STANFORD UNIVERSITY
Stanford Center for Professional Development

HOMEWORK ROUTE FORM

Stanford Center for Professional Development Student Information

Course No.	CS224W	Faculty / Instructor Name	Jure Leskovec	Date	December 10, 2013
Student Name	Arne Roomann-Kurrik			Phone	609-532-8563
Company	Twitter, Inc.			Email	kurrik@gmail.com
City.	San Francisco	State	CA	Country	USA

Check One: ○ Homework #: ○ Midterm ◉ Other Project Report

The email address provided on this form will be used to return homework, exams, and other documents and correspondence that require routing.

Total number of pages faxed including cover sheet 12

	For Stanford Use Only	
Date Received by the Stanford Center for Professional Development	Date Instructor returned graded project Score/Grade: (to be completed by instructor or by teaching assistant)	Date the Stanford Center for Professional Development returned graded project:

Please attach this route form to ALL MATERIALS and submit ALL to:

Stanford Center for Professional Development
496 Lomita Mall, Durand Building, Rm 410, Stanford, CA 94305-4036
Office 650.725.3015 | **Fax** 650.736.1266 or 650.725.4138
For homework confirmation, email scpd-distribution@lists.stanford.edu
http://scpd.stanford.edu

Last modified October 27, 2008

CS224w:
Social and Information Network Analysis

Assignment number: Project Report

Submission time: 9:30 AM **and date:** December 10, 2013

Fill in and include this cover sheet with each of your assignments. It is an honor code violation to write down the wrong time. Assignments are due at 9:30 am, either handed in at the beginning of class or left in the submission box on the 1^{st} floor of the Gates building, near the east entrance.
Each student will have a total of *two* free late periods. *One late period expires at the start of each class.* (Homeworks are usually due on Thursdays, which means the first late periods expires on the following Tuesday at 9:30am.) Once these late periods are exhausted, any assignments turned in late will be penalized 50% per late period. However, no assignment will be accepted more than *one* late period after its due date.

Your name: Arne Roomann-Kurrik
Email: kurrik@gmail.com **SUID:** 05809001

Collaborators: None

I acknowledge and accept the Honor Code.

(Signed) Arne Roomann-Kurrik

(For CS224w staff only)

Late periods: 1 2

Section	Score
1	
2	
3	
4	
5	
6	
Total	

Comments:

Building Recommendation Systems From Open Source Contributions - Project Report CS224W, Autumn 2013

Arne Roomann-Kurrik

December 10, 2013

1. Introduction

The rise of websites focused on hosting and facilitating the development of open source software has provided an interesting graph of collaborative software engineering activity. One of the most popular[1] such sites is github.com, which overlays social features such as networked user profiles, a "follow" model for projects, and an activity timeline on top of the git source control tool. GitHub's landing page proudly proclaims "Build software better, together". However, GitHub makes no attempt to grow its own network through its social features. It can be difficult to identify users with common interests and libraries which are commonly developed together, limiting the effectiveness of GitHub's ability to foster and encourage more OSS contributions.

This paper attempts to model Github's users and projects as nodes in a graph structure. By applying techniques for graph analysis and edge prediction, different approaches for social recommendations on Github are examined. By building a network of software contributions according to each model and then applying edge detection to each network, a recommendation system for social connections and related projects can be built.

2. Algorithms

Three different edge prediction algorithms were considered for the purpose of predicting connections between users and repositories in GitHub's network graph.

2.1. Apriori / AprioriTid

Agrawal & Srikant [1] presents a pair of algorithms for discovering associated products given a database of sales transactions, for example, finding rules such as diapers and formula are commonly purchased together. The algorithms Apriori and AprioriTid are presented along with a hybrid algorithm which scales linearly with the number of transactions in a given dataset.

The goal of the paper is to provide algorithms which produce *association rules* of the form $X \implies Y$. Such a rule has a *confidence* of c if $c\%$ of transactions which contain X also contain Y. The rule has *support* s if $s\%$ of transactions contain $X \cup Y$. The algorithms attempt to generate all rules given minimum levels of support and confidence.

The Apriori algorithm deals with building lists of candidate *itemsets* and then pruning candidates which do not have enough support in the transactions dataset. The process starts by building a list of all single-item itemsets with enough support. A list of itemsets is built up for each successive size of itemsets k by joining the list of $k-1$ itemsets with itself and pruning any resulting itemsets which do not have all of their subsets contained in the $k-1$ list. Then the resulting candidate list is checked for support against the dataset.

