Applying Link Prediction for Repository Recommendation on GitHub

Ting-Po Lee
tingpo@stanford.edu

Wen-Chien Chen
wencchen@stanford.edu

1 Introduction

GitHub is one of the world's most popular platforms for open source software development. As different developers have different expertise and interests, given information about the repositories to which a user has contributed, it may be useful to suggest "similar" repositories that the user may wish to contribute to. For our project, we attempt to solve this problem by performing link prediction on the GitHub repository graph using Supervised Random Walks, and then applying Personalized PageRank on the resulting graph for specific users to obtain repository recommendations.

2 Related Work

The link prediction problem is formalized in the work of Liben-Nowell and Kleinberg [1]. In their article, the authors define the link prediction problem and evaluate the performance of various methods on five coauthorship networks. The methods surveyed utilize different graph characteristics, such as node neighborhoods and the set of paths between two nodes, and techniques such as clustering and low-rank approximation. Their results show that while the surveyed methods generally perform better than the baseline of forming links for proximate node pairs, there is much room for improvement in accuracy.

The Supervised Random Walks algorithm is a link prediction method proposed by Backstrom and Leskovec [2]. Unlike the relatively simple methods surveyed in [1] which predict edges based only on graph structure, Supervised Random Walks is a hybrid approach that combines graph structure with node features (e.g. age and gender in a social network) and edge features (e.g. interaction between individuals in a friendship link). The algorithm can be considered in two phases, in which *edge strengths* are first learned using node and edge features to produce a graph with weighted edges, and a random walk with transition probabilities derived from edge strengths is then performed to obtain the nodes with which to form links for a given starting node.

Since Supervised Random Walks is central to our work, we describe it in greater detail in Section 5.

Lastly, Personalized PageRank is an extension to PageRank proposed by Page, Brin et al. in their original work on the PageRank algorithm [3]. Whereas the teleport vector is uniform over all nodes in basic PageRank, one can instead choose to specify a different probability distribution over the set of nodes, yielding personalized results for different individuals.

3 Problem Definition

The source of our data set is the set of public events that have occurred on GitHub during a specified time interval. To transform this into a more easily analyzable form, we first make the following definitions:

- We say that a user u has *contributed* to a repository r if during the time interval, there was a *PushEvent* or a *PullRequestEvent* where the actor was u and the repository was r.
- We say that a user u has *interacted* with a repository r if during the time interval, there was a *PushEvent*, *PullRequestEvent*, *ForkEvent*, *IssueEvent*, *IssueCommentEvent* or *WatchEvent* where the actor was u and the repository was r.

Our problem is specified in two parts: Link prediction on a repository graph, and repository recommendation for users.

Repository graph link prediction. Given a set of repositories R, a set of users U and a given time interval $[t_1, t_2]$, we construct the repository graph $G = (V, E)$ as follows: Let V be the set of repositories. For each pair of repositories (r_1, r_2) such that some user u contributed to both r_1 and r_2 during $[t_1, t_2]$, we add $e = (r_1, r_2)$ to G. For each edge (r, r'), we consider the following edge features: a) the number of users that have contributed to both r and r' and b) the number of users that have interacted with both r and r'. The link prediction task is then to predict the edges in G at some $t' > t_2$ given G at t_2.

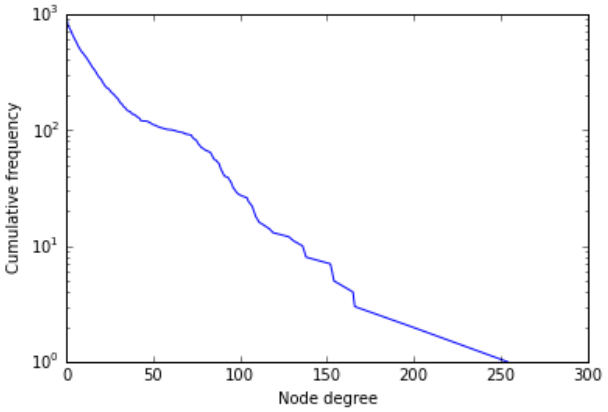

Figure 1: Cumulative degree distribution of repository graph

Repository recommendation. Given a (possibly weighted) repository graph G, a user u, and the set of repositories C_u they have contributed to, the task is to return a recommended set of repositories C'_u which the user is likely to contribute to, such that $C'_u \cap C_u = \varnothing$.

