Analyzing the structure and dynamics of large-scale social networks

Project report by
Arpan Basu (*arpan0123@gmail.com*)
of
Jadavpur University
under the supervision of
Dr. Bivas Mitra
and under the mentorship of
Mr. Ayan Bhowmick
as a part of the Summer Internship Program at
Complex Networks Research Group,
Department of Computer Science and Engineering,
IIT Kharagpur

Contents

1 Introduction

As social networks become more widespread, their impact on the society as a whole become more pervasive. Social networks can be characterized in terms of nodes (people or other objects within the network) and edges (relationships or interactions) among the nodes. Analysis of social networks lies at the heart of various applications and disciplines which include data mining, network modelling, behaviour analysis, recommender systems, law enforcement, etc. Modern social networks as generally classified as *large* due to a substantial number of nodes and edges present in the network; and as *complex* because they demonstrate pseudo-random patterns of interactions. Due to these factors, it is a challenge to find algorithms on networks which are relatively fast, and also exhibit a reasonable degree of accuracy.

Many tasks in network analysis consist of predictions over nodes and edges. For example, in the link prediction task, we may want to know whether two nodes should have a link in-between them. This is useful with regard to social networks to identify real-world friends and also in other fields such as genomics to identify interactions. However many algorithmic frameworks require careful feature engineering to be able to produce good results. We often want to use only the network representation for such tasks. However, such node-level features which are derived from the adjacency matrix are inherently sparse in nature, which in turn produces poor results when directly used in prediction tasks. In addition, the memory footprint of such features scale quadratically with the number of nodes in the network. In this report we consider a community-detection based approach for the problem of generating dense embedding vectors for the nodes in lower dimensions which uses less memory (and less time) while preserving accuracy. We compare the results of link prediction on the generated embeddings with a state-of-the-art approach — *node2vec*.

Parallel to the task of generating features, it is also difficult to generate precise predictions on social networks, whether by using embedded features or by using manually engineered features. In this report we also consider the common prediction tasks on networks, and analyse their performance when applied to real-world data. Specifically, we consider the heterogeneous *event-based social network* (EBSN) platform Meetup, an online social network whose members hold events at physical locations. Compared to strictly online networks, it is a challenge to develop high-accuracy prediction algorithms on EBSNs due to the following factors:

- Heterogeneous character of the network: EBSNs like Meetup consist of different types of entities and hence, different types of interactions among them. Utilising the varied interactions is critical for generating accurate predictions on these networks.
- Real-world interactions: In contrast to online networks, members of EBSNs interact in offline social events. These face-to-face interactions require more effort on behalf of the member and must be taken into account as the member's time is a limited resource. In addition, the dynamics of real-world relationships (like physical distances and interpersonal associations) also come into play.

The Meetup social network consists of three main entities: users, events, and groups. Groups organise events which users attend. Each event is associated with a real-world venue. Users and groups are associated with tags. Tags, for users, indicate the subjects or topics which the user is interested in, and, for groups, the primary areas of interest of the group. Groups organise events on subjects related to their tags.

We consider two recent approaches for prediction on EBSNs: one method for the task of recommending new groups to users and predicting the future group size from these recommendations; and another method for predicting the success of individual groups based on event attendance.

2 Datasets

For the tasks of node embedding generation and subsequent link prediction, we use graph datasets which are available online, chiefly from SNAP[1] [1] and KONECT[2] [2]. The following datasets were used to benchmark our performance:

- *Zachary's Karate Club* [3]: Undirected, 34 nodes, 78 edges.
- *Hamsterster friendships* [2]: Undirected, 1858 nodes, 12534 edges.
- *OpenFlights* [4]: Directed, 2939 nodes, 30501 edges.
- *Facebook* [5]: Undirected, 4039 nodes, 88234 edges.
- *Arxiv ASTRO-PH* [6]: Undirected, 18772 nodes, 198110 edges.

For the task of prediction on EBSNs, we use the data as extracted in the work by Pramanik *et al.* [7]. The data was obtained from the Meetup[3] EBSN using two crawlers for the cities Chicago, New York, and San Francisco during a time frame of eight months (from August 2015 to March 2016). The details of the two crawlers are as follows:

[1] `https://snap.stanford.edu/data/index.html`
[2] `http://konect.cc/networks/`
[3] `https://en.wikipedia.org/wiki/Meetup`

Table 1: Summary of the data collected by the two crawlers for the three cities

City	Fast Crawler		Detail Crawler				
	Groups	Members	Groups	Events	Venues	RSVPs	Member Profiles
Chicago	5727	342773	5671	31719	435553	2749595	249652
New York	17180	1026901	17094	81786	1150394	7910224	814378
San Francisco	13381	753839	13297	65266	833045	6542690	606097

1. Fast Crawler: This is a fast crawler with a cycle duration of 3 days. It collects members of all groups and generates a member to group mapping with timestamps. It does not collect any event or venue related information.
2. Detail Crawler: This is a slow crawler with a cycle duration of 7 to 10 days. It collects event details (like event venue, event attendance, etc) and group details (like member profile, events hosted, etc).