Agrawal & Srikant [1] offers several modifications of Apriori for performance or memory considerations,

[1] According to Alexa, GitHub is ranked #2 globally in the Computers > Open Source > Project Hosting category. http://www.alexa.com/topsites/category/Computers/Open_Source/Project_Hosting

but the underlying concept of building up a list of itemsets based off of smaller itemsets with enough support is shared.

Once all the itemsets with enough support are found, the process of generating association rules is fairly straightforward. To generate recommendations, a new transaction is compared against the list of association rules. If all of the *antecedents* of the rule are present in the transaction, then any *consequents* of the rule which are not present in the transaction are suggested.

2.2. Hierarchical Random Graph

Clauset et al. [2] posits that complex networks exhibit hierarchy as a central organizing principle, which explain observed topological properties. It models an existing network as a hierarchical dendrogram. Then a probabilistic model of the network is created where the connection between two nodes is based on their relative position in the dendrogram. Every node r in the dendrogram is associated with a probability p_r. Nodes in the network with lowest common ancestor r are connected with probability p_r. This model is referred to as a *hierarchical random graph*.

To determine the hierarchical structure of a real world graph, the hierarchical random graph model is fitted to observed data. MLE and a Monte Carlo sampling algorithm are applied on the space of all possible hierarchies in order to obtain the best fit for the real world data, typically generating a set of dendrograms. This set can be combined to form a *consensus dendrogram*, capturing common topological features of the set.

Edge prediction is accomplished by using the consensus dendrogram. Pairs of nodes which have a high probability of connection but are not connected in the observed graph become the candidate set for missing connections.

The paper fits a dendrogram through a maximum likelihood estimate. The maximum likelihood for a given dendrogram D is given as:

$$\log \mathcal{L}(D) = -\sum_{r \in D} L_r R_r h(\overline{p}_r)$$

Where the probabilities which maximize this likelihood at each p_r are :

$$\overline{p}_r = \frac{E_r}{L_r R_r}$$

E_r is the number of edges in the graph who have r as their lowest common ancestor in the dendrogram and L_r and R_r are the number of leaves in the left and right subtrees rooted at r. A set of dendrograms are sampled by starting with a random starting dendrogram and applying transitions to the dendrogram which are accepted according to the Metropolis-Hastings rule as given in Newman & Barkema [3]. This Monte-Carlo approach converges in around $O(n^2)$ steps at which point additional dendrograms are sampled from the Markov chain to create the consensus dendrogram.

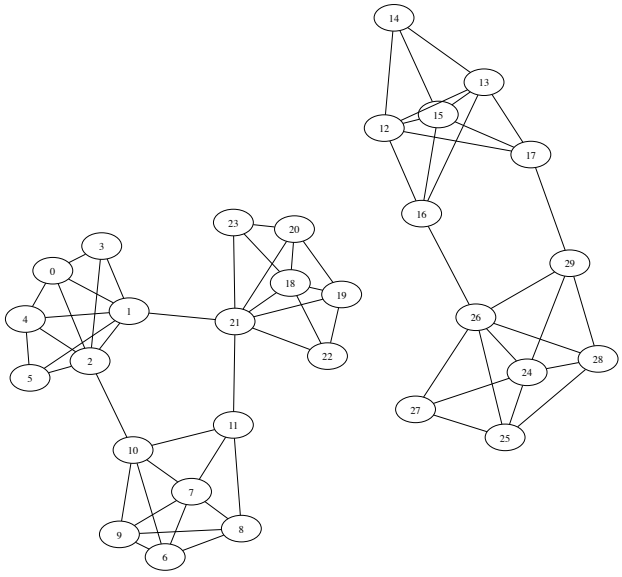

Figure 1: Network generated by connecting 5 preferential attachment clusters

The hierarchical random graph model does a reasonable job of representing clusters of well-connected nodes in a graph structure. For example, the example network shown in Figure 1 represents 5 small clusters (connected through a preferential attachment process) which have been connected to each other through the addition of random edges.

