Github Stars Recommender System

Alasdair Smith, Ed Poole
{alasdair.smith, edward.poole}@myport.ac.uk

School of Computing, University of Portsmouth, Buckingham Building, Lion Terrace, PO1 3HE

1. Abstract

A collaborative filtering approach to providing recommendations to Github users was developed to investigate the usefulness of Github stars as the basis for a recommender system. Useful recommendations are found to be generated by the item-to-item collaborative filtering algorithm.

2. Introduction

Github is a web hosting service for software development projects and is the largest of it's kind in the world. It hosts a wide variety of code, ranging from small, private repositories all the way up to large, open source projects. As of April 2013, Github has over 3.5 million users and 6 million repositories (Wauters, R, 2013).

Github adds a social layer to code hosting, allowing users to follow updates from other users and “starring” other’s work, known as “repos”, a unary rating system. Announced in August 2012, Github says that stars allow users “to keep track of repositories that you find interesting, even if you are not associated with the project” (“Stars · GitHub Help”, 2013; “Notifications & Stars”, 2012). By using these stars, it is possible to recommend different repositories to users, the concept the authors decided to study.

There are several reasons to believe that Github stars have the potential for a useful recommender system. There are few existing services in this area; although Github offers a “trending” repository service, it does not provide recommendations for individual repositories. It is used by users to mark useful or interesting repositories. Finally, there is wide adoption of the star feature, with popular repos gaining tens of thousands of stars (https://github.com/twbs/bootstrap).

This paper will present a concept for recommending Github repositories to Github users. By using item-to-item collaborative filtering, it is possible to recommend repositories based on the repository the user is currently viewing. This is similar to Amazon's "other customers have bought" recommendation feature.

3. Research

Recommender Systems are filtering systems that predict how users will interact with certain items or products. Mostly used in an eCommerce setting to recommend other products to users, they have become more widespread in places such as social media. There are two main types of collaborative filtering - content-based and collaborative filtering.

3.1. Content-based Filtering

Content-based filtering recommends items to users based on what the user has previously liked. By comparing similarities between products a particular user has rated, it is possible to generate a list of products that are likely to be of interest to the user. However this method of filtering requires knowledge of the product that is being recommended (Balabanović, M., & Shoham, Y., 1997). A benefit to this technique is that the system does not require a large collection of users for an accurate recommendation. It is completely independent of the user base (Jannach, D., & Gerhard, F., 2011).

3.2. Collaborative Filtering

Collaborative filtering is the most prominent type of recommender system. It is used in large eCommerce platforms such as Amazon to recommend items to users. Unlike content-based filtering, collaborative filtering algorithms know nothing about the actual product they are recommending. The recommendations generated are completely based on user-item relationships. Collaborative filtering can be split into two areas - user-based filtering and item-based filtering (Sarwar et al., 2001).

3.2.1. User-based Collaborative Filtering

User-based collaborative filtering works by finding similarities between certain types of users. By grouping similar users together a system can predict the behaviour of a particular user.

3.2.2. Item-based Collaborative Filtering

Item-based collaborative filtering was invented by Amazon and it calculates its recommendations using the idea of "users who bought x also bought y". Due to the nature of Github's star rating system, the most appropriate recommendation algorithm to use is a simplified item similarity computation. To recommend similar items, first isolate users who have rated both items. From there, an algorithm can be applied to calculate the similarity between both items.

As discussed above, Github stars are a unary rating system, where user ratings can only exist in two states: starred, or not starred. The latter does not necessarily imply that the user did not like the repository, they may not have seen that particular instance, or didn't like it enough to star it. The algorithm must be able to cope with this limitation.

The cosine similarity calculates how similar two items are by treating them as vectors, and then measuring the cosine of the angle between them. As seen in Figure 1, i and j are vectors each containing the ratings for a particular product. In Figure 2, i and j have been expanded to actual values. As the rating metric is unary, the values can only be 1 or 0. This calculation is completed for every pair of items - from there a list can be drawn of the most similar to least similar items.

$$sim(i,j) = \cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{||\vec{i}||_2 * ||\vec{j}||_2}$$

Figure 1: The Cosine Similarity

$$\frac{(1,1,0,1)\cdot(0,1,1,0)}{\|(1,1,0,1)\|\,\|(0,1,1,0)\|} = \frac{1}{\sqrt{3}\,\sqrt{2}}$$

Figure 2: Calculating the Cosine Similarity

4. Implementation

A user's starred repository data is freely accessible from the Github API. This allows easy programmatic access to user data (including "starred" data) and repository data. Github's API requires OAuth, a widely used open authentication protocol, for applications that use more than 5000 requests per hour. A Laravel package named `oauth-4-laravel` was used to provide OAuth authenticated requests to the API (https://github.com/artdarek/oauth-4-laravel). User permission to access OAuth is needed before an access token is given, which is again handled by the OAuth package.

