# 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 to perform 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 collaborative filtering algorithm to produce personalized function recommendations to spreadsheet users by applying these algorithms on the Fuse spreadsheet corpus. We compare the performance of the algorithms used with a baseline of 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. Finally, we also outline the roadmap to use our findings to create an Excel plugin for increasing functional awareness.

## 1 Introduction

Ko et al. define end-user programmers as 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 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 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

[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 recommended functions in spreadsheets by applying two distinct variations of collaborative filtering algorithm and compare the effectiveness of the recommendations using a cross validation based automated evaluation. We also compared their performance for recommending functions against another algorithm priorly used for recommending commands in large software systems, the most popular algorithm [10]. And we conclude by discussing how the results of our work can be used to create an Microsoft Excel plugin that can assist users to increase functional awareness by recommending functions they can use.

## 2 Related Work

Researchers have been studying and analyzing spreadsheets to better comprehend the activity of the users and design better tools to assist them. Previous research has focused on extracting structured domain information from spreadsheets [11] and visualizing spreadsheet using dataflow diagrams [12] 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 [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 highly frequent basis. The underlying assumption of this system is that all users of a software system have a similar command usage distribution. Recommender systems were developed to improve upon the system delineated by Linton.

Matejka and colleagues developed a collaborative filtering-based approach to recommend commands in AutoCAD, 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 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 requires a source of function usage preference 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 by 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 to be used for the recommender systems, along with a brief explanation of how the collabor 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 250 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 functions being used in the spreadsheet. We identified almost 7,000 individual users by combining the created by and last modified by meta information with the domain name where the spreadsheet was originally extracted. This gave us a collection of nearly 7,000 user vectors such that the vector $V$ for an individual user consists of cells, where each cell $V_i$ represents the usage frequency of the function $f_i$. 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.

#### 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 mainly two distinct steps [17] - 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.

### 3.3 Most Popular

Linton and colleagues introduced this 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. 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.

## 4 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 [18]. 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.

## 5 Results

## 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] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.