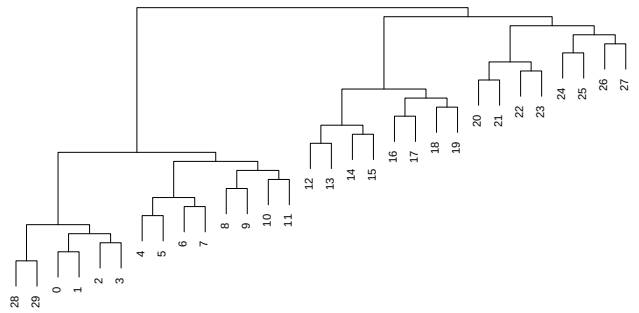

Figure 2: Initial hierarchy for clustered network

The initial random dendrogram configuration is shown in Figure 2. With this kind of network the log-likelihood growth plateaus after a small number of Monte Carlo iterations, as demonstrated in Figure 3.

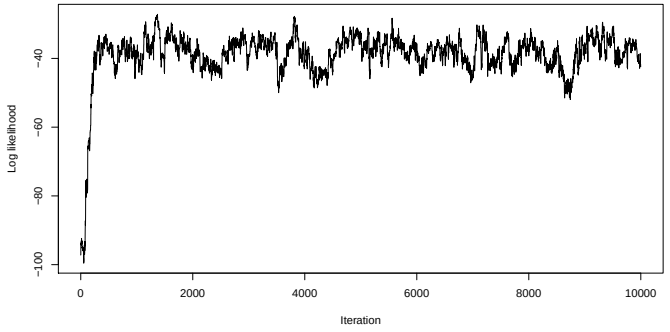

Figure 3: Log-likelihood growth on clustered network

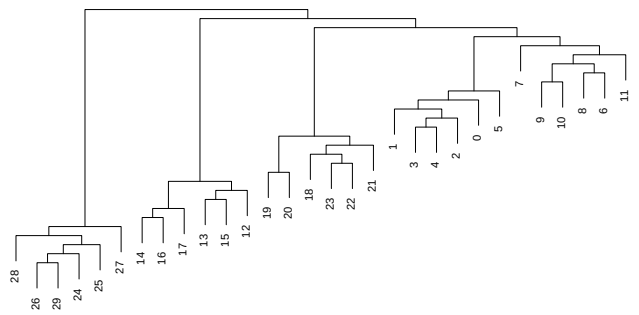

Figure 4: Clustered hierarchy after 10,000 Monte-Carlo changes

The final state of the dendrogram is shown in Figure 4 and the five clusters from the original graph are plainly visible, sharing close parent nodes with each other member of the individual clusters. Obviously the components of the network structure are represented using this model.

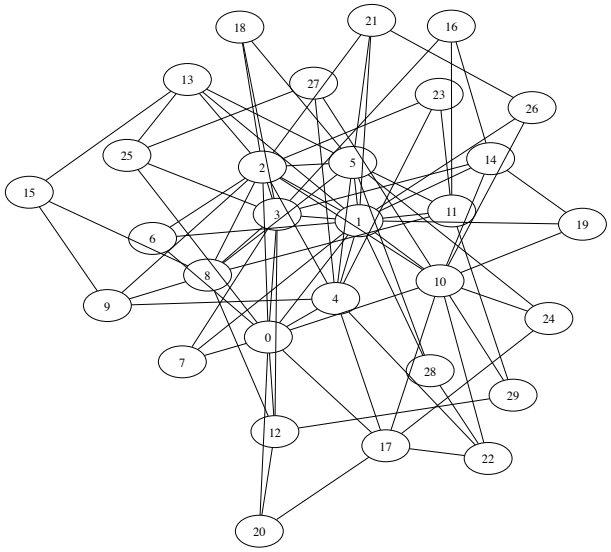

Figure 5: Network generated through preferential attachment

Unfortunately not every type of graph is well served by Hierarchical Random Graphs. For example, the paper authors specifically mention overlapping clusters as an area where the algorithm may not work well. GitHub itself seems to be similar to a preferential attachment-style network where a power law governs the distribution of node degrees. Some projects are especially popular (Bootstrap for example) and there's a very long tail of projects and users with few connections. To examine how such a network is represented by a hierarchical random graph, the network in Figure 5 was created using a standard preferential attachment approach.