When a member first joins Meetup, they are required to select some tags from a global set of tags according to their preference. This corresponds to the *member profile* in the data. A similar process exists when a group is newly created. The *headcount* feature of events provides the total number of event attendees but does not give individual member attendance details. Details of individual attendees are extracted from the RSVP replies which belong to the set {*Yes*, *No*, *Maybe*}. Event attendees are taken to be those members who select the *Yes* response to RSVPs.

The summary of the entire data collected can be found in Table 1.

3 Node Embeddings

In this section, we consider the efficient generation of node embeddings for a given network based on community detection.

3.1 Problem formulation

Let $G = (V, E)$ be a given network. If we consider the adjacency matrix based node representation, then we have a $|V|$ length vector for each node corresponding to each row of the adjacency matrix. Alternatively, we may say that there exists a function $f : V \to \mathbb{R}^{|V|}$. Such a representation has a space complexity of $O(|V|^2)$. The required problem is to generate a d length (dense) vector for each node of the network to produce a function $g : V \to \mathbb{R}^d$. Here d is a parameter denoting the length of the embedding vector for each node. In practice, we choose d to be much less as compared to $|V|$ ($d \ll |V|$). Therefore, the space complexity becomes $O(|V|d) \equiv O(|V|)$ which is an improvement on the adjacency matrix based representation. However, such an embedding should also preserve the node-level information. We show this to be true for our approach by evaluating on the task of link prediction.

3.2 Louvain algorithm for community detection

Blondel *et al.* [8] have proposed a fast method for community detection in networks by applying a simple heuristic based on modularity [9]. The algorithm consists of two stages which are repeated iteratively as highlighted below:

1. First phase: Initially, each node of the network is assigned a different community. For each node i the neighbours j of i are taken and the gain of modularity that would take place by removing i from its community and by placing it in the community of j is evaluated. The node i is then placed in the community for which this gain is maximum, but only if this gain is positive. If no positive gain is possible, i stays in its original community. This process is applied repeatedly and sequentially for all nodes until no further improvement can be achieved.

2. Second phase: The second phase of the algorithm consists of building a new network whose nodes are the communities found during the first phase. The weights of the edges between the new nodes are given by the sum of the weight of the edges between nodes in the corresponding two communities. Edges between nodes of the same community lead to self-loops. Once this second phase is completed, it is then possible to reapply the first phase again to the resulting weighted network.

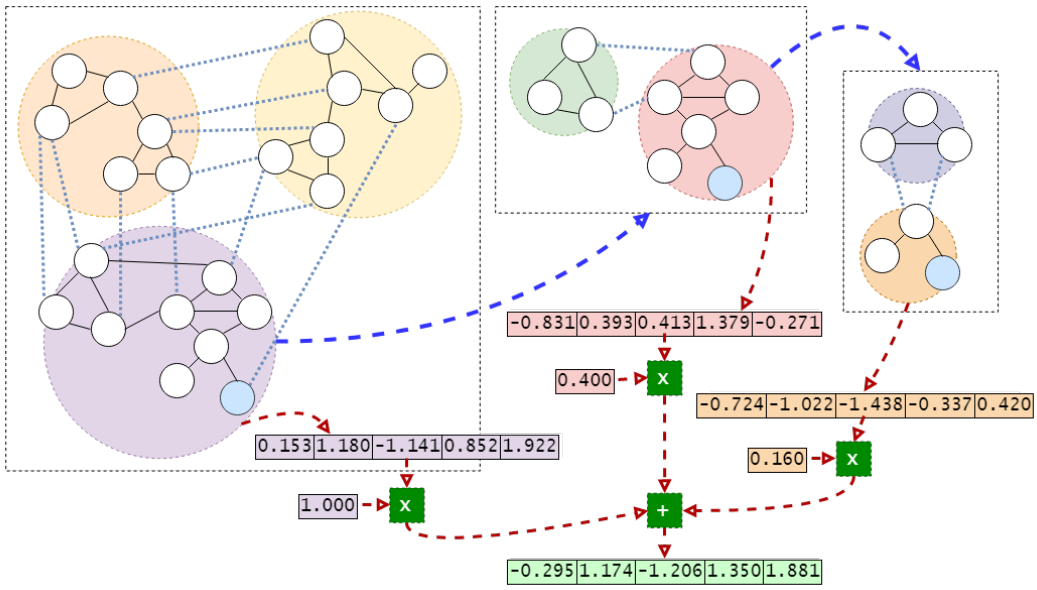
We use the official C++ implementation[4] of Louvain available at Sourceforge for conducting our experiments. Although Python-Louvain[5] is a good alternative given that we are using Python, we found it to be very slow for large graphs.

[4] `https://sourceforge.net/projects/louvain/files/`
[5] `https://pypi.org/project/python-louvain/`

3.3 Our approach for generating node embeddings

Our approach for generating node embeddings is primarily based on the recursive application of the Louvain algorithm. A pictorial outline of the method is present in Figure 1. We further elucidate the method in the text below.

