Recommending Functions in Spreadsheets from the Fuse Corpus

Shaown Sarker, Emerson Murphy-Hill
Department of Computer Science
NC State University
Raleigh, North Carolina
Email: {ssarker, emurph3}@ncsu.edu

***Abstract*—Spreadsheets are the most common form of end-user programming. Although spreadsheets have a large array of functions built-in, spreadsheet users tend to ignore using them to perform their tasks. To address this issue, we investigate recommender system technologies and consider two distinct approaches to a function recommender system for spreadsheets. In this paper, we use collaborative filtering and the most popular algorithm to recommend functions to spreadsheet users. We apply these algorithms on the Fuse spreadsheet corpus to produce personalized function recommendations to an individual spreadsheet user. Our automated evaluation shows that the collaborative filtering based approach outperforms the most popular algorithm by `percentage obtained from future evaluation`. Although recommendation in spreadsheets can be difficult compared to other software applications, the results suggest that we can still have useful function recommendations based on the users' usage history.**

I. Introduction

By definition, end-user programmers are people who are not professional software developers, but make use of tools and processes that lets them perform tasks similar to programming [1]. According to a study from 2005 [2], nearly 23 million Americans use spreadsheets, constituting 30% of the entire workforce. Spreadsheets are also very commonly used in industry, almost 90% of all analysts use spreadsheets to perform their calculations [3]. Based on their number, spreadsheet users form the largest demographic within end-user programmers.

Spreadsheet software come with lots of functions built-in. Consider Microsoft Excel, the most widely used spreadsheet application, which has more than 300 functions[1]. But there are many situations where the user neglects using them. Let us consider the case of a spreadsheet user, Titus. Titus is a fourth grade teacher in an elementary school. At the end of the school year, he wants to calculate the class grades in Excel from all the test scores entered into his spreadsheet. Instead of using the arithmetic functions in Excel, he reaches for his hand-held calculator, and uses it to calculate and enter the grade for each student. This type of usage, not taking advantage of the functions in spreadsheets, is very common with users where they lack awareness of functionality [4] of the end-user programming tool being used.

Functionality awareness is important to accomplish new tasks as well as achieving efficient completion of existing tasks. Systems to recommend commands have been used successfully to improve functionality awareness in large and complex software applications [5], [6]. Recommender systems are used generally to produce a list of predictions based on the preference of an user and the similarity of preferences of other users. In recent years, recommender systems have become quite popular and have been applied successfully in multiple sectors of business [7] and academia [8], [9].

Our contribution in this paper – we recommended functions for spreadsheet users by applying two recommender system algorithms, user-based collaborative filtering [10] and the most popular algorithm [11], on the spreadsheets in the Fuse corpus [12]. We also compare the effectiveness of the recommendations using a cross validation based automated evaluation. According to our results, the collaborative filtering based system suggests effective functions `percentage from future evaluation` better compared to the most popular algorithm based one.

II. Related Work

Researchers have been studying and analyzing spreadsheets to understand activity of the users and design better tools to assist them. Previous research has focused on extracting structured domain information from spreadsheets [13] and visualizing spreadsheet using dataflow diagrams [14] to enhance understandability of spreadsheets. For improving the maintainability and quality of the spreadsheet formulas, systems have been implemented to detect code smells in formulas and refactor them to resolve the detected smells [15]. In contrast, our work attempts to increase the functionality awareness in spreadsheets by suggesting functions.

Moreover, some research has used recommender systems to help users of a large software system with vast set of functionality to learn these functionalities. One of the notable attempts is the OWL system [11], which suggests commands to Microsoft Word users based on the most popular algorithm. In OWL, commands are recommended to an individual if she is not using certain commands at all but the community is using them on a highly frequent basis.

Matejka and colleagues developed a collaborative filtering-based approach to recommend commands in AutoCAD,

[1]https://support.office.com/en-us/article/Excel-functions-by-category-5f91f4e9-7b42-46d2-9bd1-63f26a86c0eb

a computer aided drafting software, called CommunityCommands [5]. Similarly based on the users' usage history, Murphy-Hill and colleagues studied several existing recommendation algorithms to recommend commands in Eclipse. The approach that we propose in this paper, focuses on the algorithm used by OWL and the user-based variation of the collaborative filtering used by CommunityCommands.

Both of the algorithms used in our work requires a source of function usage preference for the spreadsheet users' community. Although there are existent spreadsheet corpora like Enron [16] and EUSES [17] which have been valuable sources for spreadsheet analysts and researchers alike, in this paper we look into a very recently extracted spreadsheet corpus, Fuse [12]. Unlike Enron and EUSES which are very domain specific, Fuse contains a more diverse and much larger body of spreadsheets.

III. Methodology

In this work, we examined and applied two existing recommender system algorithms to produce function recommendations for an individual spreadsheet user. In the following section, we briefly discuss each algorithmic approach.