4 Data Collection

For experiment data, we obtained the public events on GitHub for the entire year of 2014 from GitHub Archive [4]. To ensure computational feasibility, we limited ourselves to the 1,000 most-starred repositories that were created prior to 2014 and had pushed commits in 2015 (as an approximation of repositories active throughout 2014). We wrote scripts to extract events concerning those repositories, and to generate snapshots of the repository graph and edge properties. The snapshot at the end of 6/30/2014 is used as the training data, while the snapshot at the end of the year is used as the test data on which to perform link predictions, repository recommendations and evaluation.

We ran some preliminary analysis on the snapshots to explore the graph structure. In particular, we plotted the cumulative degree distribution for the repository graph at the end of 6/30/2014, as shown in Fig. 1. Since the distribution most closely fits an exponential curve, there are likely preferential attachment dynamics.

5 Supervised Random Walks

In this section, we outline the mathematical model for Supervised Random Walks as described in [2].

Given a graph $G = (V, E)$ and a source node s, the set of candidate nodes for future links is defined as $C = \{u \in V | (s, u) \notin E\}$. We further define the future destination set $D \subset C$ consisting of nodes that form links with s in the future, and the and no-link set $L = C - D$. For each edge $(u, v) \in E$ with edge feature vector Z_{uv}, the edge strength a_{uv} is defined as:

$$a_{uv} = f_\beta(Z_{uv})$$

Supervised Random Walk predicts future links for a source node s by the Personalized PageRank scores for s. For nodes $u, v \in C$ when the associated $p_u > p_v$ we say s is more likely to connect to u in the future compared to v. Thus, given a graph $G(V, E)$, source node s and edge features Z our learning objective is to train a model with parameter β to come up with Personalized PageRank scores for s that we can use to predict future destination set D by selecting K nodes from C that have the highest PageRank scores.

5.1 The Optimization Problem

The loss function in the learning process is formed based on future destination set D and no-link set L. For each node $d \in D$ and $l \in L$, we expect the PageRank score $p_d \geq p_l$ and we will penalize any occurrence of $p_d < p_l$. Therefore we setup the loss function as follows:

$$\Gamma(\beta) = \sum_{d \in D, l \in L} h(p_l(\beta) - p_d(\beta))$$

The PageRank scores p_l and p_d depend on parameter β. The function $h(\cdot)$ imposes a penalty on $p_l - p_d > 0$. To complete the optimization problem formulation, an ℓ_2 norm regularization on the parameter β is added. This complete optimization problem for the learning process is:

$$\min_\beta F(\beta) = \sum_{d \in D, l \in L} h(p_l - p_d) + \lambda \|\beta\|^2 \qquad (1)$$

5.2 The Learning Algorithm

The PageRank scores p relate to parameter β through the random walk transition matrix Q. If we have $|V| = n$, the transition matrix Q is an $n \times n$ matrix. Given edge strength function $f_\beta(Z_{uv})$ and the

calculated edge strength a_{uv}, Q_{uv} is the conditional transition probability from node u to node v and is defined as:

$$Q_{uv}(\beta) = \begin{cases} (1-\alpha)\frac{a_{uv}(\beta)}{\sum_k a_{uk}(\beta)} + \alpha\mathbf{1}(v=s), & \text{if } (u,v) \in E, \\ \alpha\mathbf{1}(v=s), & \text{otherwise} \end{cases} \tag{2}$$

Where s denotes the random walk source node and α is the random walk restart probability, *i.e.*, with probability α the random walk will transport back to the source node s. With this formulation, each row of Q sums to 1.

The PageRank score p, by definition, is the stationary probability distribution the satisfies the following condition:

$$p^T = p^T Q(\beta) \tag{3}$$

This can be rewritten as:

$$p_u = \sum_k p_k Q_{ku} \tag{4}$$

Now that the relationship between p and β is established, we can proceed to optimize the learning objective function. We first derive the gradient of objective function F.

$$\begin{aligned} \frac{\partial F(\beta)}{\partial \beta_j} &= 2\beta_j + \sum_{d\in D, l\in L} \frac{\partial h(p_l - p_d)}{\partial \beta_j} \\ &= 2\beta_j + \sum_{d\in D, l\in L} \frac{\partial h(p_l - p_d)}{\partial (p_l - p_d)} (\frac{\partial p_l}{\partial \beta_j} - \frac{\partial p_d}{\partial \beta_j}) \end{aligned} \tag{5}$$