Figure 1: Pictorial outline of the node embedding approach



In the above figure, each dotted rectangle denotes the communities obtained by the Louvain algorithm at different depths of recursion. The first (and largest) rectangular box denotes the communities present in the original network, while the blue arrows denote subsequent recursive runs of Louvain on the respective communities. Each coloured circle within a particular box represents a different community. The smaller circles represent nodes. Adjacent nodes within the same community are joined by continuous black lines, and those in different communities by dotted blue lines.
We trace the embedding generation process for the blue node in the above diagram. Each community is assigned a randomly sampled vector. The embedding for a particular node is produced by a linear combination of the vectors of the communities to which the node belongs. The red arrows denote the data flow during this step.

Let $G = (V, E)$ be the given network. Application of the Louvain algorithm on G will produce a partition of V into disjoint subsets such that $V = \bigcup_{i=1}^{n} V_i$ where n is the total number of communities produced by Louvain. Let $S = \{G_i : G_i = G[V_i], 1 \leq i \leq n\}$ be the collection of subgraphs induced by the nodes present in the same communities. Then we can apply Louvain again to each of these subgraphs. The above process can be repeated recursively to a certain depth of recursion, say, d. For each level of recursion i, we obtain a function $f_i : V \to \mathbb{N}, 1 \leq i \leq d$ which maps each node to the corresponding community (here communities are taken as natural numbers which denote the position of the community in some fixed enumeration of the communities at the specific level).

3.3.1 Generating embeddings for the communities

Let us consider C_i which is the set of all the communities generated at the recursion depth $i(\leq d)$ and let $C_i = \{c_{i,j} : c_{i,j} \in \mathbb{N}, 1 \leq j \leq |C_i|\}$ where $c_{i,j}$ corresponds to a community. To each $c_{i,j}$ we assign a $\mathbb{R}^{dim}$ vector sampled from a probability distribution P where dim is a parameter corresponding to the length of the embedding vector. For the choice of P, we evaluate using the standard normal distribution $\mathcal{N}(0, 1)$ and the standard uniform distribution $\mathcal{U}(0, 1)$. Thus for each level of recursion i, we have a function $g_i : C_i \to \mathbb{R}^{dim}, 1 \leq i \leq d$ which maps each community (at that level of recursion) $c_{i,j}$ to a dim length vector.

3.3.2 Generating embeddings for the nodes

As there exist functions f_i and g_i for each recursion depth i, we obtain a sequence S_v (see expression 1) for each node v in the network, d being the maximum recursion depth. Each element of S_v is a dim length vector.

$$S_v = \{g_1(f_1(v)), g_2(f_2(v)), ..., g_d(f_d(v))\} \tag{1}$$

It only remains to combine these vectors into a single vector representing the embedding vector for the node. We use the notation $S(i)$ to denote the i^{th} element of the sequence S. Let us consider another sequence A such that

$A \in \mathbb{R}^d$. Then the resulting embedding vector for the node is given by expression 2 where E_v denotes the embedding vector for a node v.

$$E_v = \sum_{i=1}^{d} (A(i) \times S_v(i)) \tag{2}$$

We note that A is simply a sequence of real numbers. We evaluate the approach with two choices of A:

1. In the first case, A is in decreasing geometric progression with first term 1 and common ratio $\alpha < 1$. More weightage is given to top-level (or global) communities.
2. In the second case, A is in increasing geometric progression with last term 1 and common ratio $\alpha^{-1} > 1$. More weightage is given to bottom-level (or granular) communities.

3.4 Experimental setup

To evaluate effectiveness of our approach, and also to find the best alternative among the choices of P and A, we find the top-k scores on different datasets. For some fixed value of k, the top-k score is found as follows:

- Firstly, we need a measure of similarity among two node embeddings. We use the cosine similarity (see expression 3) for this purpose. It produces a value between 0 and 1 with 0 denoting complete dissimilarity and 1 denoting complete similarity.

$$CosineSim(v_1, v_2) = \frac{v_1 \cdot v_2}{|v_1||v_2|} \tag{3}$$

- Then we find the similarity among all pairs of nodes in the network, and sort the values in a decreasing order according to the similarity score. We take the first k node pairs from the sorted values.
- Finally we find the top-k score which is the fraction of node-pairs having an edge between them in the network from among the k node-pairs.

The different parameter choices are summarized in Table 2. The top-k scores on the *Facebook* and *OpenFlights* graphs are present in Tables 3 and 4 respectively. The scores for the *Hamsterster* and *Karate* graphs are present in Tables 20 and 21 respectively in Appendix A. We notice that the scores decrease with increasing k values as expected. However, when A is in increasing G.P. (and bottom-level communities are assigned a higher priority), the fall-off at higher k values is more significant as compared to the case when A is in decreasing G.P. (and top-level communities are assigned a higher priority). This can be explained by the fact that broad-level node interactions are less marked in the former case as compared to the latter. Granular interactions perform satisfactorily up to a certain threshold, beyond which node-level information is required at a global level for accurate predictions. Additionally, we find the the standard uniform distribution has a slightly worse performance as compared to the standard normal distribution in the case of the directed *OpenFlights* graph.

