of neighbors separately and extend the immediate neighbors to n-hop neighbors, the common neighbor feature between two users is then becoming the count of paths between the two users following different meta paths. For path-based features, such as Katz, it can be extended as a combination of paths following different meta paths. Hence, each meta path defines a unique topology between objects, representing a special relation. Meta paths between two object types can be obtained by traversing on the DBLP network schema, by using standard traversal methods such as the BFS (breadth-first search) algorithm. As the network schema is a much smaller graph compared with the original network, this stage is very fast. For co-committer relation, we extract all the meta paths within a length constraint, say 4, starting and ending with the user type A.

Once the topologies given by meta paths are determined, the next stage is to propose measures on these meta paths. In this paper, we propose four measures along the lines of topological features in homogeneous networks. These are path count, normalized path count, random walk, and symmetric random walk, which are defined as follows.

Path Count: Number of path instances between authors following \( \text{R } PC_R(a_i, a_j) \)

Normalized Path Count: Normalize path count following R by the degree of authors\( \text{NPC}_R(a_i, a_j) = \frac{PC_R(a_i, a_j)}{PC_R(a_i, i) + PC_R(a_j, j)} \)

Random Walk: Consider one way random walk following

\[
\text{RW}_R(a_i, a_j) = \frac{PC_R(a_i, a_j)}{PC_R(a_i, i) + PC_R(a_j, j)}
\]

Symmetric Walk: Consider random walk in both directions

\[
\text{SRW}_R(a_i, a_j) = \text{RW}_R(a_i, a_j) + \text{RW}_{R^{-1}}(a_j, a_i)
\]

To simplify the algorithm, we choose the first type of measure in our analysis, that is to say, we first select the meta path in Github community network, then count the number of path for each type of meta of each pair of user.

### 2) The Co-Commitment Predictive Model:

Second, we introduce the relationship prediction model which models the probability of co-authorship between two authors as a function of topological features between them. Given the training pairs of authors, we first extract the topological features for them, and then build the prediction model to learn the weights associated with these features. In this paper, we choose the standard method, namely, the logistic regression model as the prediction model. For each training pair of authors.

In our project, we choose the standard method, namely, the logistic regression model as the prediction model. For each training pair of authors \(< a_1, a_2 >\), let \( x_i \) be the \((d + 1)\) dimensional vector including constant 1 and \( d \) topological features between them, and \( y_i \) be the label of whether they will be co-authors in the future \( y_i = 1 \) if they will be co-authors, and otherwise 0, which follows binomial distribution with probability \( p_i \). The probability \( p_i \) is modeled as follows:

\[
p_i = \frac{e^{x_i \beta}}{e^{x_i \beta} + 1}
\]

where \( \beta \) is the \((d + 1)\) coefficient weights associated with the constant and each topological feature. We then use standard MLE to derive \( \beta \), that maximizes the likelihood of all training pairs:

\[
L = \prod_i p_i^{y_i} (1 - p_i)^{1 - y_i}
\]
D. Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

This method follows from the CMA-ES first introduced by Bliss et al. [10] in 2013 to model link prediction as a linear combination of similarity indices that describes the topological similarity of local, quasi-local and global measures. However, global measures such as SimRank, Katz and Matrix Forest Index are excluded for computational reasons. The algorithm claims to have fast convergence and good accuracy. In addition to topological features, node attributes are included to improve link prediction. They claim that previous works rely on network architecture to derive the set of predictors. Furthermore, this set of predictors may change with time. Topological similarity indices encode information about overlap of node’s neighborhoods. Node specific attributes serve as a complement to topological indices. This method claims to not require any parametric thresholds or undersampling to counter the computational complexity for large networks.

The goal is to use an evolutionary algorithm to find coefficients for the weight vector that maximize the number of correctly predicted links. Adjustments of weight is done through CMA-ES to remove the assumption that all similarity index weighs the same. Given an objective function, \( f \), we generate a random sequence from a normal distribution such that these search points reduce the objective function. Randomized search address the problems of local optima and noise/outliers. Evolution algorithms imitate biological systems where individuals represent candidate solutions evolve over generations via selection, reproduction, mutation and recombination.

Below is the algorithm pseudo-code for CMA-ES provided by Hansen et al.

In the evolution step, new candidates (offsprings) are sampled according to normal distributions. Each distribution is modeled by a covariance matrix with zero mean and overall standard deviation. 

\[
\lambda_k^{(g+1)} \sim m^{(g)} + \sigma^{(g)} \mathcal{N}(0, C^{(g)}) \quad , k = 1, \ldots, \lambda
\]

At each generation (iteration), the mean (m), covariance matrix (C) and step size (\( \sigma \)) are computed.

