CS 410 Course Project: Improving a System - Collaborative Filtering with Social Exposure: A Modular Approach to Social Recommendation

Captain: Philip Cori (pcori2), Team Member 1: Henry Moss (htmoss2), Team Member 2: Kyle Maxwell (kylem6)

Collaborative Filtering with Social Exposure: A Modular Approach to Social Recommendation

Overview

Our project is improving the recommendation algorithm discussed in this paper. To improve this system, we built on an existing recommendation system framework called RecQ. This system provides a host of different recommendation algorithms and datasets to test them on, including the one proposed in this paper.

The Paper

Key Excerpt from the Abstract

This paper is concerned with how to make efficient use of social information to improve recommendations. Most existing social recommender systems assume people share similar preferences with their social friends. Which, however, may not hold true due to various motivations of making online friends and dynamics of online social networks. Inspired by recent causal process based recommendations that first model user exposures towards items and then use these exposures to guide rating prediction, we utilize social information to capture user exposures rather than user preferences. We assume that people get information about products from their online friends and they do not have to share similar preferences, which is less restrictive and seems closer to reality.

Relevant Summary

Of the two methods presented in the paper, we focused on the method they describe as *social boosting.* We will refer to this algorithm as SERec and more specifically, SERec Boost. The primary assumption leveraged by social boosting is that a user's exposure to an item is boosted by their friends. People are likely to receive information about a product from friends' discussion and shared feelings. As such, a consumer is more likely to have exposure to an item if their friends have interacted with the item. This can help the model disambiguate between the situations in which a user did not interact with an item because they dislike the item, or they did not notice the item.

SERec has two main components, the Rating Component and the Social Exposure Component. The Rating Component is a matrix factorization model for rating prediction. The Social Exposure component calculates the exposure priori for each user item pair. The modularity of the system

lends itself to extensibility and improvement. As suggested in the conclusion of the paper, the system presents itself with multiple avenues for exploring alternative methods for matrix factorization, and for integrating social exposure information.

Software Implementation

For information about the implementation and architecture of the entire RecQ system, please refer to the original repository. Our implementations build on the architecture of this system by adding our custom algorithm extensions to the RecQ framework.

Development Environment Setup

1. Install miniconda
2. Create a new miniconda environment with Python 2.7
3. Run `conda install mkl-source`
4. Clone this repository
5. Download the lastfm dataset and add it to a folder named “dataset” within the repository
6. Activate the environment and install dependencies with `conda install --file requirements.txt`
7. Run `python main.py`
8. Follow the prompt and entire the desired algorithm to run

Usage

The original SERec boost algorithm discussed in the paper can be run by inputting the corresponding value ("s10") for the SERec algorithm. Similarly, each of our individual extensions can be run by inputting the corresponding value displayed in the prompt. Users can compare results with the baseline results of the original SERec boost algorithm:

Evaluation statistics are generated for a recommendation list of length 100.

Precision	0.0479560590416
Recall	0.485595079529
F1	0.0872914462838
MAP	0.156959948956
NDCG	0.357199124032

Team Member Contributions

Philip Cori

Algorithm description

I implemented an algorithm for measuring friend closeness, which is a suggested extension mentioned in the original paper. It can be run by inputting “s11” upon running main.py. To measure friend closeness, I use the formula:

closeness = n / m , where n is the number of mutual friends between the two friends, and m is the total number of friends between the two friends.

The intuition behind this is that it would seem two friends are closer friends if they share many mutual friends. More mutual connections should imply more personal interaction, exposure, and therefore closeness. Furthermore, two friends are considered closer if they have less total friends, which gives more weight to their own friendship.

Results

Three different formulas were tested based on the above idea.

1. *Closeness = n / m * 10*

Precision	0.0478126532299
Recall	0.48445040571
F1	0.0870353717949
MAP	0.157030933278
NDCG	0.35660261507

Mean	1.177611443146
Median	0.666666666666
Stdev	1.583484225324
Min	0.072463768115
Max	10.0

It can be seen that unfortunately the results did not improve from the above implementation. All measures are within 0.04% of the baseline results. From further analysis, I found that only 9.4% of friendships consist of friends with any mutual friends. This low percentage can partially explain why adding a closeness measure may not have much impact, since it depends largely on the number of mutual friends as the determining factor. On the right certain statistics are shown to give an idea of the distribution of the closeness between friends. It seems that there is a fairly wide distribution of closeness measures, which could lead to unstable results as well.

2. *Closeness = n^2 / m * 10*

Precision	0.0473455166473
Recall	0.479583256631
F1	0.086182863339
MAP	0.15522424781
NDCG	0.353131761035

Mean	1.3374456997343211
Median	0.7142857142857142
Stdev	1.965461767041007
Min	0.07246376811594203
Max	39.67032967032967

Next, I tried squaring the number of mutual friends. The logic behind this was to give this factor even more weight, such that the closeness benefits quadratically with the more mutual friends they have. However, it seemed to diminish our results slightly further. It likely over-weighed some closeness terms, causing other friend exposures to be dominated by relationships with even only a few mutual friends.