Table 2: Parameters for the top-k score calculations

Id	α	dim	d	A	P	Clipping
1				decreasing G.P.	$\mathcal{N}(0,1)$	–
2				decreasing G.P.	$\mathcal{N}(0,1)$	$[-1,1]$
3	0.4	64	3	increasing G.P.	$\mathcal{N}(0,1)$	–
4				decreasing G.P.	$\mathcal{U}(0,1)$	–
5				increasing G.P.	$\mathcal{U}(0,1)$	–

Note: For the *Karate* graph, *dim* is taken as 10.

We choose the parameters corresponding to id 1 in Table 2 for all subsequent experiments.

In addition to the top-k scores, we also find the AUC scores in a randomly sampled set of node-pairs containing positive (existing) and negative (non-existing) edges in equal proportions. The prediction set is taken to be the cosine similarity among the node pairs, while the ground truth is a binary vector of 0s and 1s — 0s indicating negative edges and 1s indicating positive edges. The scores for the *Facebook*, *OpenFlights*, *Hamsterster* and *Karate* graphs are described in Tables 5, 6, 7 and 8 respectively. Here f is a parameter denoting the fraction of the total edges present in the network used as positive examples. We note that the performance is better for denser networks.

Table 3: Top-k scores for the *Facebook* graph

k	Parameter Id (see Table 2)				
	1	2	3	4	5
25	0.93	0.97	0.89	0.85	1.00
50	0.83	0.81	0.81	0.85	0.83
75	0.85	0.83	0.72	0.85	0.82
100	0.85	0.82	0.75	0.85	0.82
200	0.83	0.83	0.80	0.85	0.83
1000	0.81	0.79	0.79	0.77	0.81
2000	0.83	0.83	0.80	0.79	0.83
5000	0.79	0.79	0.79	0.77	0.80
10000	0.80	0.80	0.79	0.80	0.80
20000	0.72	0.72	0.70	0.72	0.70
40000	0.56	0.57	0.38	0.56	0.36
80000	0.41	0.41	0.20	0.41	0.19

Table 4: Top-k scores for the *OpenFlights* graph

k	Parameter Id (see Table 2)				
	1	2	3	4	5
10	0.93	1.00	0.93	0.87	1.00
25	0.88	1.00	0.93	0.85	0.96
50	0.88	0.95	0.89	0.87	0.93
75	0.86	0.90	0.88	0.87	0.91
100	0.87	0.82	0.87	0.82	0.88
200	0.79	0.84	0.83	0.75	0.83
500	0.72	0.68	0.70	0.67	0.72
1000	0.60	0.52	0.58	0.54	0.53
1500	0.51	0.45	0.48	0.52	0.52
2500	0.48	0.54	0.52	0.44	0.44
5000	0.43	0.44	0.41	0.33	0.30
7500	0.39	0.39	0.39	0.23	0.15
10000	0.33	0.33	0.30	0.15	0.08
15000	0.27	0.27	0.21	0.11	0.06
20000	0.23	0.23	0.16	0.10	0.04
30000	0.18	0.18	0.11	0.09	0.04

Table 5: AUC scores for the *Facebook* graph

Iter.	AUC Score at $f =$		
	0.50	0.75	1.00
0	0.96	0.96	0.96
1	0.96	0.96	0.96
2	0.96	0.96	0.96
3	0.96	0.96	0.96
4	0.96	0.96	0.96
5	0.96	0.96	0.96
6	0.96	0.96	0.96
7	0.96	0.95	0.96
8	0.95	0.95	0.96
9	0.95	0.95	0.95
Mean	0.958	0.957	0.959

Table 6: AUC scores for the *OpenFlights* graph

Iter.	AUC Score at $f =$		
	0.50	0.75	1.00
0	0.91	0.91	0.90
1	0.90	0.91	0.90
2	0.90	0.90	0.90
3	0.90	0.90	0.89
4	0.90	0.89	0.89
5	0.89	0.89	0.89
6	0.89	0.89	0.89
7	0.89	0.88	0.89
8	0.89	0.88	0.89
9	0.88	0.88	0.89
Mean	0.895	0.893	0.893

3.5 Link prediction

In the actual problem of link prediction, the goal is to predict the formation of edges among nodes in the future, given the state of the network at present. However, the common procedure of evaluation is to take the network and remove a certain fraction of edges. Then we would like to predict these missing edges. The training set is generated by taking a certain fraction f of edges from the network as positive examples and an equal number of negative examples from the non-edges. It should be noted that the resultant network formed by the positive examples should include all nodes and must not contain single isolated nodes. Complete connectivity is desirable, but not necessary. The test set comprises of the remaining edges of the network as positive examples, and an equal number of randomly sampled non-edges (not present in the training set) as negative examples.

To generate an embedding for a node-pair (u, v), we simply generate the Hadamard (or element-wise) product of the two node embeddings i.e. $E_u \odot E_v$ which produces a *dim* length vector. These edge-level embeddings are used as features for the corresponding link prediction task. We find the AUC and accuracy scores using common Machine Learning classification algorithms — *Logistic Regression*, *Random Forest Classifier*, and, *K-Nearest Neighbours Classifier*. We use the Scikit-Learn[6] [10] library in Python for this task. The corresponding observations for the *Facebook* and *Arxiv ASTRO-PH* graphs are present in Tables 9 and 10 respectively; we note that the *Random Forest Classifier* produces the best results. We also compare the AUC scores at $f = 0.5$ with *node2vec* [11], a proven method for generating node (and hence, edge) embeddings. Our method produces AUC scores which are better or are at-par with *node2vec* as represented in Table 11.

