

An Algorithm for Temporal Analysis of Social Positions

Scott Christley and Greg Madey
University of Notre Dame
{schristl,gmadey}@nd.edu

Abstract

Positional analysis of a social network seeks to group actors into disjoint subsets according to their social position in the network. Such analysis requires a measure to determine whether two actors are equivalent and thus hold the same social position; for social networks, structural or approximate structural equivalence using Euclidean distance or correlation is one well-known measure. In the Data Mining literature, the problem of grouping into disjoint subsets is termed clustering, and popular algorithms include K-means, K-mediod, and Expectation-Maximization(EM). In this article, we present a novel algorithm that uses a statistical test for determining, with a degree of confidence, the structural in-equivalence of two actors; actors are thus separated out into disjoint subsets or clusters which correspond to the social positions in the network. The algorithm does not require the number of clusters to be specified a-priori, and we show how the algorithm can be simply extended to analyze the social positions over time. We use our algorithm to analyze the social positions of the Open Source Software Community at SourceForge.net.

Contact:

Scott Christley
Dept. of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46656

Tel: 1-574-631-7596

Fax: 1-574-631-9260

Email: schristl@nd.edu

Key Words: Social Network Analysis, Positional Analysis, Data Mining, Open Source Software, Temporal Analysis

Acknowledgement: Dr. Nitesh Chawla for fruitful data mining discussions and SourceForge.net for providing data.

Support: This work was supported in part by the National Science Foundation, CISE/IIS-Digital Society & Technology, under Grant No. 0222829.

An Algorithm for Temporal Analysis of Social Positions

Scott Christley and Greg Madey

Positional analysis [Wasserman & Faust, 1994] seeks to group actors into disjoint subsets according to their social position in the network. A social position can be considered a pattern of embeddedness in the social network, so actors who are similarly embedded occupy similar social positions. The social network analysis concept behind social positions is that actors in the same position are indistinguishable from each other, and those actors can be interchanged without altering the structure of the social network. Determining whether two actors occupy the same social position requires some equivalence measure; for social networks, structural equivalence is the defining measure; that is, do both actors hold the exact same relations to other actors in the network. Generally, actors are rarely exactly structurally equivalent, so approximate measures need to be used in order to obtain worthwhile results; Euclidean distance and correlation are two well-known approximate measures. There are other techniques like CONCOR and hierarchical clustering which can be considered as well.

In the Data Mining literature [Hand et al, 2001; Witten & Frank, 2000], the general problem of grouping elements into disjoint subsets is called clustering, and popular algorithms include K-means, K-mediod, and Expectation-Maximization(EM). K-means clustering can be used when approximate structural equivalence is desired with Euclidean distance or correlation; however, it requires the number of clusters, K , to be specified a priori. K-mediod is identical to K-means except it uses the median value to describe the cluster instead of the mean; while, EM is an iterative application of K-means to find the appropriate number of clusters.

An underlying assumption for many of these techniques is the existent of an appropriate equivalence metric, and some go further in assuming the underlying distribution for clusters is a Gaussian (normal) distribution. These are tenuous assumptions as it is unclear whether those distributions hold for structural patterns found in typical social networks. In this paper, we present a novel algorithm that clusters actors into social positions. By utilizing a non-parametric statistical test, it makes no assumptions on the underlying distribution of the social network data; likewise, it actually determines structural in-equivalence of actors thus avoiding issues with similarity metrics. We extend the algorithm to perform temporal analysis of social positions and discuss how different perspectives of time provide different dynamic views of the social network. Lastly, we use our algorithm to analyze the social positions of the Open Source Software Community at SourceForge.net.

Clustering with a Statistical Test

Unlike a distance metric that determines how close sample data is in an N -dimensional space, a statistical test compares sample data to determine if the samples come from the same underlying distribution. For positional analysis, we assume that each social position has a different underlying distribution that defines some sample data; however, this distribution is not known a priori. Therefore, the task of determining the social positions is best done by making as few assumptions as possible on the underlying distributions for each social position while still being able to determine if a sample data set does or does not come from the same distribution as another sample data set. The non-parametric test we use for analyzing social positions in Open Source Software is Fisher's contingency-table test for variables with more than two categories [Krauth 1988]; however, our algorithm is independent of the actual statistical test so other more appropriate non-parametric tests may be used for different data sets.

Given two independent samples (S_1 and S_2) of univariate measurements, the first sample with n_1 random variables and the second sample with n_2 variables, where n_1 is not necessarily equal to n_2 , each random variable in each sample is placed into one of C possible categories. The null and alternative hypotheses become:

H_0 : The distributions of S_1 and S_2 do not differ.