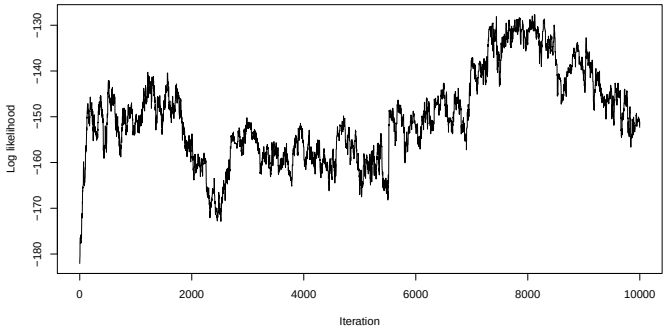

Figure 6: Log-likelihood growth for preferential attachment network

The log-likelihood growth for this network is shown in Figure 6. It did have some initial sharp gains but did not plateau in 10,000 iterations like the previous example, despite having a similar number of nodes.

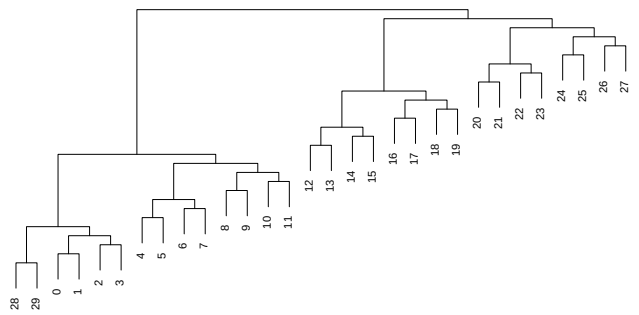

Figure 7: Initial hierarchy for preferential attachment network

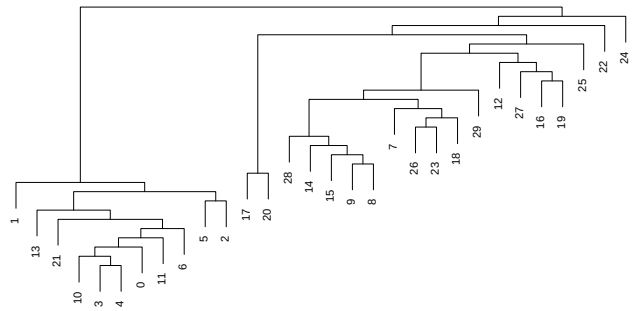

Figure 8: Clustered hierarchy after 10,000 Monte-Carlo changes

The pre- and post-Monte Carlo dendrograms are shown in Figure 7 and Figure 8, respectively. The change is more pronounced in this example. Interestingly, the post network seems to have arranged in favor of unbalanced nodes, where a dendrogram cut will peel off a single node at a time. This contrasts Figure 4, where the dendrogram cuts tend to be balanced on both sides. This "peeling" of nodes is likely due to the very well-connected center of the graph, where the nodes requiring the fewest cuts to remove are at the periphery of the network.

2.3. Logistic Regression

Taskar et al. [5] deals with the problem of identifying related entities through analysis of graphs with heterogenous link types. Inferring hierarchical structure (such as professor -> student) from simple links is mentioned, as well as link prediction.

A *relational Markov network* is used to generated a probabilistic model of a link graph to predict and classify links in the graph. This model relies on *relational clique templates* to describe cliques of connected nodes. A template might specify a relation between the categories of two web pages connected by a hyperlink, for example. An unrolled Markov network is generated and gradient descent is used to fit a set of feature weights to the training data.

Because the clique templates relate the labels of the entire network, fitting weights involves the exponential sum over every possible configuration of the clique. This is not tractable for even small networks, so belief propagation is used to compute a posterior distribution over the label variables.

Even by following the algorithms presented in Taskar et al. [4] and Yedidia et al. [6], it was not possible to converge a set of weights for GitHub data. The RMN approach gives much flexibility but at the cost of simplicity and speed.

As a fallback, the logistic regression model used as a baseline in Taskar et al. [5] was implemented. Given that RMNs showed performance gains over this model, it serves as an upper bound of how a RMN may perform for link prediction. In particular, features mentioned in Taskar et al. [5] for predicting social connections between students in student housing were adopted - specifically, the number of shared connections between any two nodes, as well as the relative degree of the node (how well connected it is to the rest of the graph).