[6] `https://scikit-learn.org/`

Table 7: AUC scores for the *Hamsterster* graph

Iter.	AUC Score at $f =$		
	0.50	0.75	1.00
0	0.77	0.80	0.78
1	0.77	0.79	0.78
2	0.77	0.78	0.77
3	0.76	0.78	0.77
4	0.76	0.77	0.77
5	0.76	0.77	0.76
6	0.75	0.76	0.76
7	0.74	0.75	0.76
8	0.74	0.75	0.76
9	0.74	0.75	0.76
Mean	0.756	0.77	0.767

Table 8: AUC scores for the *Karate* graph

Iter.	AUC Score at $f =$		
	0.50	0.75	1.00
0	0.80	0.76	0.73
1	0.74	0.73	0.71
2	0.72	0.71	0.70
3	0.69	0.70	0.66
4	0.68	0.67	0.66
5	0.66	0.66	0.66
6	0.62	0.65	0.65
7	0.61	0.64	0.65
8	0.58	0.61	0.64
9	0.54	0.59	0.62
Mean	0.664	0.672	0.668

Table 9: Link prediction scores on the *Facebook* graph

f	*Logistic Regression*		*Random Forest Classifier*		*K-Nearest Neighbours Classifier*	
	AUC	Acc	AUC	Acc	AUC	Acc
0.90	0.9765	0.9379	0.9852	0.9614	0.9812	0.9575
0.75	0.9730	0.9357	0.9842	0.9563	0.9790	0.9557
0.50	0.9744	0.9346	0.9820	0.9510	0.9771	0.9493

4 Predictions on EBSNs using MMCs

In this section, we consider the utility of a recommendation-based approach based on multivariate Markov Chains (MMCs) for predicting the future size of the groups present in EBSNs.

4.1 Preface

Heterogeneous networks in general consist of multiple types of entities and their associated interactions. Recommendation is a useful task with respect to these networks. It is important for a prediction model to explicitly consider the influence strength between the different entities which helps to produce better performance with respect to prediction or recommendation tasks. Random Walks with Restart (RWR) is a widely used method for recommendation tasks on networks. Although proposed primarily for homogeneous networks, it can be extended [12] to heterogeneous graphs by considering all the nodes and edges to be of the same type. The main drawbacks with this method are the loss of the distinctive properties of nodes and edges; and that the method is based on univariate Markov Chains.

Pham *et al.* [13] propose a general recommendation model (named *HeteRS*) on heterogeneous networks which improves upon the RWR-based approaches. It is based on multivariate Markov Chains (MMCs) and utilizes a learning based framework to find appropriate model parameters by maximizing an objective function using gradient ascent. This is in contrast to some MMC-based approaches which need manual parameter assignment. *HeterRS* takes the heterogeneous network graph as input and calculates the proximity among nodes by simulating a random walk process using MMC. *HeteRS* can therefore be applied on Meetup which is a heterogeneous EBSN. We then use the *group-to-user* recommendations generated by the model to predict the future group size; and evaluate the predictions using a binning-based metric.

4.2 Experimental setup

We extract the relevant network information from the Meetup data (mentioned in section 2) which is required for the algorithm. As the data is of a temporal nature, we consider the timeline on the basis of event times — starting with the first event, and ending with the last event in the dataset. For training the model, we consider the first sixty percent of the entire timeline. The resultant network is augmented with session nodes as mentioned in [14] and is provided as input to the recommendation model. The model then generates the outputs as mentioned below:

- Probability distribution vectors $\mathbf{u}$, $\mathbf{e}$, $\mathbf{g}$, $\mathbf{t}$, $\mathbf{s}$, $\mathbf{v}$ representing the probabilities that users, events, groups, tags, sessions and venues are visited (at some time step t), respectively. Of particular interest is the $\mathbf{g}$ vector which, for a particular user, gives the probabilities of visiting each group.
- Transition probabilities between the different types of nodes which are used in calculating the distribution vectors.

Table 10: Link prediction scores on the *Arxiv ASTRO-PH* graph

f	*Logistic Regression*		*Random Forest Classifier*		*K-Nearest Neighbours Classifier*	
	AUC	Acc	AUC	Acc	AUC	Acc
0.90	0.8662	0.8109	0.9571	0.9042	0.9391	0.8771
0.75	0.8328	0.7955	0.9559	0.9069	0.9392	0.8774
0.50	0.8412	0.7906	0.9401	0.8848	0.9216	0.8585

Table 11: Comparison of the AUC scores of *node2vec* and our approach at $f = 0.5$

Dataset	AUC scores	
	node2vec	our approach
Facebook	0.9680	0.9820
Arxiv ASTRO-PH	0.9366	0.9401

4.2.1 Filtering

The original data as a whole consists of a sizeable number of nodes of each type of entity. To achieve faster results due to time constraints, we employ a filtering strategy for the various component nodes of the network. The filtering strategy for each type of component node is as follows:

- Users: Users are filtered by their cumulative event attendance.
- Groups: Groups are filtered by their size in terms of number of users.
- Events: Events are filtered by the number of attending users.
- Tags: Tags are filtered by their usage frequency over all the groups and users present.
- Venues: Venues are filtered by the total number of events that took place at that venue.

