Recommending Functions in Spreadsheets from the Fuse Corpus

Shaown Sarker, Matthew Neal, Nisarg Vinchhi
Department of Computer Science
North Carolina State University
Raleigh, NC 27606
{ssarker,meneal,nvinchh}@ncsu.edu

Abstract

Spreadsheets are the most widely used form of end-user programming. Although spreadsheets have a large array of functions built-in, the majority of spreadsheet users often do not utilize them in performing their tasks. To address this lack of function usage, we investigate recommender system technologies and consider two distinct approaches to a function recommender system for spreadsheets. In our work, we apply two variations of the collaborative filtering algorithm to produce personalized function recommendations for spreadsheet users by applying these algorithms on the Fuse spreadsheet corpus. We compare the performance of the algorithms used with a baseline of the Most Popular algorithm, also known as Linton's algorithm. In this paper, we present the feature extraction process from the spreadsheet corpus, the algorithms used and their comparative performance. Our cross validation results show that with a limited number of recommendations, our item-based recommendation algorithm outperforms both the baseline and the user-based approach.

1 Introduction

Ko et al. define end-user programmers as people who are not professional software developers, but make use of the tools and processes that let 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 used not only for home and small businesses, but they are also very commonly used in industry. Almost 90% of all analysts use spreadsheets to perform their calculations [3]. Based on their numbers, spreadsheet users form the largest demographic within end-user programmers.

Spreadsheet software comes 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 that lack awareness of the 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

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

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 recommend functions in spreadsheets by applying two distinct variations of the collaborative filtering algorithm and compare the effectiveness of the recommendations using a cross validation based automated evaluation. We also compare performance for recommending functions against another algorithm previously used for recommending commands in large software systems: the Most Popular algorithm [10]. We conclude by discussing how the results of our work can be improved by conflating our user vectors set with other existing spreadsheet corpora and a real-world user evaluation of our recommendations.

2 Related Work

Researchers have been studying and analyzing spreadsheets to better comprehend the activity of users and design better tools to assist them. Previous research has focused on extracting structured domain information from spreadsheets [11] and visualizing spreadsheets using dataflow diagrams [12] to enhance understandability. For improving the maintainability and quality of spreadsheet formulas, systems have been implemented to detect code smells in formulas and refactor them to resolve the detected smells [13]. 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 early notable attempts is the OWL system [10] developed by Linton et al., 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 frequent basis. The underlying assumption of this system is that all users of a software system have a similar command usage distribution.

Matejka and his colleagues developed a collaborative filtering-based approach called CommunityCommands [5] to recommend commands in AutoCAD, a computer aided drafting software. Similarly based on 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 as a baseline and the user-based and item-based variations of the collaborative filtering algorithm used by CommunityCommands.

Both of the algorithms used in our work require a source of function usage preferences for the spreadsheet users' community. Although there are existent spreadsheet corpora like Enron [14] and EUSES [15] which have been valuable sources for spreadsheet analysts and researchers alike, in this paper we look into a very recently extracted spreadsheet corpus: Fuse [16]. Unlike Enron and EUSES which are very domain specific, Fuse contains a more diverse and much larger body of spreadsheets. Fuse is also reproducible, thus the recommender system in our work can leverage of a larger spreadsheet corpus if needed, which can be extracted using the system developed by Fuse's creators.

The rest of the paper is organized as follows: we describe the feature extraction process by which we retrieved the user preferences for the recommender systems, along with a brief explanation of how collaborative filtering and the baseline algorithms work. We then describe our evaluation strategy followed by the results observed. In the end, we discuss any threat to validity of our work and the future work that can be extended based on our contribution.