Given Eq. 4, the gradient of PageRank socre p can therefore be written as:

$$\frac{\partial p_u}{\partial \beta_j} = \sum_k \left(\frac{\partial p_k}{\partial \beta_j} Q_{ku} + p_k \frac{\partial Q_{ku}}{\partial \beta_j} \right) \tag{6}$$

Define a PageRank score gradient vector as follows:

$$p'_{\beta_j} = \left(\frac{\partial p_1}{\partial \beta_j}, \frac{\partial p_2}{\partial \beta_j}, ..., \frac{\partial p_n}{\partial \beta_j} \right)^T$$

We can then rewrite Eq. 6 in matrix form:

$$\left(p'_{\beta_j}\right)^T = \left(p'_{\beta_j}\right)^T Q + p^T \frac{\partial Q}{\partial \beta_j} \tag{7}$$

Elements Q_{uv} of Q are directly related to $a_{uv} = f_\beta(Z_{uv})$, and the partial derivatives are obtained by differentiating Eq. 2. We omit the result as the derivation is relatively straightforward.

By combining Eq. 3 and Eq. 6, we can compute the PageRank vector and its gradient using the following equations:

$$p(t+1)^T = p(t)^T Q(\beta) \tag{8}$$

$$\left(p'_{\beta_j}(t+1)\right)^T = \left(p'_{\beta_j}(t)\right)^T Q + p^T \frac{\partial Q}{\partial \beta_j} \tag{9}$$

We first compute the PageRank vector by repeatedly solving Eq. 8 until convergence. The result is then plugged into Eq. 9, which is also computed until convergence to obtain the gradient. This allows us to evaluate Eq. 5 and optimize the objective function using gradient descent or other optimization algorithms.

6 Repository Recommendation

Given edge strengths learned using Supervised Random Walks, we can produce a weighted repository graph, where a higher edge strength indicates greater commonality in terms of users and contributors between two repositories. Assuming that users are more likely to be interested in working on software projects similar to ones to which they have contributed, we can recommend repositories for individual users using Personalized PageRank.

Given a user u and the set of repositories C_u they have contributed to, we can run Personalized PageRank on the weighted repository graph, and using Q from Supervised Random Walks as the transition matrix and C_u as the teleport set. Repositories with the highest PageRank scores that are not in C_u are then returned as recommendations.

7 Validating Supervised Random Walks on Synthetic Data

To validate and evaluate the performance of our implementation of Supervised Random Walks, we set up experiments with synthetic graphs and test the algorithm's ability to recover parameters used to generate the graphs and its ability to predict the nodes in future destination set D. The setup is largely the same as the one used in [2].

7.1 Experiment Setup

Synthetic graphs consist of 1,000 nodes and are generated with the Copying model as follows: Initially, 3 connected nodes are put in the graph. Additional nodes arrive sequentially, forming 3 edges with existing nodes. Destination nodes of the new edges are

determined such that with probability $\pi = 0.5$, the destination node is chosen uniformly at random from existing nodes, and with probability $1 - \pi$, the destination node is selected with probability proportional to its degree. This produces preferential attachment.

Given a graph $\tilde{G} = (V, E)$ constructed using the above method, we then randomly generate features for each edge. Features are first generated at the node level: For each node $u \in V$, two features X_{u1}, X_{u2} are independently generated from the normal distribution $N(0, 1)$. Then for each edge $(u, v) \in E$, two features Z_{uv1}, Z_{uv2} are generated by $Z_{uvi} = X_{ui}X_{vi}$. To track the effect of unobserved features, we can also apply random noise to node features, *i.e.* $X'_{ui} = X_{ui} + \epsilon_{ui}$, where $\epsilon \sim N(0, \sigma^2)$.

Edge strengths a_{uv} are then calculated to form a transition matrix for random walks. In our synthetic data, the strengths are derived from an exponential function:

$$a_{uv} = \exp\left(Z_{uv1}\beta_1 + Z_{uv2}\beta_2\right)$$

where $\beta = [1, -1]$.