3. *Closeness = $log_2(n/m + 1) * 10$*

Precision	0.0484710617097
Recall	0.491026838687
F1	0.0882323703985
MAP	0.158975281865
NDCG	0.360987047292

Mean	0.920228915514409
Median	0.7369655941662061
Stdev	0.6752581440071108
Min	0.10092890885078087
Max	3.4594316186372978

Given that squaring mutual friends further decreased results, I tried using a log transform that instead introduces diminishing returns from a higher closeness score. It can be seen that the standard deviation of the closeness measure is much less and every connection is being treated more equally. I add 1 to n/m to prevent any closeness measures from becoming negative. The results now are actually a slight improvement over the original baseline results. Although precision deteriorated slightly, NDCG, MAP, F1, and Recall improved by 1.06%, 1.28%, 1.08%, and 1.11% respectively.

Conclusions

From these results, it can be concluded that using mutual friends and total number of friends can accurately model the closeness between two friends. However, as discovered by experimenting different transformations of this idea, it is important not to smooth this measure slightly and not overweight certain connections. A dataset that would yield even better results

for the introduced formula would contain a more dense social network where more mutual friends are present, as well as a pattern that friendships with a higher n/m measure do in fact imply a stronger correlation between the way two friends “rank” items (ie. artists in the Lastfm dataset).

Henry Moss

For this project, I tried to evaluate how we could incorporate social contagion into the existing recommendation algorithm, as this was one of the last suggestions in the original paper for further improvement. Unfortunately, after a lot of research into social contagion, it seems to be a fairly arbitrary concept and difficult to measure. The recommendation algorithm is already trying to calculate how much one user is potentially influenced by the friends they are in contact with, which is a simple definition of what social contagion is.

Additionally, I helped Philip experiment with his algorithm for measuring friend closeness. After trying to find a way to implement TF-IDF weighting into our algorithm, I hypothesized that it could be beneficial to have diminishing weight on increased number of friends, so that it values closer friends at an increased rate. This boosted our NDCG values from around 0.356 to 0.36. After that, I tried other rates of diminishing return with different log powers of log_3 and log_{10}, along with Philip’s idea to prevent any of the values from being negative, but found that neither were as successful as log_2.

1. *Closeness* = $log_3(n/m + 1) * 10$

Precision	0.0339437367304
Recall	0.344395578472
F1	0.0617967648459
MAP	0.0755357909461
NDCG	0.216285701749

Mean	0.9437401212835262
Median	0.5874549356790257
Stdev	1.07063674944364
Min	0.06572152931440885
Max	6.309297535714574

2. *Closeness* = $log_{10}(n/m + 1) * 10$

Precision	0.0315127388535	Mean	0.4553789242585401
Recall	0.319221388131	Median	0.2802872360024353
F1	0.0573627683574	Stdev	0.5136685928528778
MAP	0.0546393023159	Min	0.03135713852858582
NDCG	0.182298909362	Max	3.0102999566398116

These were both unsuccessful ideas, as using log3 decreased the MAP and NDCG scores by 51.8% and 39.5%, while using log10 decreased the MAP and NDCG scores by 65.2% and 48.9%, respectively.

I also looked into social structural influence and decided that one way to try to implement this into our project was by stretching the data to include friends of friends, in addition to just counting the closeness of direct friendships. I tried changing n, which was originally the number of mutual friends between user 1 and user 2, to also include any mutual friends of user 2's friends. This expands the network out a degree, and I was hoping that with more data to work with, the recommendation algorithm could be more successful.

3. *Closeness = n / m * 100, where n is the number of mutual friends and friends of friends*

Precision	0.0446282847314	Mean	11.607520985265921
Recall	0.452188532552	Median	6.896551724137931
F1	0.0812387859384	Stdev	15.32296991067004
MAP	0.144929322365	Min	0.5681818181818182
NDCG	0.332269062625	Max	100.0

As you can see, this slightly lowered our MAP and NDCG score, which we have been using as our main measurements of improvement. I experimented with a few other factors into the algorithm, such as using log_2 again, but these were the best results I came up with overall.

Kyle Maxwell

Algorithm description

The potential improvement I explored was the modification of the matrix factorization model in the Rating Component of the SERec Boost algorithm. By incorporating the Weighted Rating Matrix Factorization methods from Collaborative Filtering for Implicit Feedback Datasets (Yifan Hu et al, KDD 2009), I was able to achieve small but successful improvements.

This matrix factorization latent factor model uses implicit feedback data as an indication of positive and negative preference associated with varying confidence levels. Meaning, it not only models the user's preferences of an item, but models the probability that they have consumed the item. I utilized this implicit feedback model to first converge on the user and item latent factor vectors without any use of the social connectivity information.

Then, I used the SERecBoost Social Exposure Component to update the social exposure prior based on those latent factors. Finally, the original SERecBoost algorithm is initialized with the pre-trained latent factors, and only requires a small number of iterations for the expectation-maximization algorithm to adequately converge.

Results

Statistics at 100 Recommendations, 5-fold cross validation

Precision	0.0492
Recall	0.4981
F1	0.0895
MAP	0.1657
NDCG	0.3705

Conclusions

As suggested in the original Collaborative Filtering with Social Exposure, there is still room for additional work and improvement. By utilizing other novel matrix factorization techniques, we were able to achieve 5.6% MAP and 3.7% NDCG score increase.