3 Methodology

3.1 Feature Extraction

The Fuse spreadsheet corpus consists of nearly two hundred fifty thousand distinct spreadsheets. The creators of Fuse extracted the meta information for these spreadsheets in JSON format using Apache POI library which includes the user that created and last modified the spreadsheet, and the frequency of the functions being used in the spreadsheet. We identified almost seven thousand individual users by combining the created by and last modified by meta information along with the domain name where the spreadsheet was originally extracted. This gave us a collection of nearly seven thousand user vectors, such that the vector V for an individual user consists of cells, in which each cell V_i represents the usage frequency of the function f_i. Consider the sample user identifying key, user vector tuple below:

```
('Greenfield, Laura#Greenfield, Laura#www.cde.state.co.us',
        {'SUM': 8, 'Plus':  179, 'Divide':  179})
```

Here, this user has used 'SUM', 'PLUS', and 'DIVIDE' functions 8, 179, and 179 times respectively over her spreadsheets in Fuse.

3.2 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. In our work, we look at the user-based and item-based variations of collaborative filtering.

3.2.1 User-based Collaborative Filtering

Although collaborative filtering can be applied in many ways, the approaches falling under the category of user-based collaborative filtering are composed of two distinct steps [17] - find the users with similar preference patterns to that of the input user, and then use the preferences of similar users to obtain predictions for the input user.

We use the user vectors extracted in the feature extraction step here. To compensate for the overriding influence of functions that are frequently used by a large number of users, we use a weighting function named 'function frequency inverse user frequency' (ff-iuf) which is similar to the well-known 'term frequency inverse document frequency' (tf-idf) technique used to determine the importance of a word or term to a document in a collection [18]. The ff-iuf for an user i and a function j can be defined as:

$$ff\text{-}iuf_{ij} = \alpha \cdot ff_{ij} \cdot iuf_{ij}$$

Where, α is a tuning parameter, and

$$ff_{ij} = \frac{\text{Frequency of function j for user i}}{\text{Total frequency of all functions used by user i}}$$

$$iuf_{ij} = \log \frac{\text{Total number of users}}{\text{Number of users that use function j}}$$

In the weighted vectors, a higher value signifies that function is used frequently by that individual user, but that function is used by a relatively small portion of the total users.

To obtain the users with similar function usage to the input user, we used cosine similarity function that measures the angle between the users' vectors. When the cosine similarity is near zero, the angle between the vectors are closer to a right angle and the vectors are not very similar. On the other hand, if this value is close to 1, then the vectors are nearly colinear and are quite similar. By comparing the cosine similarity value between the input user and the other vectors, we determine the n most similar users, where n is another tuning paramter in our system.

After we have the similar users, we recommend the top m functions that are used by the similar users but not by the input user, such that m is another tuning parameter. The recommended functions are ordered by their cumulative function frequency over the n similar users.

3.2.2 Item-based Collaborative Filtering

In this approach, using the ff-iuf weighted vectors, we define a vector for each function j, in which a cell i in the vector is the ff-iuf value for user i and function j.

Then we generate a function-to-function two dimensional similarity matrix M, where the matrix element m_{jk} represents the cosine similarity value between function vectors for function j and k.

Given the input user, we recommend m functions that are not used by the user, ordered by their similarity score from the similarity matrix, where for all function i and function k, such that i is not used but k is used by input user:

$$\text{similarity score for function i}, s_i = average(m_{ik})$$

3.2.3 Most Popular

Linton and colleagues introduced the Most Popular algorithm in their OWL [10] system which recommended commands in Microsoft Word. The Most Popular algorithm suggests functions that are most widely used by the community but not used by the input user. 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 the 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.

4 Evaluation

For evaluating the system based on the algorithms in the previous section, we used the k-folds cross validation technique, which enabled us to evaluate our system using the existing feature vectors extracted from the Fuse corpus. We used a 14 fold cross validation, splitting the entire set of almost 7,000 user vectors in 14 slices and treating a single slice as testing set and the rest as our training set.

In our cross validation, we took a single user vector from the testing set as input vector. In this input vector, we removed a function at random and retrieved the recommendations for this modified input vector. If the set of recommendations contained the removed function, then we counted this as a successful recommendation. The success rate was represented as a percentage of each individual testing set. The overall 14-fold cross validation success rate was calculated as a cumulative percentage of these individual 14 success rates.