A source node s is randomly picked from the 3 initial nodes in $\tilde{G}$ from the Copying model, then Personalized PageRank scores are calculated for s with transition matrix Q calculated from a_{uv} and restart probability $\alpha = 0.1$. With these PageRank scores, the future destination set D_s is obtained by picking the top K nodes that are not currently connected to s in $\tilde{G}$. In our test, we have $K = 100$.

Given a graph $\tilde{G}$, edge features Z, source node s and training sets D_s, L_s, we expect Supervised Random Walks to be able recover parameter $\hat{\beta}$ that is close to $[1, -1]$. We also wish to test the algorithm's performance in predicting D_s.

7.2 Learning Process Parameters

For the learning process, we choose the loss function $h(\cdot)$ to be:

$$h(\delta) = \begin{cases} \delta^2, & \text{if } \delta > 0, \\ 0, & \text{otherwise} \end{cases}$$

Since we have a 2-dimensional feature for each edge in our synthetic graph, overfitting is less likely to be an issue in our model and we impose no parameter regularization, *i.e.* $\lambda = 0$ in $F(\beta)$. For optimization, we've applied the BFGS algorithm, which uses fewer iterations than gradient descent and does not require manually specifying a learning rate.

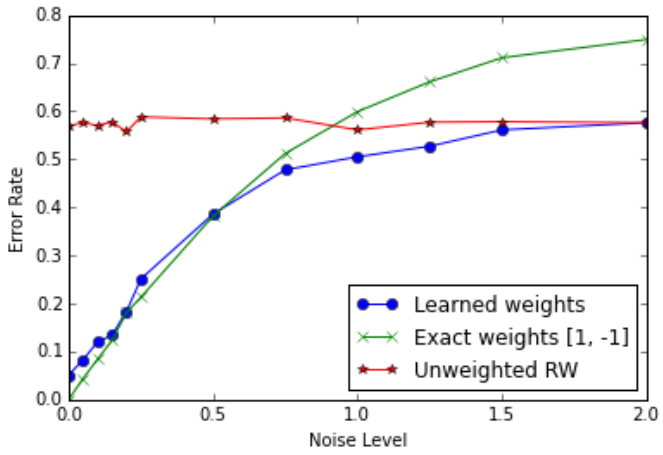

Figure 2: Error rate of destination set predicted by Supervised Random Walks and benchmarks. The noise variance ranges from 0 (indicating no noise) to 2.0.

7.3 Synthetic Data Test Results

Synthetic graphs with different noise levels are tested for prediction accuracy. In the graph without noise, we expect the learned parameter $\hat{\beta}$ to be very close to $[1, -1]$. As the noise level increases, the prediction accuracy may gradually decrease.

Accuracy is measured by error rate, which is calculated by the portion of true future destination set D_s recovered by the predicted future destination set $\hat{D}_s$. To evaluate relative performance, we compare the error rates of Supervised Random Walks to using the true parameter $[1, -1]$ and unweighted random walk.

Fig. 2 demonstrates the performance of Supervised Random Walks compared to the benchmarks as noise variance increases from from 0 (indicating no noise) to 2.0. For each noise level, 10 independent graphs were generated and the average error rate for the 10 graphs is shown in the figure.

For synthetic graphs with no noise, Supervised Random Walks achieves near-zero error rate, as expected. As the noise level increases, Supervised Random Walks starts to outperform exact weights. This is because the learning process adapts to rising noise level, and is able to learn parameters that result in less error. Since unweighted random walk does not take node or edge features into account, its performance is stable across noise levels. The performance of Supervised Random Walks gradually approaches that of unweighted random walk as the noise level in-

creases. This is expected, as node and edge features become poor predictors when the noise level is high, and graph characteristics come to dominate prediction performance.

Unlike the results reported in [2], the error rate of Supervised Random Walks is slightly higher than the exact weights for low noise levels in our synthetic data. This is likely because we used a slightly looser convergence condition, making the recovered parameter β less exact. However, such low noise is unlikely in real data, and so we believe this to be a reasonable trade-off for better computational feasibility.

8 Evaluation on GitHub Data

Having validated our implementation of Supervised Random Walks on synthetic data, we now proceed to evaluate it on the GitHub Archive data set. We first apply Supervised Random Walks to the repository graph and evaluate its performance in link prediction, and then we apply Personalized PageRank to the weighted repository graph from the first step and evaluate its performance in repository recommendation for individual users.

8.1 Prediction Model Setup

In this section, we briefly describe the functions and parameters we chose for Supervised Random Walks and our rationale.