A. Collaborative Filtering

Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The core concept of collaborative filtering is to apply a nearest neighbor method between a user's preferences and the preference of a large user community, and provide the user with recommendations by extrapolating based on how her selection or preference relates to that of the community.

Although collaborative filtering can be applied in many ways, the approaches falling under the category of user-based collaborative filtering are composed of mainly two distinct steps [18] - find the users with similar preference patterns to that of the input user and then use the preference of the similar users to obtain predictions for the input user.

To apply this technique, we need to measure the similarity between two users. In order to do so, we define a vector for each user's spreadsheets and compare the vectors to find the similar usages. Our function vector is defined such that the vector V for an individual user consists of cells, where each cell V_i represents the usage frequency of the function f_i.

To generate the input user's vector we extract function usage frequencies from multiple files created by the user. The more spreadsheets created by the same user is given as input the better representative vector for the user's function usage we get. As for the potential *pool* of similar user vectors, we used the spreadsheets of the Fuse corpus. The authors of Fuse extracted meta information such as the user name who created the spreadsheet along with function usage frequencies. We classified the spreadsheet meta information and discarded any spreadsheet that did not use any functions. This reduced the number of spreadsheets under consideration to less than 13 thousand. These spreadsheets were then grouped by the user name. We proceeded to extract the function vector from each of these groups in the similar way the input user vector was extracted. The final number of function vectors in this *pool* of potential similar user vectors was `insert actual number from modified code here`.

To measure the similarity between users, we used the cosine similarity function, which measures the cosine of the angle between the user's vectors as described above. When the similarity function evaluates closer to 0, the vectors are substantially orthogonal to each other, which indicates that the users are dissimilar with respect to function usage. A similarity function value of closer to 1, indicates that the vectors are nearly collinear and the users' function usages are quite similar.

To find the similar user vectors for the given vector, we calculated the similarity function value between each of the vector in the *pool* and the input vector, after this the vectors in the *pool* were sorted based on their cosine distance in ascending order. We selected n most similar vectors from this ordered list. Here, n is a tuning parameter, which in our case was between 3 and 5.

We compared the frequencies of each individual function between each of the similar vectors and the input vector, and added a function to the recommendation set only if the function had a frequency value in one of the similar vectors but not in the input vector. The set of recommended functions was then ordered by the aggregate frequency of the functions within the similar vectors, this way the functions in the recommendation list appeared in order of their usage in the similar vectors. Since the list of possible recommendations were quite large, we limited the number of functions suggested to a maximum of 10 for the sake of brevity and relevance.

B. Most Popular

Linton and colleagues introduced this algorithm in their OWL [11] system which recommended commands in Microsoft Word. The most popular algorithm suggests functions that are most widely used by the community. To implement this algorithm, we took the cumulative frequency count for each function over all the spreadsheets in Fuse, which is referred to as the *confidence level* of a function. When we are given an input user vector as described in the previous section, this algorithm recommends functions with highest confidence level values, excluding any function that is present in the user's input vector.

The underlying assumption of this algorithm is that the functions most frequently used by the community are also the most useful functions. For this reason, this algorithm has an advantage over collaborative filtering - it can recommend functions to users whose function usage history is not known (the input vector consists of entirely zeros). However, this algorithm suffers from lack of personalization, as it does not take into consideration the usage frequencies of the functions in the input user vector.

IV. Evaluation

For evaluating the system implemented based on the algorithm in the previous section, we used a cross validation technique, which enabled us to evaluate our system using the existing feature vectors extracted from the Fuse corpus. Cross validation is a model evaluation technique, where the input data is partitioned into two complementary subsets – one subset is used to perform the analysis (training set) and the other is used to validate the analysis on the training set (testing set). Generally, multiple passes of cross validation are performed using different partitions to cope with variance in the dataset and the results are averaged.

We used a variation of cross validation, called Leave p-out (LPO) cross validation [19]. According to LPO, all possible training/testing partitions are generated by removing p samples from the complete set. The testing set consists of the p samples and the training set contains the rest $n - p$ samples, where n is the size of the original dataset. Thus, a dataset of length n will have $\binom{n}{p}$ possible partitions in LPO cross validation.

We used the function vectors extracted from Fuse to perform LPO cross validation. In LPO cross validation, such a function vector V_i will be partitioned into the training set $S_{training}$ and the testing set $S_{testing}$, where the size of $S_{testing}$ is p. The $S_{training}$ set with the rest of the function frequencies set to 0, is used as an input vector for the collaborative filtering and the most popular algorithm based recommender systems. Each system produces the recommendation set R_S. We define the correct number of recommendations for this partition as the number of functions that are both in set R_S and $S_{testing}$ and calculate the result of the LPO evaluation for this function vector h_i as:

$$h_i = \frac{\sum_{k=1}^{n} |R_{S_k} \cap S_{testing_k}|}{n}$$

Where k presents each distinct partition and n is $\binom{|V_i|}{p}$. We calculated h_i for every vector V_i in the *pool* with a p value of 3. And finally, the h_i values are averaged over the entire vector *pool* from Fuse to get the evaluation result.