Recombination is modeled by selecting new mean values for the distribution functions as a weighted average of the selected points. 

\[
m^{(g+1)} = m^{(g)} + c_m \sum_{i=1}^{\mu} w_i (\lambda_i^{(g+1)} - m^{(g)})
\]
and linear order of weights. \( \lambda \) is the population size of the samples and \( c_m \) is the learning rate.

Mutation involves pertubing the distributions with a random vector. Pairwise dependencies between the variables are modeled as a covariance matrix. The CMA part serves to update the covariance matrix. The covariance matrix is initialized from a single population of one generation. Rank-\( \mu \) and rank-one updates are used to update the covariance matrix.

Rank-\( \mu \) update uses information from previous generations to remedy the problem of using small population size to estimate the covariance matrix. Exponential smoothing is applied to place higher weights on recent generations.

\[
C^{(g+1)} = (1 - c_\mu) C^{(g)} + c_\mu \frac{1}{\sigma^{(g)}} C^{(g+1)}
\]

where \( c_\mu \) is the learning rate and less than 1. It controls how much prior information is retained for updating the covariance matrix.

Rank-one update updates the covariance matrix in the generation sequence using a single selected step only instead of using all selected steps from a single generation in Rank-\( \mu \) update.

\[
C^{(g+1)} = (1 - c_1) C^{(g)} + c_1 y^{(g+1)} y^{T (g+1)}
\]

where \( y^{(g+1)} = \frac{x^{(g+1)} - m^{(g)}}{\sigma^{(g)}} \)

Finally, rank-one and rank-\( \mu \) updates are added to exploit the benefits for both updates. "On the one hand, the information from the entire population is used efficiently by the so-called rank-\( \mu \) update. On the other hand, information..."
of correlations between generations is exploited by using the evolution path for the rank-one update. The former is important in large populations, the latter is particularly important in small populations.”[11]

\[ C^{(g+1)} = (1 - c_1 - c_\mu \sum w_j) C^{(g)} + c_1 \sum \frac{p^{(g+1)}_i (p^{(g+1)}_i)^T}{\text{rank-one update}} + c_\mu \sum w_j H_{U\alpha}^{(g+1)} (H_{U\alpha}^{(g+1)})^T \]

The methodology to use CMA-ES for predicting links is depicted in Figure 4. First, we identify similarity scores between pairs of nodes (users) and construct N adjacency matrices where N is the number of similarity indices used. We used 8 similarity indices for our network: Common Neighbors, Salton Index, Hub depressed Index, Hub Promoted Index, Leicht-Holme-Newman Index, Katz Index, Preferential Attachment and Resource allocation index. The indices are computed using an open source package called PyNetSim. The scores are rescaled to range from 0 to 1 and stacked into a tensor form, \( S \). We used CMA-ES to evolve the weight vector, w such that the error in link prediction is minimized. Using CMA-ES with both rank-1 and rank-\( \mu \) updates, we evolve w for 256 generations. During each epoch, we randomly sample a population of candidate solutions for w from a Gaussian cloud that was generated from the surviving individual from the previous generation. The survivor is the solution that minimizes the prediction error. In order to avoid local optima, restarts can be implemented. However, we did not use restarts in this project. Our fitness function cross reference links for the top 5, 10 and 15 with the training data. By incorporating fitness functions at different scales, we investigate the sensitivity of the top k on link prediction performance in validation. For inference, we use the final survivor for linear combination of the score matrices to predict the links in the test data.

VI. RESULTS

For PathSim and P-PageRank We output the top 5, 10 and 15 co-committers for each user in our test data. A total of 1,258 users are identified. For PathPredict, we select the top 10 users who frequently appear in our data set and output the accuracy rate and AUC score for each user with the pairs of rest users.

A. PathSim

Using meta-path ARA, the accuracy for top 5, top 10, and top 15 co-committers are 55.99%, 40.28%, and 32.89%, respectively. Using meta-path ARLRA, the accuracy for top 5, top 10, and top 15 co-committers are 29.77%, 19.22%, and 15.52%, respectively.

B. P-PageRank

Using meta-path ARA, the accuracy for top 5, top 10, and top 15 co-committers are 40.36%, 29.37%, and 24.34%, respectively. Using meta-path ARLRA, the accuracy for top 5, top 10, and top 15 co-committers are 22.22%, 12.26%, and 9.15%, respectively.