H_A : The distributions S_1 and S_2 differ.

A significant result from the statistical test is interpreted as the null hypothesis is rejected and the alternative hypothesis is accepted; while a non-significant result indicates we fail to reject the null hypothesis. This leads very naturally into an algorithm for clustering that is based upon in-equivalence of data instead of equivalence; in particular, if the null hypothesis is rejected then two samples cannot be from the same underlying distribution (with some level of confidence) therefore those samples cannot be the same social position so they are put into separate clusters. By doing a pair-wise statistical test between each sample, we separate all of the samples which are different, and we are left with a set of samples where we could not reject the null hypothesis and that set then becomes one cluster. We continue this pair-wise comparison process with the remaining unclustered samples, i.e. the samples that got rejected or excluded from a cluster, until all of the samples have been placed into a cluster. The pseudocode for the algorithm is given below.

```

While (still unclustered samples)
  Put all unclustered samples into one cluster
  While (some samples not yet pairwise compared)
    A = Pick sample from cluster
    For each other sample, B, in cluster not yet compared to A
      Run statistical test on A and B
      If significant result
        Remove B from cluster

```

Listing 1: Pseudocode for Clustering Algorithm

In the worst case, if each sample gets put into its own cluster, then each sample will be pair-wise tested with each other sample resulting in $O(n^2)$ computational complexity. This means our algorithm is as efficient or better than other standard data mining clustering algorithms. Another interesting feature is that the confidence level (α value) can be used to adjust how the algorithm clusters; a high confidence level means that stricter evidence is needed to reject the null hypothesis thus resulting in larger clusters while a lower confidence level results in smaller and/or more clusters. For our analysis of the Open Source Software Community, we used $\alpha = 0.05$ and have not compared the resultant clusters from different confidence levels.

Activity Type	Activity Description
Submit bug	Person submits a new bug report.
Assign bug	Bug report is assigned to person.
Submit support request	Person submits a new support request.
Assign support request	Support request is assigned to person.
Submit patch	Person submits a new patch.
Assign patch	Patch is assigned to person.
Submit feature request	Person submits a new feature request.
Assign feature request	Feature request is assigned to person.
Submit todo	Person submits a new to-do item.
Assign todo	To-do item is assigned to person.
Submit other artifact	Person submits an artifact that is not one of the predefined categories of bug report, support request, patch, feature request, or to-do item.
Assign other artifact	Uncategorized artifact is assigned to person.
New forum message	Person posts a new forum message.
Followup forum message	Person posts a forum message that is a follow-up to an existing forum message.
Modify project	Person makes an administrative modification to the project; the modification is uncategorized, but they are typically tasks like adding/removing members, changing permissions, updating project settings, etc.
File release	Person posts a new file release; this is typically associated with releasing a new version of the software to the public.
New project task	Person creates a new project task.
Assign project task	A project task is assigned to person.
Modify project task	Person modifies an existing project task.
Create document	Person creates a new document.
Create people job	Person posts a new job; these are similar to help-wanted ads where a project is looking for somebody with particular skills.

Table 1: Activities Performed in the Open Source Software Community

Social Positions of the Open Source Software Community

The data set we analyze is a January 2003 data dump of SourceForge.net [SourceForge 2003]. SourceForge.net, sponsored by VA Software, is the largest OSS development and collaboration site, and the data dump is derived from a PostgreSQL relational database used as the back-end database for the SourceForge.net website. The social

network we construct is a multi-relational, weighted, bipartite (two-mode or affiliation) network; the nodes are individuals and projects, and the edges link individuals to projects. Each edge between an individual and a project represents a relationship based upon an activity that individual performs for that project, and the weight on the edge corresponds to the quantity of that activity performed by the individual. Multiple edges can exist between an individual and a project where each edge represents a different relationship; we define the different relationships as different types of activities the individual can perform. Table 1 lists the twenty-one types of activities that we extracted from the SourceForge data.

For our clustering algorithm, each user/project pair and the corresponding activities that user performs for that project becomes a single sample, so the statistical test compares a user/project pair against another user/project; each random variable is a single activity event performed by the user and recorded into one of the twenty-one activity categories. Our data set has a total of 209994 user/project pairs with activity from the beginning of SourceForge.net to January 2003 which contains over two million activity events spanning almost 10 years. The result of running our algorithm on this data produced 1577 clusters.