After filtering, we get three datasets each corresponding to one of the three cities (as mentioned in section 2). We also combine them and generate two more datasets. A brief highlight of the datasets is given below:

1. *Chicago*: contains filtered network data from Chicago.
2. *New York*: contains filtered network data from New York.
3. *San Francisco*: contains filtered network data from San Francisco.
4. *Combined*: contains the combined filtered data from Chicago, New York and San Francisco. Light filtering has been to done to restrict the maximum number of nodes of each type at around 10^4.
5. *Combined Small*: a heavily filtered version of the combined data.

The node statistics after filtering is summarized in Table 12.

Table 12: Node statistics after filtering

Node Type	Number of nodes				
	Chicago	*New York*	*San Francisco*	*Combined*	*Combined Small*
Users	3656	3637	3588	7842	3032
Groups	5463	5468	5429	10020	5869
Events	6614	6374	6225	8490	5791
Tags	4297	4117	4035	6678	4277
Venues	5121	4046	4043	9295	4377

4.2.2 Prediction

To predict the future size of the groups, we first predict the groups that a user will join. For this task, we take k groups with the highest probabilities from the user's $\mathbf{g}$ vector that the user is not a part of. We then consider the user to have joined these k groups in the future. If we now perform this step for all users, and shift our perspective to the groups, we have the predictions for the future group sizes by adding the above predictions for the users to the initial group size. We recount that this prediction generation step is done for the first sixty percent of the timeline.

For the ground-truth corresponding to the predictions, we consider the group sizes corresponding to the full timeline, restricting the groups to those existing in the first sixty percent of the timeline. For evaluating the accuracy of the

predictions, we consider binning the values after normalizing them in the range $[0, 1]$ (by dividing with the maximum value from among the predicted and ground-truth values). A prediction is considered to be correct if its bin is the same as the ground-truth value's bin; otherwise the prediction is considered as incorrect.

4.2.3 Binning and corresponding accuracies

In this section we mention the bin (boundary) values, the actual data values corresponding to those, and the overall accuracies. Tables 14, 15 and 16 compare the accuracy scores across different k values from 1 through 5 on the *Combined* dataset. We note that there is a slight gain of accuracy as k increases. The remaining tables are present in Appendix B. We mention the tables corresponding to a particular dataset in Table 13 for easy reference.

Table 13: Table references for all the datasets

Dataset	Tables
Chicago	22, 23, 24
New York	25, 26. 27
San Francisco	28, 29, 30
Combined Small	31, 32, 33
Combined	14, 15, 16

Table 14: *Combined*(1)

Bin Boundary	Actual Value
0	0.0
0.007	54.2
0.02	154.7
0.04	309.4
0.11	851.0
0.22	1701.9
0.44	3403.8
0.74	5724.6
1	7736.0

k	Accuracy
1	0.86
2	0.87
3	0.88
4	0.88
5	0.88

Table 15: *Combined*(2)

Bin Boundary	Actual Value
0	0.0
0.005	38.7
0.025	193.4
0.05	386.8
0.1	773.6
1	7736.0

k	Accuracy
1	0.80
2	0.82
3	0.82
4	0.83
5	0.83

Table 16: *Combined*(3)

Bin Boundary	Actual Value
0	0.0
0.0026	20.1
0.0091	70.4
0.0194	150.1
0.0388	300.2
0.0776	600.3
0.1551	1199.9
0.2585	1999.8
1	7736.0

k	Accuracy
1	0.64
2	0.65
3	0.65
4	0.66
5	0.66

5 Predictions on EBSNs using GRNF

In this section, we consider the effectiveness of a deep neural network based method to predict the popularity of newly created groups in Meetup. Li *et al.* [15] propose the method under consideration which uses not only group-level features, but also the internal structural features of the group represented through *Group Role Neural Fingerprints* (GRNF; see section 5.3.3). The method is described in the subsequent sections.

5.1 Meetup group popularity prediction

The group popularity prediction problem is stated as follows:

> Given all the related information of a group within a time window of $[0, n]$ months, and a time interval of m months, predict the group's total RSVP number (popularity) within a future time window of $[n + m, 2n + m]$ months

5.2 Group-level features

These features indicate the summary statistics of the group at a broad level. Fourteen group-level features are used, which are described in Table 17. However, using only group-level features may not yield the best performance as it

ignores the more granular features of the entities inside the group. For this limitation, the internal group features are considered in section 5.3.

Table 17: Group-level features used

Entropy of the time distribution over all times the events are held.
Average distance between any two events the group held.
Variance of the event-event distances.
Average distance between any event and any participating member.
Variance of the event-member distances.
Average distance between any member and any other member in the same group.
Variance of the member-member distances.
Entropy of the location distribution overall venues the event are held.
Density of the group's social graph.
Total degree of the groups social graph.
Event number the group has held.
Average RSVP number of all past events.
Variance of RSVP numbers of past events.
Sum RSVP number of past events.