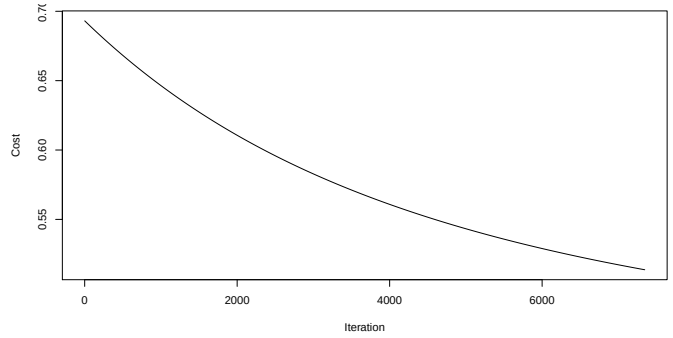

Figure 9: Gradient descent over repo-repo edges.

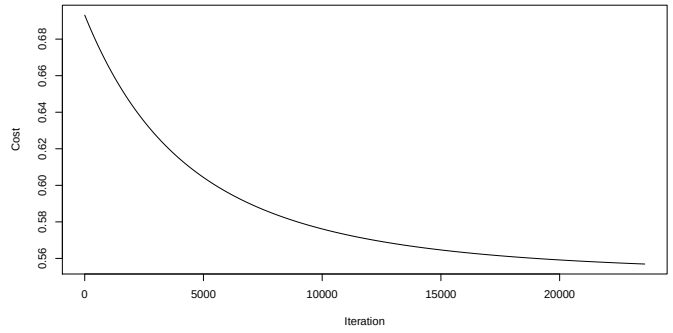

Figure 10: Gradient descent over user-user edges.

Gradient descent was run until the change fell below a given threshold. Examples of this growth for repo-repo edges and user-user edges are given in Figure 9 and Figure 10, respectively.

3. Model / Data Collection

The GitHub dataset is readily available. An attempt to make the public timeline available for research is available at http://www.githubarchive.org/. This dataset is also available in Google's BigQuery[2] service, which provides convenient querying functionality for ad-hoc calculations, and covers the entirety of Github's public timeline. The data presented in this set is in the form of events, which typically associate a user with a repository and a specific action label, such as *pushed to*, *forked*, *starred*, or interacted with an *issue* of.

3.1. User / Repository Mapping

Instead of dealing with strings, it was preferrable to assign a numeric ID to each repository. This was easily done in BigQuery:

```
SELECT
  ROW_NUMBER() OVER (
    ORDER BY repository_url ASC
  ) AS repository_id,
  repository_url
  FROM [githubarchive:github.timeline]
  GROUP EACH BY repository_url
```

Similarly, users were assigned integer IDs in order to reduce the amount of memory needed to store the entire graph in memory. However, as of November 7, 2013, this dataset contained the actions of $1,801,079$ Github users, representing all public GitHub activity. This proved to be too much data for all but the Apriori algorithm to handle, so the dataset was split up into smaller pieces for analysis.

3.2. Dataset Partitions

The dataset was broken up into partitions by querying over event attributes in the GitHub dataset and selecting subsets of events to build networks out of. The `golang_recent` and `ruby_recent` datasets were built out of events which dealt with specific programming languages. The `top_*_repos` datasets select events pertaining to "top" repositories, as indicated by number of pushes or number of times starred.

Dataset	Users	Repos
golang_recent	5,713	11,481
ruby_recent	34,655	53,668
top_pushed_repos	1,204	100
top_starred_repos	516	100

3.3. User/Repository Transactions

The Apriori algorithm operates on lists of transactions, where a transaction represents a list of items which are acted on in a group. A Github user's history of interactions with various projects was used as the model for a transaction, so that the list of repositories a user contributed to counted as a transaction containing all of those repositories.

Eventually the data was massaged into a list of `user_id`, `list[repository_id]` pairs such as:

```
1234,"2624791,2624780,...,1344521"
```

The list of users who had interacted with a repository were also considered a transaction of users.

3.4. Repository/Repository and User/User

The Hierarchical Random Graph model operates on unlabeled graph edges, so it was suitable to join the user/repository table with itself to produce edge lists for both users and repositories.

3.5. Feature Vectors

To train the Logistic model, a training set of all possible edges was generated, and then matched against the edges generated in Section 3.4 to produce exists/does not exist labels.

In addition to edge labels, it was important to produce feature vectors to train the model. From Taskar et al. [5], the percent of shared neighbors between two

[2] https://bigquery.cloud.google.com/

nodes was computed, as well as the percent of the network the node links to. These were added as simple feature vectors.

4. Performance Analysis

4.1. Success Metrics

To objectively measure the impact of the recommendation systems, the Matthews correlation coefficient (listed in 1) was observed, along with accuracy (2) and precision (3). MCC is a reasonable measure in cases where the labeled classes are of different sizes and is a handy single value analog for the performance of a classifier.