5 Results

In all our recommendation systems, we had one common tuning parameter - the number of functions recommended, m. Although a smaller value for m is preferred for usability purposes, we ran our cross validation using four different values for m: 1, 3, 5, and 10. For the user-based collaborative filtering algorithm, we had another tuning parameter - the number of similar user vectors used to generate the recommendations. We chose two values for this tuning parameter: 10 and 20, as values below 10 would often not fill the required $m = 10$ function recommendations. With these tuning parameters, we had the following four distinct cross validation run categories, each with four different runs for each distinct m value:

- Most popular
- Item-based collaborative filtering

- User-based collaborative filtering with 10 similar user vectors.
- User-based collaborating filtering with 20 similar user vectors.

Our cross validation success rate represents the overall cumulative success rate over the entire 14-slice user vector training sets. Figure 1 shows the percentage success rates over the number of recommendations as a line graph, and table 1 shows the exact success rate percentage values over the number of recommendations.

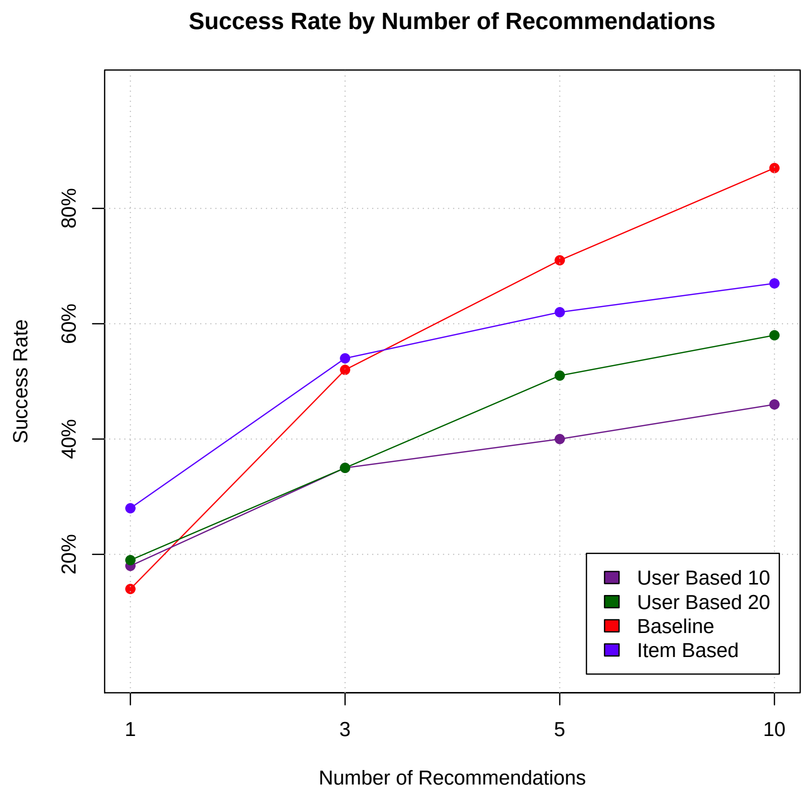

Figure 1: Number of recommendations vs success rate

NumRecs	Item	User 10	User 20	Baseline
1	27.66%	17.76%	18.50%	14.10%
3	54.29%	34.50%	35.10%	52.22%
5	62.45%	39.94%	51.00%	70.64%
10	66.70%	46.28%	57.80%	87.30%

Table 1: Number of recommendations and success rate

From the results, we can see that for 1 and 3 recommendations, item-based algorithm performs considerably better compared to the rest. The performance of the two user-based approaches are identical for $m = 1, 3$, but they outperform the baseline only for $m = 1$. For $m = 10, 20$, the baseline algorithm clearly outperforms the collaborative filtering based approaches, with the item-based algorithm stil performing better than both user-based approaches.

6 Discussion

Our results show that the item-based collaborative filtering algorithm performs best for smaller numbers of recommended functions, but as the number of recommendations, m, increases; the increase in performance is small compared to the baseline - most popular algorithm. The high success rate of the most popular algorithm for larger values of m can be explained by the lack of function usage diversity within the Fuse corpus, thus when recommending the top 10 or 20 functions from the most popular list, it is quite likely that a large number of users will only be using these functions. This reinstates our original hypothesis that most spreadsheet users only use a limited number of functions.