Loss function. We use the Wilcoxon-Mann-Whitney loss function (WMW) with width b, which has the following form:

$$h(x) = \frac{1}{1 + \exp(-x/b)}$$

WMW is chosen as unlike squared loss where negative PageRank differences (from correct rankings) are trimmed, WMW makes use of both positive and negative PageRank differences and thus retains more information in model training, leading to better link prediction accuracy on our data set. The width parameter $b = 0.01$ is empirically tuned based on the magnitude of differences between PageRank scores in our data.

Edge strength function. We choose to use a logistic function for edge strengths, as it has a range of $(0, 1)$, which reduces the possibility of overflow in model training and evaluation:

$$a_{uv} = \frac{1}{1 + \exp(-Z_{uv1}\beta_1 - Z_{uv2}\beta_2)}$$

Restart probability and regularization. We choose the restart probability to be $\alpha = 0.3$ based on the suggestion in [2], which performs well for link prediction in our experiments. As we observe overfitting in the absence of model regularization, we set the regularization parameter to be $\lambda = 50$ based on several runs of the experiment.

Optimizer. For real-world evaluation, we use L-BFGS to train our model. Compared to BFGS, L-BFGS is more memory efficient and provides better efficiency in solving large-scale problems.

8.2 Experimental Setup

Snapshots of the GitHub repository graph at the end of 6/30/2014 and 12/31/2014 are used to train and test our model. The training set is formed by randomly picking 200 nodes among those that form at least 20 links between 7/1/2014 and 12/31/2014. Another 100 nodes in this set are randomly chosen to be the test set. The extra condition is applied so that we can compare the top 20 link predictions to the true links formed for nodes in the test set.

The Supervised Random Walks model is trained with the future link set D for each node in the training set being the new neighbors added between 7/1/2014 and 12/31/2014, and the no-link set L being the nodes that are not neighbors at the end of 12/31/2014. The trained model is then used to perform link prediction for nodes in the test set.

Given the parameter β learned by Supervised Random Walks, a weighted repository graph is produced using the 6/30/2014 snapshot. With this, we perform repository recommendation as described in section 6 for all users that have contributed to at least 1 repository between 1/1/2014 and 6/30/2014, and at least 4 or more new ones between 7/1/2014 and 12/31/2014. The criteria are set to make it possible to compare the recommended repositories to ones the user actually contributed to.

8.3 Repository Graph Link Prediction

We compare the link prediction performance of Supervised Random Walks to that of unweighted random walk. Figure 3 shows the average number of correctly predicted links as the number of predictions increases. With 20 predictions, Supervised Random Walks and unweighted random walk achieved an average of 6.66 and 6.51 hits, respectively. This result is close to that obtained in [2] for the Facebook dataset.

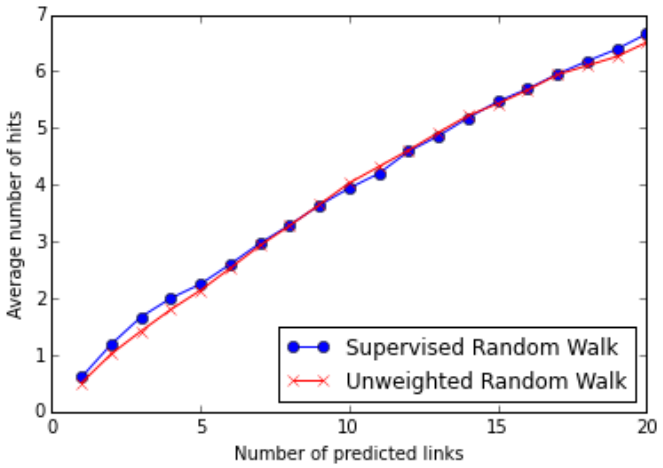

Figure 3: Average number of link prediction hits with increasing number of predicted links.