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{1}$$

$$\frac{TP+TN}{TP+FP+FN+TN} \tag{2}$$

$$\frac{TP}{TP+FP} \tag{3}$$

When scoring Apriori, a *True Positive* was identified as a rule where both antecedents and consequents existed in a given transaction. A *False Positive* occurred if the antecedent existed in the transaction, but some or all of the consequent did not.

For a given transaction, a *False Negative* occurred if, after examining all rules, no positive match was made, but at some point a rule with consequents which existed as a subset of the transaction was processed. A *True Negative* occurred if no rule had consequents which were subsets of the transaction.

When scoring Hierarchy, the most likely edges in the dendrogram were compared against the existing edges in the graph. The least likely were compared to evaluate negative answers.

The Logistic algorithm already had labels built into its dataset, so *True,False/Positive,Negative* scores were simple to compute.

4.2. Cross Validation

K-fold cross validation was used to train and test the recommendation systems. Datasets were split into $k = 4$ pieces, and the analysis was performed k times, each time holding a dataset in reserve to use for testing.

4.3. Apriori Parameter Selection

The Apriori algorithm has two main tunable parameters named `minsup` and `minconf` which represent the minimum thresholds for support and confidence for a rule to be valid.

A reasonable goal of a recommendation system is to provide recommendations for all users. Several runs of the Apriori algorithm were staged across the entire GitHub dataset to see how the performance varied according to the metrics described in Section 4 but the number of rules were also graphed, to see how each parameter affected the number of rules which could generate suggestions for users.

When choosing `minconf`, generally higher is better up to the maximum confidence of 100%, as shown by its effect on all of the scored metrics in Figure 11, Figure 12, and Figure 13.

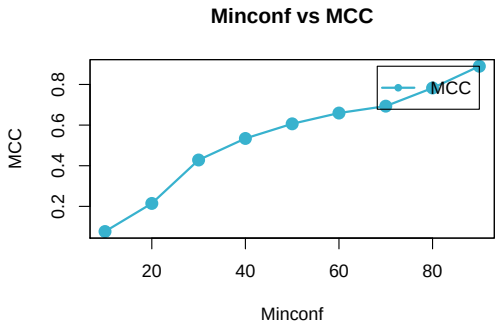

Figure 11: Effect of minconf on MCC, minsup = 100

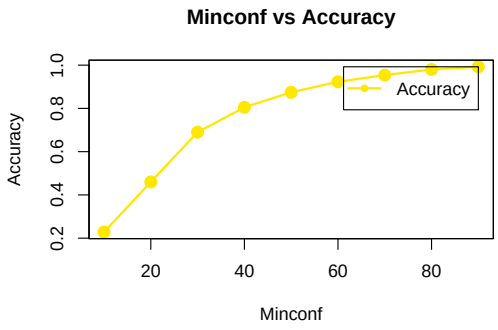

Figure 12: Effect of minconf on Accuracy, minsup = 100

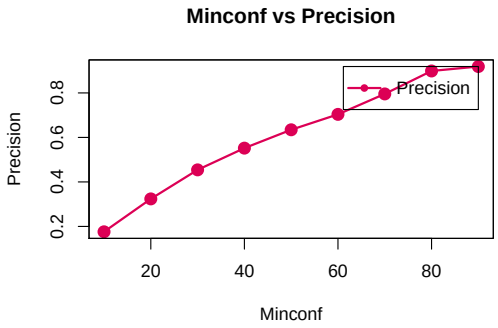

Figure 13: Effect of minconf on Precision, minsup = 100

However, the raw number of rules generated drops as `minconf` increases as shown in Figure 14, even generating no rules for smaller datasets. Therefore, it was required to select an appropriate value for each dataset in order to generate enough rules to be useful.

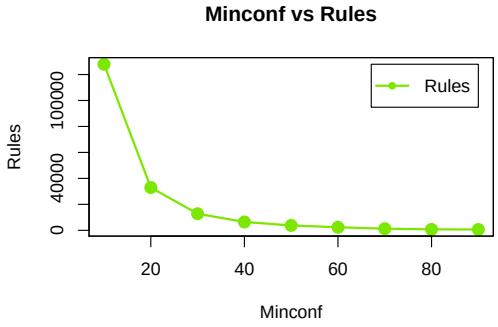

Figure 14: Effect of minconf on rule count, minsup = 100

The `minsup` parameter generally performs better when it is smaller. However, the number of rules increases dramatically when `minsup` crosses a minimum threshold, as every pair of items becomes a rule. This is shown in Figure 15.