In our results, the user-based algorithm approach performed considerably worse compared to both baseline and the item-based algorithms across the board. The better performance of item-based over user-based can be explained based on the way these algorithms retrieve the recommendations. In the item-based approach we look up the similarity score in the matrix, which is compiled over the entire training set, thus providing a refined similarity score value and a more highly correlated set of recommendations. Whereas in the user-based approach, we only look at the functions in the similar users, thus limiting the possible quality of the recommendations to the set of functions in the similar users.

7 Future Work & Conclusion

One of the limitations of the recommendation systems used in this work was that the function usage rate and diversity was quite low in the Fuse spreadsheet corpus, where only a very small percentage of the spreadsheets contained function usage. This however can be alleviated considerably by incorporating other existing spreadsheet corpus like Enron [14] and EUSES [15] into the extracted user function vectors used in our algorithms.

Spreadsheet files, being static, do not contain any temporal or sequential information regarding function discovery. Given function discovery sequence information, it would be possible to apply a number of other recommendation systems that could result in a much higher success rate. For example, the discovery variations of the algorithms as described by Murphy-Hill and colleagues, which produced recommendations with the highest success rates in their automated evaluation [6]. In the discovery variations of the algorithms, instead of using function frequency based vectors, user 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 useful the recommendations actually are. It would be possible to conduct a study, in which we generate recommendations based on spreadsheet users in industry and have the actual owners of the spreadsheets evaluate their personalized function recommendations. Alternatively, we could also measure the effectiveness of our recommendations by observing if users accept the recommendations by actually using the function being recommended.

In this paper, we presented two distinct collaborative filtering approaches to recommend functions in spreadsheets to increase functional awareness. Our results also show that one of the algorithms - the item-based approach, performs best with a smaller number of functions recommended. These results can effectively be used to create personalized function recommendations for spreadsheet users. Given the lack of diversity in function usage among spreadsheet users and the large demographic of end-user programmers, we believe there is a large population that can benefit from effective, personalized, and contextual function recommendations.

References

[1] Andrew J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, et al. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*, 43(3):21, 2011.

[2] Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the numbers of end users and end user programmers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 207–214. IEEE, 2005.

[3] WL Winston. Executive education opportunities millions of analysts need training in spreadsheet modeling, optimization, monte carlo simulation and data analysis. *OR MS TODAY*, 28(4):36–39, 2001.

[4] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A survey of software learnability: metrics, methodologies and guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 649–658. ACM, 2009.

[5] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. Communitycommands: command recommendations for software applications. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 193–202. ACM, 2009.

[6] Emerson Murphy-Hill, Rahul Jiresal, and Gail 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*, page 42. ACM, 2012.

[7] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[8] Mei-Hua Hsu. A personalized english learning recommender system for esl students. *Expert Systems with Applications*, 34(1):683–688, 2008.

[9] Sean M McNee, Nishikant Kapoor, and Joseph 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*, pages 171–180. ACM, 2006.

[10] Frank Linton, Deborah Joy, Hans-Peter Schaefer, and Andrew Charron. Owl: A recommender system for organization-wide learning. *Educational Technology & Society*, 3(1):62–76, 2000.

[11] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Automatically extracting class diagrams from spreadsheets. In *ECOOP 2010–Object-Oriented Programming*, pages 52–75. Springer Berlin Heidelberg, 2010.

[12] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Breviz: Visualizing spreadsheets using dataflow diagrams. *arXiv preprint arXiv:1111.6895*, 2011.

[13] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Detecting and refactoring code smells in spreadsheet formulas. *Empirical Software Engineering*, 20(2):549–575, 2015.

[14] Felienne Hermans and Emerson Murphy-Hill. Enrons spreadsheets and related emails: A dataset and analysis. Technical report, Delft University of Technology, Software Engineering Research Group, 2014.

[15] Marc Fisher and Gregg Rothermel. The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[16] Titus Barik, Kevin Lubick, Justin Smith, John Slankas, and Emerson Murphy-Hill. Fuse: A reproducible, extendable, internet-scale corpus of spreadsheets. 2015.

[17] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[18] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.