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 software. 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 makes 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 commonly used for analytical purposes 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 softwares come with lots of functions built-in. Considering that Microsoft Excel, the most widely used spreadsheet application, has more than 300 functions[1], but there are many situations where the generic user neglects using them. Let us consider the case of a typical 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 trend of not taking advantage of the functions in spreadsheet is very common with users where they lack awareness of functionality [4] of the end-user programming tool being used.

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

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].

In this paper, we recommend 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

Collaborative filtering is one of the most prominent recommending technologies available. 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 relate to that of the community. Collaborative filtering has been applied to recommend a fringe of products and services, ranging from movies [13] to school courses [8], [14].

However, there has been little effort to use recommendation systems to help users of a large software system with vast set of functionality to learn these functionalities. One of the notable attempts to use recommender systems to recommend command is the OWL [11], which makes command recommendations to Microsoft Word users. OWL recommends commands to an individual if she is not using certain commands at all but the community is using them on a frequent basis.

Matejka and colleagues developed a collaborative filtering-based approach to recommend commands in AutoCAD, a computer aided drafting software, called CommunityCommands [5]. The researchers of CommunityCommands

showed that the collaborative filtering derived command recommendations were preferred by the AutoCAD users over the history based recommendation approach used in OWL.

In this paper, we build on these prior works by applying the concept of recommender systems to recommend functions in spreadsheets. We use the Fuse spreadsheet corpus as a source of function usage preference of the spreadsheet users' community and recommend functions contextualized to an individual user by applying collaborative filtering algorithm.

III. Methodology

According to the formal definition, 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) [2]. Although collaborative filtering can be applied in many ways, most approaches falling under the category of user-based collaborative filtering are composed of mainly two distinct steps - 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 preferences. 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 we get. As for the potential pool of similar user vectors, we decided to use Fuse, a spreadsheet corpus containing nearly 250 thousand unique spreadsheets extracted from the Common Crawl[3] index [12]. The authors of Fuse also extracted meta information such as created by and function usage frequencies. We classified the spreadsheet meta information and discarded any spreadsheet that did not use any function. This reduced the number of spreadsheets under consideration to less than 13 thousand. These spreadsheets were then grouped by the user name `(There are a large number of spreadsheets where this information is not present, how do we refer to them in the paper?)`. 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. Given the function vector for two users a and b as V_a and V_b respectively, the similarity function s is given by:

$$s(a,b) = \cos({}_{V_a}\theta_{V_b}) = \frac{V_a \cdot V_b}{||V_a||||V_b||} = \frac{\sum_{k=1}^{n} V_{a_k} V_{b_k}}{\sqrt{\sum_{k=1}^{n} {V_{a_k}}^2 {V_{b_k}}^2}}$$

When the similarity function evaluates closer to 0, the vectors are substantially orthogonal to each other, which indicates that the users are dissimilar. A similarity function value of closer to 1, indicates that the vectors are nearly collinear and the users are quite similar. For ease of our implementation we used the complement of the cosine similarity: $1 - similarity(i,j)$, which is often commonly referred to as *cosine distance*.

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, where 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 cumulative frequency of the functions in the similar vectors, this way the functions in the recommendation list appear in order of their usage in the similar vectors.

IV. Evaluation

For evaluating the system implemented based on the algorithm in the previous section, we used a cross validation [15] 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 [16]. 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-based

[2]https://en.wikipedia.org/wiki/Collaborative_filtering

[3]http://www.commoncrawl.org

recommender system, which 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 function vector pool with a p value of 3.

V. Results

To compare the performance of the collaborative filtering-based recommender system, we also implemented the most popular algorithm [11]. The most popular algorithm recommends functions that are most widely used in the Fuse corpus, but not used by the active user. We used the LPO cross validation to evaluate the recommendations generated by this algorithm and compared the results. Our results show that the collaborative filtering-based system correctly recommends functions for `insert future calculated number here` users compared to `insert future calculated number here` correct recommendations from the most popular algorithm.

`Graphic comparing the two algorithms. Bar charts?`

VI. Discussion

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

VII. Limitations & Future Work

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

One of the limitation of the recommendation system 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 [17] and EUSES [18] into the user function vectors pool.

We also made an explicit assumption that the active user's spreadsheets will result in a function vector with some non-zero frequencies. If the active user's spreadsheets are simple data dumps without any function usage, the recommender system can't produce viable recommendations as the similar user vectors in this case will be the ones with lowest function frequency values.

Spreadsheet files, being static, don't contain any temporal or sequential information regarding function discovery. Given this information, it will be possible to recommend more contextualized and personalized functions for the spreadsheet users.

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 survey, 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

`Acknowledgements if any (LAS information maybe?)`

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, "Communitycommands: 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] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "Movielens unplugged: experiences with an occasionally connected recommender system," in *Proceedings of the 8th international conference on Intelligent user interfaces*. ACM, 2003, pp. 263–266.

[14] R. Farzan and P. Brusilovsky, "Social navigation support in a course recommendation system," in *Adaptive hypermedia and adaptive web-based systems*. Springer, 2006, pp. 91–100.

[15] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection." Morgan Kaufmann, 1995, pp. 1137–1143.

[16] S. Arlot, A. Celisse *et al.*, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.
[17] 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.
[18] 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.