The implementation that was chosen was a variation of the item-based collaborative filtering algorithm, which uses cosine similarity as the similarity metric. The algorithm compares a given repository to other repositories using the starred data from users that have starred the given repository. In other words, "users who liked this repository, also like that repository" and are therefore recommended those repositories.

4.1. Dataset Creation

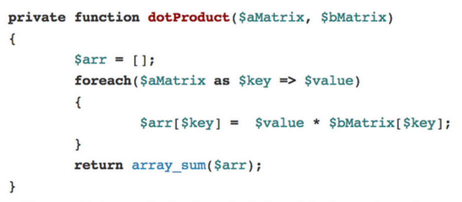
A proof-of-concept application was built, that creates a sample dataset using the Github API and then performs similarity calculations to provide recommendations within the dataset. A database structure was created to hold the dataset, containing three tables: `repos`, to hold data related to individual starred repositories; `users`, to hold data related to individual users; and `stars`, a join table to hold the relationships between `users` and `repos`. The `stars` table simply contains the `user_id` and `repo_id` for each star, thereby holding the item data.

The initial iteration of the application used a whitelist of users to construct the dataset. For each of these users, a request was made to the API for their list of starred repositories. This data, along with user data was stored in the database. However this solution was not optimal, as there was little "overlap" between the user's stars. Only 5 repositories were starred by more than one person, out of over 100 repositories. This is known as the *cold start problem*, and is prevalent in recommender systems. (Schein et al., 2002, p253).

This problem was mitigated in the second iteration, by using a single repository that every user has starred. In a real-world application, this could be any given repository on Github, as the dataset would be the entirety of the service. In this proof-of-concept however, a repository named `var_dumpling` by user *alexnaspo* was chosen (https://github.com/alexnaspo/var_dumpling). The reasoning is that, at time of writing, the repository has only 17 stars, giving a small dataset to work with. By taking this approach, it is guaranteed that every user has at least one overlap with another, thereby reducing the cold start problem. With this starting point the dataset can be created, by requesting the "stargazers" of the starting point repository from the API. This returns the list of users that have starred the starting point repository and stores it in the database. The list of users is then looped through to request each individual's starred data, which is stored in the data set.

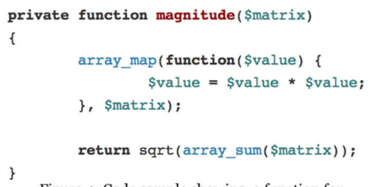
4.2. Similarity Calculation

Once the dataset has been created, the application can give recommendations to repositories within the dataset. As discussed above, cosine similarity calculations will be made to compare repositories. The figures below show the code samples that are used in the application for the calculations. Using these calculations, the repositories are ranked by similarity, and the top 5 returned as a JSON object list.



```
private function dotProduct($aMatrix, $bMatrix)
{
        $arr = [];
        foreach($aMatrix as $key => $value)
        {
                $arr[$key] =  $value * $bMatrix[$key];
        }
        return array_sum($arr);
}
```

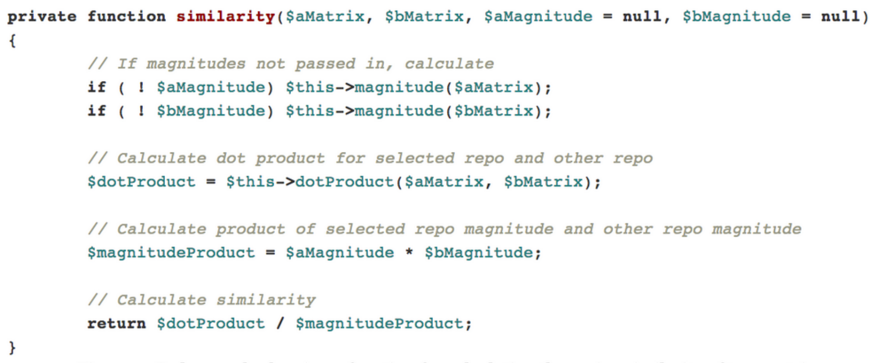
Figure 3: Code sample showing calculation of the dot product of two matrices



```
private function magnitude($matrix)
{
        array_map(function($value) {
                $value = $value * $value;
        }, $matrix);

        return sqrt(array_sum($matrix));
}
```

Figure 4: Code sample showing a function for calculating the magnitude of two matrices



```
private function similarity($aMatrix, $bMatrix, $aMagnitude = null, $bMagnitude = null)
{
        // If magnitudes not passed in, calculate
        if ( ! $aMagnitude) $this->magnitude($aMatrix);
        if ( ! $bMagnitude) $this->magnitude($bMatrix);

        // Calculate dot product for selected repo and other repo
        $dotProduct = $this->dotProduct($aMatrix, $bMatrix);

        // Calculate product of selected repo magnitude and other repo magnitude
        $magnitudeProduct = $aMagnitude * $bMagnitude;

        // Calculate similarity
        return $dotProduct / $magnitudeProduct;
}
```

Figure 5: Code sample showing a function for calculating the cosine similarity of two matrices

5. Results

Results were generated using datasets from both iterations, with data retrieved from the API on 23rd November 2013. Recommendations were generated using Laravel as the *target repository* to find similar repositories for.