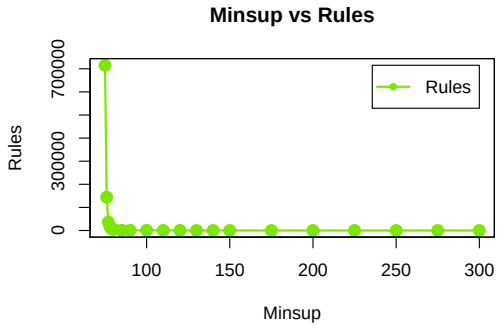

Figure 15: Effect of minsup on rule count, minconf = 80

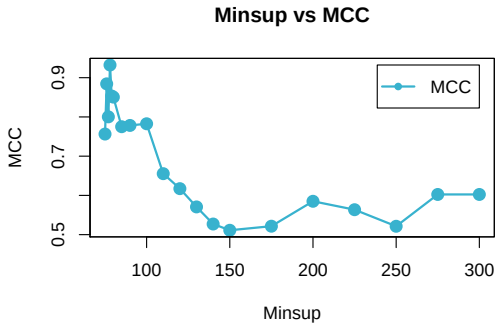

Figure 16: Effect of minsup on MCC, minconf = 80

Therefore, a value of `minsup` which be as small as possible while still generating a manageable number of rules was chosen for each dataset.

5. Recommendations

Two recommendation products were designed:

5.1. Projects you may want to contribute to

The result of predicting edges on the Repo/Repo edge graph will be to build a list of projects a given user may want to consider interacting with.

For Apriori, the generated rules are applied to the user's existing project list. Rules generate a list of candidate projects.

Hierarchy assigns a probability to all combinations of nodes. Once the hierarchical model is trained, then selecting the highest probability edges between the

user's repositories and other repositories is possible.

Logistic regression requires a bit more work to predict, as the shared neighbors feature will need to be calculated between all of the user's repositories and the rest of the network.

5.2. Developers you may wish to follow

When operating on the User/User edge graph, the result of edge prediction will be to suggest users a given user may wish to interact with.

Aside from the differences in sizes between the Repo/Repo graphs and the User/User graphs, there is not a fundamental difference between either recommendation approach.

6. Results

The results indicate that there does not appear to be a silver bullet for GitHub recommendations. Looking at the MCC score of the repository recommendations (Figure 17) the Apriori approach did much better on the larger per-language datasets than the smaller "top-N" datasets. It seems to indicate that Aproiri is not only fast enough to handle larger sets of data, but that it performs better on a larger dataset as well.

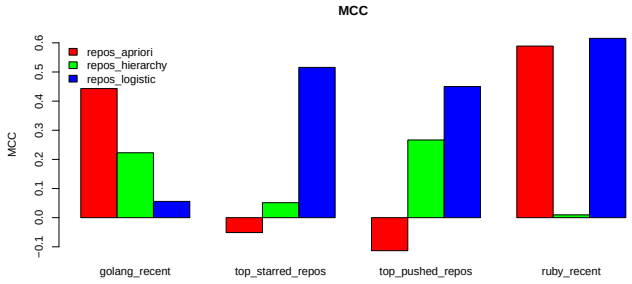

Figure 17: Repo/Repo suggestions vs. MCC

Simple logistic regression performed well on all but one dataset, but that may be due to overfitting and/or the small number of features in the model used.

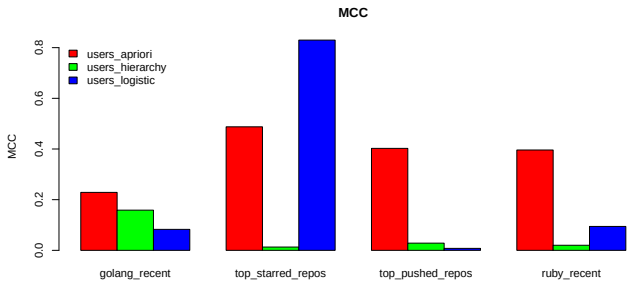

Figure 18: User/User suggestions vs. MCC

Looking at the MCC scores of user recommendations (Figure 18), the Apriori algorithm seemed to do reasonably well across all datasets, likely because there were more users than repos in the small "top-N" datasets. Logistic regression was surprisingly inconsistent across these datasets.