V. Results

1) `Discuss the results of the two approaches in detail.`
2) `Graphic comparing the two algorithms. Horizontal bar charts?`

VI. Discussion

1) `Consider naming it to contribution and/or merging it with Results section.`
2) `Any future tool related discussions here, if any.`

VII. Limitations & Future Work

From the results presented here, there are a number of attributes of the approaches applied here, that provides opportunities for future work.

One of the limitation of the recommendation systems used here was that the *pool* of potential similar function vectors was small, given that only approximately 5% of the Fuse spreadsheets used formulas. This however can be alleviated considerably by incorporating other existing spreadsheet corpus like Enron [16] and EUSES [17] into the user function vectors *pool*.

We also made an explicit assumption for the user-based collaborative filtering algorithm that the input user's spreadsheets will result in a function vector with some non-zero frequencies. If the input user's spreadsheets are simple data dumps without any function usage, the collaborative filtering based recommender system cannot produce useful recommendations as the similar user vectors in this case will be the ones with lowest function frequency values. However, the most popular algorithm can still recommend functions in this case.

Spreadsheet files, being static, do not contain any temporal or sequential information regarding function discovery. Given the function discovery information, it will be possible to apply the discovery variations of the algorithms as described by Murphy-Hill and colleagues, which produced the recommendations with the highest useful rates in their automated evaluation [6]. In the discovery variations of the algorithms, instead of using function frequency based vectors, users' vectors consist of discovery patterns that reflect the order in which an user discovers functions in spreadsheets. The collaborative filtering version with discovery finds user vectors with similar discovery patterns and recommends functions the similar users have discovered but the user has not. In the most popular algorithm with discovery, the most discovered function in the community is recommended to an user who has not discovered that function yet.

Although we used an automated method to evaluate our system, it cannot replace the recommendations being evaluated by real life spreadsheet users. This type of evaluation can prove how much useful the recommendations actually are. It is possible to conduct a study, either web-based or real life, where we generate recommendations based on spreadsheet users in the industry and have the actual owners of the spreadsheets evaluate their personalized function recommendations.

Acknowledgment

Thanks to the Laboratory of Analytical Sciences for funding this research.

References

[1] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers *et al.*, "The state of the art in end-user software engineering," *ACM Computing Surveys (CSUR)*, vol. 43, no. 3, p. 21, 2011.

[2] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," in *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*. IEEE, 2005, pp. 207–214.

[3] W. Winston, "Executive education opportunities millions of analysts need training in spreadsheet modeling, optimization, monte carlo simulation and data analysis," *OR MS TODAY*, vol. 28, no. 4, pp. 36–39, 2001.

[4] T. Grossman, G. Fitzmaurice, and R. Attar, "A survey of software learnability: metrics, methodologies and guidelines," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 649–658.

[5] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice, "Community-commands: command recommendations for software applications," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. ACM, 2009, pp. 193–202.

[6] E. Murphy-Hill, R. Jiresal, and G. C. Murphy, "Improving software developers' fluency by recommending development environment commands," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 42.

[7] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003.

[8] M.-H. Hsu, "A personalized english learning recommender system for esl students," *Expert Systems with Applications*, vol. 34, no. 1, pp. 683–688, 2008.

[9] S. M. McNee, N. Kapoor, and J. A. Konstan, "Don't look stupid: avoiding pitfalls when recommending research papers," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. ACM, 2006, pp. 171–180.

[10] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, "Recommending and evaluating choices in a virtual community of use," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co., 1995, pp. 194–201.

[11] F. Linton, D. Joy, H.-P. Schaefer, and A. Charron, "Owl: A recommender system for organization-wide learning," *Educational Technology & Society*, vol. 3, no. 1, pp. 62–76, 2000.

[12] T. Barik, K. Lubick, J. Smith, J. Slankas, and E. Murphy-Hill, "Fuse: A reproducible, extendable, internet-scale corpus of spreadsheets," 2015.

[13] F. Hermans, M. Pinzger, and A. van Deursen, "Automatically extracting class diagrams from spreadsheets," in *ECOOP 2010–Object-Oriented Programming*. Springer Berlin Heidelberg, 2010, pp. 52–75.

[14] ——, "Breviz: Visualizing spreadsheets using dataflow diagrams," *arXiv preprint arXiv:1111.6895*, 2011.

[15] ——, "Detecting and refactoring code smells in spreadsheet formulas," *Empirical Software Engineering*, vol. 20, no. 2, pp. 549–575, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9296-2

[16] F. Hermans and E. Murphy-Hill, "Enrons spreadsheets and related emails: A dataset and analysis," Delft University of Technology, Software Engineering Research Group, Tech. Rep., 2014.

[17] M. Fisher and G. Rothermel, "The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–5.

[18] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.

[19] S. Arlot, A. Celisse *et al.*, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.