Social Position	Description	Size
Brief Flame	The largest cluster with primary activities of modify project, new forum message and file release.	122654
Message Posting	Only posts forum messages. Combination of four clusters with different posting behavior; the largest cluster at 39613 mainly posts follow-up forum messages with a significant number of new messages.	50067
Task Management	Significant usage of the project task management provided by SourceForge.net. This social position is a combination of five clusters which all have considerable activity for modify project and file release.	2762
Release Management	Primary activities of file release and modify project in various ratios. This social position is a combination of five clusters which have practically no other activity beyond the two main types.	6509
Documentation	Combination of four clusters with one cluster primarily creating new documents using SourceForge.net's document management. The other clusters have create document activity but also modify project, file release and/or forum message postings.	1266
Job Posting	Two clusters with a significant amount of create people job; a facility provided by SourceForge.net that allows projects to advertise help-wanted ads looking for people with specified skills who can contribute to the project.	899
Artifact Management	Significant usage of artifact management provided by SourceForge.net. Artifacts include bug reports, support requests, feature requests, patches, to-do items, and other project-defined artifacts. This social position is a combination of six clusters with different clusters having varied activity ratios for the different activity types, and some clusters also have considerable activity for modify project and file release.	1674
Administrators	Primary activity of modify project. Composed of four clusters; this social position also has some activity for followup forum message and file release.	10377
Not Categorized	The remaining very small clusters that were not analyzed.	13786
Total user/project pairs		209994

Table 2: Social Positions in the Open Source Software Community

The largest cluster has an activity distribution that looks relatively like no other cluster; almost all of the activity is one of three types: modify project, new forum message, or file release. The modify project activity type dominates with almost 75% of the total activity; new forum messages account for around 20%, and file releases is a distant third with a mere 5% of the total activity. Such an activity distribution seems hard to categorize with any normal notions of standard software development, but it does correlate well with the belief that many open source projects are not successful and that they become dormant soon after their initial creation. Our hypothesis is that the scenario runs something like this. You and maybe some of your friends have a great idea for some open source software, so you go to SourceForge.net and create a new project. There is a flurry of project modifications as you

setup your friends as members, setup project permissions, create a project description, assign a project status and category, and other various organizational tasks. Some will then post a new forum message, possibly welcoming new users to the project; and lastly, a small few will actually have software that is posted as a file release. Then, nothing happens. We call this social position, Brief Flame, for the spurt of initial activity that quickly goes away.

After the largest cluster, we analyzed the next thirty largest clusters which, with the largest cluster, accounts for over 90% of all the activity in the community. Upon reviewing the activity distributions of those clusters, we subjectively combined clusters into general categories; Table 2 provides the results of our analysis with the categories corresponding to meaningful social positions for the Open Source Software Community. Note that the non-categorized item includes all of the remaining clusters that we did not analyze. Detailed analysis regarding how the social positions relate to software development practices, user and project lifecycle, and project success will be provided in a future paper.

Temporal Analysis of Social Positions

We consider a simple extension to our clustering algorithm in order to provide temporal analysis of social positions. The essential idea is to chunk the data into time intervals and run the clustering algorithm on each data chunk; thus, a time series of social positions is obtained which represents the evolution of social positions in the social network. In a stable network, one where the process that governs the underlying mechanism for the formation of the social network stays constant, we would expect the social positions to change little over time; though the actors who hold a specific social position may change as well as the quantity of actors. However, in a self-organizing or adaptive network, one where the underlying process is changing or possibly has a lifecycle associated with it, we would expect the social positions to change; furthermore, the changing social positions should correlate in some manner to the underlying process governing the social network.

Performing temporal analysis of social positions requires that we consider more closely our conceptualization of time in relation to the social network as it determines how we chunk the data for analysis. One perspective is global (absolute, wall-clock) time which looks at the social network as a whole from time period to time period, so one talks about the state of the network in 2003 compared to the state of the network in 2004 or what changes occurred in February versus what occurred in January. Nodes and edges are added or removed or their attributes changed; global network measures are used to describe the changes, and contributions at the actor level tend to get washed out in the aggregate. The other perspective is local (relative) time that takes an egocentric viewpoint by looking at each actor in the social network; here each actor's first appearance in the social network becomes time 0, and structural and attribute changes are recorded from time period to time period. With the local perspective, the data does not generally correspond to any actual social network because actors join the network at different global times; instead, its focus on the individual actor allows egocentric measures to be analyzed in a time relative way to the overall social network. We argue that the local time perspective is more appropriate for the temporal analysis of social positions as it is an egocentric measure, so that is what we use for analysis of the Open Source Software Community.