Table 18: Member-level features used

Total degree of the member.
Event number the member has participated in current group.
Group number the member has joined.
Entropy of member's attendance distribution over the groups the member joined.
Entropy of event number distribution over the groups the member joined.
Entropy of event fraction distribution over the groups the member joined.
Average degree of the member's 1-hop neighbors.
Average event number the 1-hop neighbors have participated.
Average group number the 1-hop neighbors have joined.
Average entropy of 1-hop neighbors' attendance distribution over the groups they have joined.
Average entropy of 1-hop neighbors' event number distribution over the groups they joined.
Average entropy of 1-hop neighbors' event fraction distribution over the groups they joined.

5.3 Internal group features

5.3.1 Member-level features

For some group g, the event co-participation social graph is a graph $G(U^g, E^g, W^g)$ where U^g is the set of users in group g, E^g denotes all edges between members and W^g represents all edge weights. Two users u_i and u_j are connected if they co-participated in at least one event in group g, and the weight of the edge is given as the total number of events the two users have co-participated in. Corresponding to the group's event co-participation social graph, twelve member-level features are extracted as described in Table 18.

5.3.2 Member role discovery

For finding the role features corresponding to each group member, *non-negative matrix factorization* (NMF) [16] is employed. For a given member-feature matrix $X \in \mathbb{R}^{n \times f}$, a rank $r(< min(n, f))$ approximation $MF \approx X$ is generated where $M \in \mathbb{R}^{n \times r}$ and $F \in \mathbb{R}^{r \times f}$. Here n and f denote the number of members and features per member respectively; and r is a parameter corresponding to the number of roles. The matrix M thus produced is the member-role matrix each row of which represents the corresponding member's role associations.

5.3.3 Group-Role Neural Fingerprints

In order to extract structural information from the member roles, the circular fingerprints algorithm [17] can be used. However, it has some limitations: it can only handle graphs of fixed sizes; and it is highly sensitive to slight variations in the input. Improving upon these limitations, Duvenaud *et al.* [18] proposed a convolutional neural network based framework for the circular fingerprints algorithm which produces a real-valued vector as output. It is extended (in [15]) from the case of chemical molecules for application on social networks, and is termed as *Group-Role Neural Fingerprints* (GRNF). GRNF produces a real-valued vector as output from the given social graph of the group as input.

5.4 Neural network architecture

We implement the neural network in PyTorch[7] [19] and make use of its automatic differentiation library due to the unconventional GRNF convolutional network. The forward pass implementation is highlighted below:

1. The group level feature vector is first downsampled by passing through a neural network.
2. The output vector thus obtained is then concatenated with the output vector obtained from the GRNF algorithm.
3. The concatenated vector is then passed through a final neural network to obtain the predictions.

The above network is trained end-to-end by *stochastic gradient descent* using *mean squared error* (MSE) as the objective function. *Exponential Linear Unit* (ELU) was used as the activation function, and gradient clipping to a global norm of 2 was performed to counter the exploding gradients problem.

5.5 Results

We evaluate the results for $n = 4$ and $m = 1$. The *root-mean-squared error* (RMSE) of our implementation is similar to the RMSE mentioned by the original authors. We report the accuracy of the model after binning; the information being present in Table 19.

Table 19: Values of the bin boundaries and the obtained accuracy using GRNF

Bin Boundary Value	
0	
20	
80	
200	
Accuracy	0.62

Although the results are reasonable, we would like to note that the neural network is highly sensitive to the random initialization of the weights. Many iterations converge to local optima, learning to predict the mean of the expected outputs. This is the case because these local optima also produce MSE values comparable to the iterations which converge accurately.

6 Acknowledgements

I would like to express my gratitude to *Dr. Bivas Mitra* and the *Department of Computer Science and Engineering, IIT Kharagpur* for hosting me as an intern.

I would also like to thank *Mr. Ayan Bhowmick* who guided me throughout the project.

References

[1] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[2] Jérôme Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350, New York, NY, USA, 2013. ACM.

[3] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.

[4] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32(3):245 – 251, 2010.

[5] Jure Leskovec and Julian J. Mcauley. Learning to discover social circles in ego networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 539–547. Curran Associates, Inc., 2012.

[7] `https://pytorch.org/`

[6] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007.

[7] Soumajit Pramanik, Midhun Gundapuneni, Sayan Pathak, and Bivas Mitra. Can i foresee the success of my meetup group? In *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '16, pages 366–373, Piscataway, NJ, USA, 2016. IEEE Press.

[8] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.

[9] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864, New York, NY, USA, 2016. ACM.

[12] D. Nie, Y. Fu, J. Zhou, Z. Liu, Z. Zhang, and C. Liu. A personalized recommendation algorithm via biased random walk. In *2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 292–296, May 2014.

[13] T. N. Pham, X. Li, G. Cong, and Z. Zhang. A general recommendation model for heterogeneous networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(12):3140–3153, Dec 2016.