The least promising algorithm across both user and repository recommendations turned out to be the Hierarchical Random Graph. Given the analysis in Section 2.2, it seems that the GitHub network is not easily represented by the dendrogram. Given the long training times of the algorithm and its inability to model overlapping clusters, it seems like a poor fit for GitHub data (despite being pretty interesting and fun to implement).

The accuracy and precision scores for repository and user suggestions are shown in Figure 19 and Figure 20, respectively. These scores do not really demonstrate much other than relatively consistent prediction and accuracy scores for each algorithm across each dataset. The MCC score is more interesting from an algorithm performance point of view.

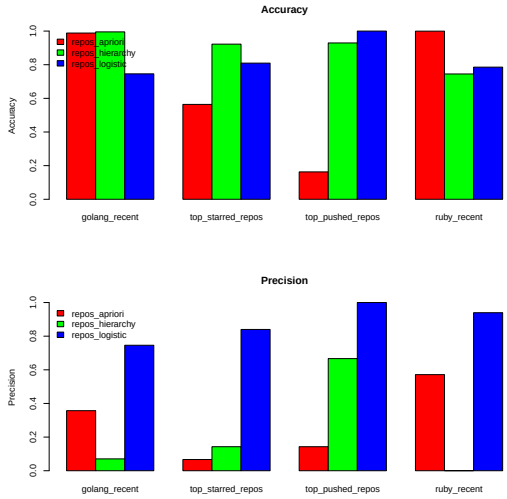

Figure 19: Repo suggestions

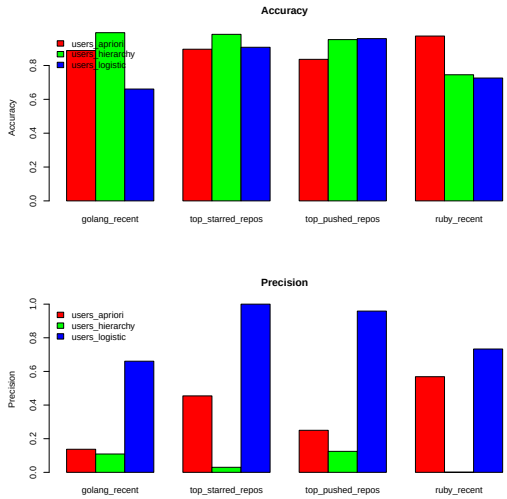

Figure 20: User suggestions

7. Further Work

It would have been desirable to produce a recommendation system for any given user or repository in the Github network. However, the only algorithm capable of operating over the entire graph was Apriori. Therefore the goal of developing a simple "do these recommendations make sense for account X" sanity check system was not feasible. Seeing how Apriori managed to do relatively well on the larger datasets, it may be good enough to create such a system out of a single Apriori run over all the Github data.

Overall, no model performed so well that a breakthrough recommendation system for GitHub seems likely out of the models tested in this work. To continue investigation into these kinds of suggestions, improvement to the models will be necessary.

The Logistic model performed inconsistently, so more tuning is possible. Adding additional features or combinations of features would probably help the classifier. The Relational Markov Network model seems very promising in that it can capture complicated relationships in networks, so further attempts to train weights using that approach would likely have merit.

Good models to investigate would seem to be any which are able to model overlapping clusters of users or repositories. This corresponds with the intuition that a GitHub user may contribute to different circles of projects. For example, imagine a web developer who contributes to a set of JavaScript-based client libraries, but also writes game framework code in C during their spare time. Prolific programmers do not seem to typically constrain themselves into singular, well-defined clusters all of the time.

References

[1] Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proc. of the 20th VLDB Conference*.

[2] Clauset, A., Moore, C., & Newman, M. (2008). Hierarchical structure and the prediction of missing links in networks. *Nature*.

[3] Newman, M. E. J., & Barkema, G. T. (1999). *Monte carlo methods in statistical physics*. Clarendon Press, Oxford.

[4] Taskar, B., Abbeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. *UAI'02 Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*.

[5] Taskar, B., Wong, M., Abbeel, P., & Koller, D. (2006). Link prediction in relational data. *Neural Information Processing Systems (NIPS)*.

[6] Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2000). Generalized belief propagation. *TR2000-26*.