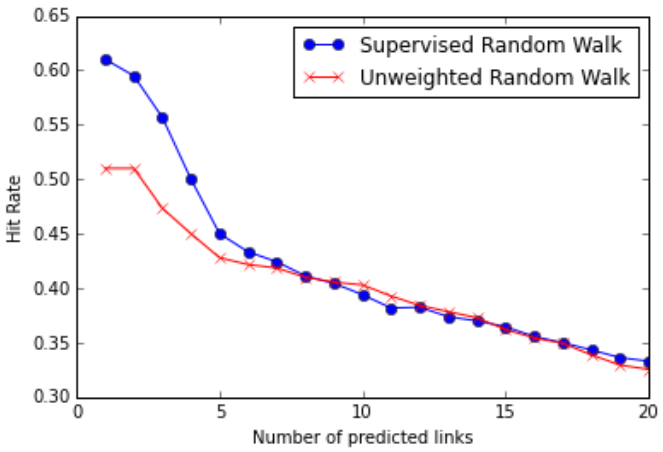

Figure 4: Accumulated link prediction hit rate with increasing number of predicted links.

Figure 4 shows the average ratio of correct predictions. With 5 or fewer predictions, Supervised Random Walks outperforms its unweighted counterpart by exploiting edge features. As the number of prediction increases, the difference becomes insignificant as the network structure becomes the dominant factor.

The parameter learned by Supervised Random Walks is $\beta = [9.89, 4.15]$, with the first term being the weight for the number of common contributors, and the second being that for the number of users that have interacted with both repositories. This is consistent with our expectation that the number of common contributors plays a greater role in link formation.

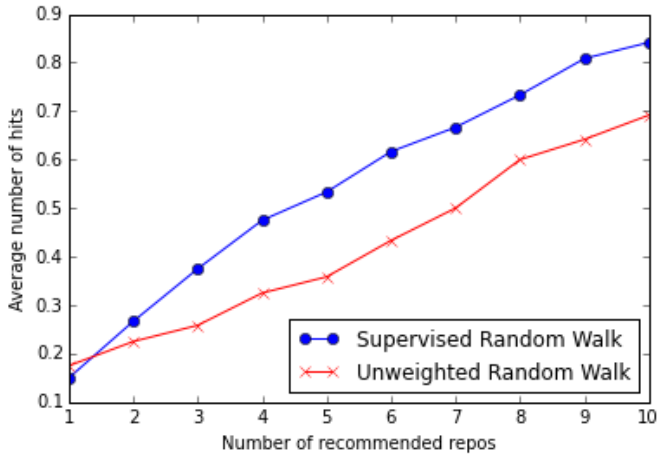

Figure 5: Accumulated link prediction hit rate with increasing number of predicted links.

8.4 Repository Recommendation to Users

We evaluate the repository recommendation performance of our approach using the average number of repositories in our recommendations that the users ended up contributing to between 7/1/2014 and 12/31/2014. The results are compared against those obtained by running Personalized PageRank on an unweighted repository graph. Figure 5 shows the number of correctly predicted repositories as the number of recommendations increases.

With 10 recommendations, the average number of correct recommendations achieved by Supervised Random Walks and unweighted random walk are 0.85 and 0.69, respectively. This suggests that the repository graph as constructed is not a good predictor for the interaction between users and repositories. Additionally, we believe these factors may have contributed to the poor performance:

- For many users, the teleport set used to compute Personalized PageRank contains only one element, making personalized recommendation more difficult.
- Since we are performing repository recommendation using weighted repository graph originally computed for link prediction on the repository graph, there is an additional level of indirection. In particular, parameters suited to link prediction may not be suited to repository recommendation.

Despite the results, it's worth noting that Supervised Random Walks visibly outperforms unweighted

random walk with 2 or more recommendations. This implies that the edge features of the repository graph do play some part in user-repository interaction dynamics.

9 Conclusion

We have implemented Supervised Random Walks, validated it on synthetic data and applied it to the GitHub Archive data set, showing that it is an effective method for link prediction on the GitHub repository graph, and performs well compared to unweighted random walk. We have additionally proposed, implemented and evaluated a method for recommending repositories to users based on Personalized PageRank and the weighted repository graph from Supervised Random Walks. While it is not particularly effective, it still outperforms unweighted random walk.

Future research directions include adding node features to Supervised Random Walks to improve link prediction performance, and exploring other methods for repository recommendation that better capture the interactions between users and repositories.

References

[1] D. Liben-Nowell, J. Kleinberg. The Link Prediction Problem for Social Networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, 2004.

[2] L. Backstrom, J. Leskovec. Supervised Random Walks: Predicting and Recommending Links in Social Networks. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, 2011.

[3] L. Page, S. Brin, R. Motwani, T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, 1999.

[4] `https://www.githubarchive.org/`