5.1. Initial Iteration

Using the initial iteration, that creates the dataset from a whitelist of users, are as follows:

Repository name	**Language**	**Starred by `user_id`**
total-impact-core	Python	1
Total-Impact	PHP	1
subvertle	Python	1
backbone	Javascript	1
todo.txt-android	Java	1

Table 1: Recommendations generated by the first iteration dataset

These results indicate that the dataset is not interrelated enough to generate good results. The recommendations are only relevant to one user, and simply returning a list of other repositories that they previously starred. The recommendations are not relevant to the starting repository (Laravel), with only one PHP repository recommended. Another recommendation is unrelated to the web, but to Android.

5.2. Second Iteration

Using the second iteration, the dataset is created from a base repository that every user has starred. Then using repository that they in turn starred, the dataset is generated. The results are shown in Table 2:

Repository name	**Language**	**Starred by `user_id`**
Laravel-Guard	PHP	1, 20, 17, 18
Ratchet	PHP	1, 12, 13, 14, 15, 17
react	PHP	1, 12, 13, 17
laravel-ide-helper	PHP	12, 13, 15, 17, 18
sentry	PHP	12, 13, 15, 17

Table 2: Recommendations generated by the second iteration dataset

Each of the recommendations are starred by multiple users in the dataset, demonstrating that there is more overlap between users. Additionally, all of the recommended repositories are related to the target repository (Laravel). All are written primarily in PHP.

6. Conclusion

The results indicate that the second iteration of dataset generation is much more suited to providing good recommendations than the first iteration. The first iteration provided recommendations that were only relevant to a single user who was included on the user whitelist. This cold start problem was mitigated in the second iteration by using a list of users who had all starred the same repository.

This second iteration provided much more useful recommendations. Each of the recommendations were relevant to the target repository. Some recommendations were directly related to the target repository, including plugins for the Laravel framework or tools for aiding development with Laravel. Some recommendations where alternative PHP frameworks or libraries.

This is a subjective assessment of the results, as no empirical test was devised to measure the relevancy of the results. Given more time, an empirical test would have been beneficial to have performed, however the authors feel that if the recommendations were useful or interesting, then the algorithm has at least some value.

Overall, the authors feel that a cosine similarity test is a valid and useful method to provide recommendations for repositories on Github. With a sample dataset that more accurately reflects the entire Github dataset, relevant and useful recommendations were generated.

Further studies are required to fully understand whether a cosine similarity test is useful for providing recommendations for repositories on Github. These studies could include enlarging the dataset to test a more complete model of Github's data, or increasing the complexity of similarity measures. For example, no attempt to study the context of programming language was made, and this is an area of potential study.

7. Appendix

Code for the project was released under an MIT License:
https://github.com/40thieves/git-stars

8. References

Balabanović, M., & Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM, 40*(3), 66–72. doi:10.1145/245108.245124

Jannach, D., & Gerhard, F. (2011). Tutorial: Recommender Systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Barcelona. Retrieved from http://ijcai-11.iiia.csic.es/files/proceedings/Tutorial IJCAI 2011 Gesamt.pdf

Notifications & Stars. (2012). *Github blog*. Retrieved November 23, 2013, from https://github.com/blog/1204-notifications-stars

Sarwar, B., Karypis, G., Konstan, J., & Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the tenth international conference on World Wide Web - WWW '01* (pp. 285–295). New York, New York, USA: ACM Press. doi:10.1145/371920.372071

Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '02* (p. 253). New York, New York, USA: ACM Press. doi:10.1145/564376.564421

Sparling, E. I., & Sen, S. (2011). Rating: how difficult is it? In *Proceedings of the fifth ACM conference on Recommender systems - RecSys '11* (p. 149). New York, New York, USA: ACM Press. doi:10.1145/2043932.2043961

Stars · GitHub Help. (2013). Retrieved November 23, 2013, from https://help.github.com/articles/stars

Wauters, R. (2013). Code-sharing site Github turns five and hits 3.5 million users, 6 million repositories. *The Next Web*. Retrieved November 23, 2013, from http://thenextweb.com/insider/2013/04/11/code-sharing-site-github-turns-five-and-hits-3-5-million-users-6-million-repositories/