We ran our algorithm for 113 time periods; we chose a time period of thirty days to roughly correspond to one month, so the total data set spans almost 10 years. Table 3 shows the results for the first four time periods which is all we currently analyzed; period 1 is the activity performed from local time 0 (time of first activity event for each user/project pair) or day 1 until day 29, period 2 is day 30 to day 59, and so on. The total number of user/project pairs for period 1 does not match the total in Table 2 because some activity records either had no date or had invalid dates, so that data was discarded. We ran our clustering algorithm on the data for each time period separately then grouped the largest clusters together into the social positions that we had previously described. Notice that for the first period we get results that correspond well to our analysis for activity over all time. By the second period, however, there is a significant shift; the Brief Flame social position is gone completely, giving credence to its name, and we see the introduction of a new social position called Handyperson. A Handyperson has significant activity in many activity types with no set of activities (which corresponded to the other social positions) that dominate; the activities performed by the Handyperson appears to be a combination of the Artifact Management, Message Posting, Release Management, and Task Management social positions. What is also significant is the large drop in total number of user/project pairs; this is a direct effect of the Brief Flame people becoming inactive after the first period. Likewise there is a significant drop in the Message Posting social position, presumably people who initially only post forum messages have either left the project or have started performing other activities which has shifted their social position. By the third period, the Handyperson social position has taken a clearly dominant role in the community. Also, people who do purely administration tasks has decreased significantly possibly indicating that the project has stabilized thus requiring few administrative tasks, or possibly the administrators have taken on other tasks to shift their social position. Lastly, by the fourth period, the results are less conclusive; the Handyperson

social position is now a combination of four clusters instead of just one as in previous periods thus indicating variation in the activity patterns. Also we see that Release Management and Task Management have increased; maybe this is an indication of a life-cycle shift for the project whereby it is entering strong implementation phase with tasks being created and completed with corresponding file release containing the changes.

Social Position	Period 1	Period 2	Period 3	Period 4
Brief Flame	127302	0	0	0
Message Posting	49754	1418	828	151
Administrators	10356	5415	905	496
Release Management	6304	1001	796	869
Task Management	3466	625	254	401
Artifact Management	1967	0	0	0
Documentation	1130	0	0	0
Job Posting	1125	0	0	0
Not Categorized	4904	2002	1313	1105
Handyperson		7282	8280	6664
Total User/Project Pairs	206308	17743	12376	9686
Total Clusters	397	183	143	139

Table 3: Temporal Social Positions in the Open Source Software Community

Conclusion

Our algorithm which performs clustering based upon a statistical test avoids many of the implicit assumptions made by standard data mining algorithms; assumptions which may not be valid when analyzing social networks. We have shown the usefulness of our algorithm by applying it to extract the social positions from activity data for the Open Source Software community at SourceForge.net without having to provide the number of clusters a priori. We have extended our algorithm to perform temporal analysis of social positions with the same data set; we have shown that our temporal analysis results correspond with our results for the static analysis, and we have also shown that such temporal analysis provides useful insights into the changes of social positions for the social network. There is considerable more analysis that can be performed with social positions for Open Source Software; tracing individuals to see when they shift social positions and under what circumstances or analyzing projects to determine what social positions make up a successful project are two examples of additional analysis. We hope to perform and present such analysis in a future paper.

Our algorithm is not without issues however; it suffers from one of the key problems that is also problematic for other clustering mining algorithms. Specifically because a statistical test, like a distance metric, does not satisfy the mathematical property of transitivity, it is not an equivalence relation, so the clusters it produces are not unique. The clusters produced are highly dependent upon the order in which the samples are compared; likewise, finding the optimal set of clusters is at least NP-hard so no easy algorithmic solution is available. We have run our algorithm with a few random permutations of the SourceForge.net data set and compared to the results presented in this paper; we get the same relative ordering of social positions but the actual number of user/project pairs differs slightly, so we are tentatively confident that our results are correct. For the future, we will investigate approximation and randomized versions of our algorithm that can provide a more accurate clustering.

References

- [Krauth, 2001] Joachim Krauth, *Distribution-Free Statistics: An Application-Oriented Approach*, Elsevier Science Publishers, Amsterdam, The Netherlands, 1988.
- [Hand et al, 2001] David Hand, Heikki Mannila and Padhraic Smyth, *Principles of Data Mining*, MIT Press, Cambridge, Massachusetts, 2001.
- [Sourceforge, 2003] <http://www.sourceforge.net>.
- [Wasserman & Faust, 1994] Stanley Wasserman and Katherine Faust, *Social Network Analysis: Methods and Applications*, Cambridge University Press, Cambridge, UK, 1994.
- [Witten & Frank, 2000] Ian H. Witten and Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Academic Press, San Diego, California, 2000.