[14] T. N. Pham, X. Li, G. Cong, and Z. Zhang. A general graph-based model for recommendation in event-based social networks. In *2015 IEEE 31st International Conference on Data Engineering*, pages 567–578, April 2015.

[15] Guangyu Li, Yong Liu, Bruno Ribeiro, and Hao Ding. On group popularity prediction in event-based social networks. In *12th International AAAI Conference on Web and Social Media, ICWSM 2018*, pages 644–647. AAAI press, 1 2017.

[16] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.

[17] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010. PMID: 20426451.

[18] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015.

[19] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Appendix A Tables corresponding to the top-k scores

This appendix contains the tables denoting the top-k scores on the *Hamsterster* and *Karate* graphs respectively.

Table 20: Top-k scores for the *Hamsterster* graph

k	Parameter Id (see Table 2)				
	1	2	3	4	5
10	0.83	0.86	1.00	0.83	0.97
25	0.85	0.80	0.84	0.85	0.85
50	0.87	0.88	0.87	0.85	0.87
75	0.87	0.85	0.87	0.82	0.87
100	0.86	0.82	0.82	0.83	0.84
200	0.81	0.76	0.81	0.77	0.77
500	0.69	0.67	0.71	0.67	0.68
1000	0.57	0.60	0.55	0.54	0.55
1500	0.57	0.62	0.59	0.59	0.59
2500	0.55	0.58	0.55	0.56	0.57
5000	0.43	0.43	0.41	0.43	0.40
7500	0.35	0.35	0.28	0.36	0.27
10000	0.31	0.31	0.21	0.31	0.21
12000	0.27	0.27	0.18	0.27	0.18

Table 21: Top-k scores for the *Karate* graph

k	Parameter Id (see Table 2)				
	1	2	3	4	5
10	0.99	0.95	0.97	0.99	0.99
20	0.75	0.75	0.77	0.76	0.76
40	0.49	0.49	0.50	0.50	0.50
60	0.55	0.55	0.53	0.55	0.54
80	0.51	0.51	0.45	0.51	0.44

Appendix B Tables corresponding to the ***HeterRS*** accuracies

This appendix contains the observation tables corresponding to the datasets of *Chicago*, *New York*, *San Francisco* and *Combined Small* (as mentioned in section 4.2.3) with $k = 5$.

Table 22: *Chicago*(1)

Bin Value	Actual Value
0	0.0
0.007	7.2
0.02	20.6
0.04	41.2
0.11	113.2
0.22	226.4
0.44	452.8
0.74	761.5
1	1029.0
Accuracy	0.48

Table 23: *Chicago*(2)

Bin Value	Actual Value
0	0.0
0.005	5.1
0.025	25.7
0.05	51.5
0.1	102.9
1	1029.0
Accuracy	0.56

Table 24: *Chicago*(3)

Bin Value	Actual Value
0	0.0
0.0194	20.0
0.0680	70.0
0.1457	149.9
0.2916	300.1
0.5831	600.0
1	1029.0
Accuracy	0.80

Table 25: *New York*(1)

Bin Value	Actual Value
0	0.0
0.007	19.2
0.02	54.8
0.04	109.7
0.11	301.6
0.22	603.2
0.44	1206.5
0.74	2029.1
1	2742.0
Accuracy	0.69

Table 26: *New York*(2)

Bin Value	Actual Value
0	0.0
0.005	13.7
0.025	68.6
0.05	137.1
0.1	274.2
1	2742.0
Accuracy	0.62

Table 27: *New York*(3)

Bin Value	Actual Value
0	0.0
0.0073	20.0
0.0255	69.9
0.0547	150.0
0.1094	300.0
0.2188	599.9
0.4376	1199.9
1	2742.0
Accuracy	0.71

Table 28: *San Francisco*(1)

Bin Value	Actual Value
0	0.0
0.007	24.8
0.02	70.9
0.04	141.8
0.11	390.0
0.22	779.9
0.44	1559.8
0.74	2623.3
1	3545.0
Accuracy	0.66

Table 29: *San Francisco*(2)

Bin Value	Actual Value
0	0.0
0.005	17.7
0.025	88.6
0.05	177.3
0.1	354.5
1	3545.0
Accuracy	0.72

Table 30: *San Francisco*(3)

Bin Value	Actual Value
0	0.0
0.0056	19.9
0.0198	70.2
0.0423	150.0
0.0846	299.9
0.1693	600.2
0.3385	1200.0
1	3545.0
Accuracy	0.67

Table 31: *Combined Small*(1)

Bin Value	Actual Value
0	0.0
0.007	20.9
0.02	59.6
0.04	119.2
0.11	327.8
0.22	655.6
0.44	1311.2
0.74	2205.2
1	2980.0
Accuracy	0.72

Table 32: *Combined Small*(2)

Bin Value	Actual Value
0	0.0
0.005	14.9
0.025	74.5
0.05	149.0
0.1	298.0
1	2980.0
Accuracy	0.65

Table 33: *Combined Small*(3)

Bin Value	Actual Value
0	0.0
0.0067	20.0
0.0235	70.0
0.0503	149.9
0.1007	300.1
0.2013	599.9
0.4027	1200.0
1	2980.0
Accuracy	0.72