C. PathPredict

Using meta-path count of ARLRA, ARORA, ARERA as features, binary variable 0 and 1 (0 means non co-committer and 1 means co-committer), we select the top seven users among the data set, and the accuracy and AUC score for their top-5 co-committers.
D. CMA-ES

Figure 7 shows the evolution of the weight vector. We see that convergence happened early during generation 50. The fast convergence can be solved by offering restarts in the algorithm. The colormap also shows that Resource Allocation, Leicht-Holme-Newman Index and Common neighbors indices have positive contribution to predicting links whereas indices such as Preferential Attachment and Salton indices have negative weights.

Using the computed weight vector, we construct the top 5, 10, 15 co-committers and compare to the ground truths in the test data. For top 5 results, the accuracy is roughly 49.36% with 2484 out of 5032 predictions correct. For top 10 results, the accuracy is roughly 38.23% with 4328 out of 11322 predictions correct. For top 15 results, the accuracy is roughly 31.85% with 5609 out of 17612 predictions correct.

VII. DISCUSSION

Github is a social network in a loose sense: programmers on Github may not know each other even if they have co-committed for the same repository for a long time. Unlike coauthors of a paper, a repository has almost no upper limit on the number of co-committers. So it’s not surprising that the prediction result on Github based on social network structure is not as accurate as the one on DBLP dataset. Another possible cause for the lower accuracy is that we don’t have attributes or labels for Github repositories. For research papers like the ones in DBLP, labels like terms and published venues are provided clearly. They are very useful to classify related papers into fields. It’s relatively harder to get labels for Github repositories.

We sorted out the top 3000 users from train and test data to reduce the dataset for inspection. However, as a result, the number of committers for these top committers varies from as small as 5 to 15. Furthermore, we speculate that these top users may be top committers in their respective repository. However, frequent committers in their respective repositories may be confused when predicting links.

VIII. FUTURE WORK

We can do text mining on readme files of Github repositories to get more features. For example, in the readme, repository owners often describe the setup environment of the repository. If it is a library for a certain framework or it’s an application for certain problem, it’s highly likely we can retrieve that information from readme. That way, we have more features to aid our prediction. With PathPredict, we can predict not only who an user is going to co-commit with, but also when. So one of the things we can do in the future is to get Github pull events data with time stamp and to measure how accurate PathPredict is on Github dataset.

For the Path Predict, as we can regard it as a classification problem, logistic regression is not the only choice we can try to be path predictive model. It is entirely possible to implement other classification models like support vector machine, random forest and neural network. Also, simple path count can be replaced by other types of features like discriminative frequent pattern such as "Python", "PullRequest" and so on. Originally, we want to try DPClss introduced in Class. Due to the limited time, we hope to apply such a brand new and powerful classification algorithm to such a large heterogeneous network of Github Community.

APPENDIX

In this section, we give the details of how beta which could make the maximum likelihood could 
be derived. Basically, the maximal likelihood can be expressed as:
\[ \mathcal{L}(\theta; x_1, \ldots, x_n) = f(x_1, x_2, \ldots, x_n \mid \theta) = \prod_{i=1}^{n} f(x_i \mid \theta). \]

In binary case, the whole estimation process is listed as below
\[ L(p; x) \approx f(x; p) = \prod_{i=1}^{n} f(x_i; p) = \prod_{i=1}^{n} p^{x_i}(1 - p)^{1-x_i} \]

In our case, it is
\[ L = \prod_{i=1}^{n} p_i^{x_i}(1 - p_i)^{(1-x_i)} \]

where \( p_i = \frac{e^{\beta e_i}}{1 + e^{\beta e_i}} \), and get rid of \( p_i \). Let \( L = \prod_{i=1}^{n} \left( \frac{e^{\beta e_i}}{1 + e^{\beta e_i}} \right)^{x_i} \left( 1 - \frac{e^{\beta e_i}}{1 + e^{\beta e_i}} \right)^{(1-x_i)} \), and take the derivative of \( \beta \) to obtain the beta that maximizes the likelihood of all training size.

REFERENCES


[5] Sun, Yizhou, Rick Barber, Manish Gupta, Charu C. Aggarwal, and Jiawei Han. “Co-author relationship prediction in heterogeneous bibliographic networks.” In Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on, pp. 121-128. IEEE, 2011.


[12] Sun, Yizhou, Rick Barber, Manish Gupta, Charu C. Aggarwal, and Jiawei Han. “Co-author relationship prediction in heterogeneous bibliographic networks.” In Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on, pp. 121-128. IEEE, 2011.