

Assessment of Conversation Co-mentions as a Resource for Software Module Recommendation

Daniel Xiaodan Zhou
University of Michigan
School of Information
1075 Beal Ave, Ann Arbor, MI 48109
mrzhou@umich.edu

Paul Resnick
University of Michigan
School of Information
1075 Beal Ave, Ann Arbor, MI 48109
presnick@umich.edu

ABSTRACT

Conversation double pivots recommend target items related to a source item, based on co-mentions of source and target items in online forums. We deployed several variants on the drupal.org site that supports the Drupal open source community, and assessed them through clickthrough rates. A similarity metric based on correlation of mentions rather than mere co-occurrence reduced the problem of over-recommending the most popular modules, but additional corrections for recency and uniqueness of mentions were not helpful. Detection of more module mentions in conversations dramatically improved the quality of recommendations, even though the detection algorithm then had more false positives. Recommendations based on conversation co-mention were more effective than those based on co-installation, because co-installation data only led to recommendations of complementary modules and not substitutes. Recommendations based on co-mention were more effective than those based on text similarity matching for navigating from the most popular modules, but less effective than text matching for less popular modules.

Categories and Subject Descriptors

H.3.3 [Information Systems]: Information Search and Retrieval – *information filtering, selection process.*

General Terms

Algorithms, Design

Keywords

Conversation double pivot, evaluation, item-item recommender, Drupal

1. INTRODUCTION

People rely on recommender systems when they need to allocate their limited attention to only a few things that might interest them the most. Successful recommender systems have been reported in several application domains, such as movies [1], Usenet articles [9], and consumer products [7].

Typically, recommender systems produce personalized predicted ratings or top-N lists [2]. Another use case is to recommend non-

personalized sets of related items on each item page, as part of a search or browsing process among pages for items of potential interest. For example, Amazon suggests, on the page for an item, various other items that might be of interest. The suggested items may be complementary (“buying a razor? Perhaps you’d like blades”) or they may be substitutes (“perhaps you’d like this other brand of razors instead”). The recommendations on any particular page are the same for all visitors—they are personalized only to the extent that different users will visit different pages.

Presentation of related-items while browsing is well-suited to the domain of software module recommendations. Many software platforms allow third party contributed add-on extensions, or modules. For example, Perl, Python, and other programming languages have add-on software libraries or modules available. Browsers like Firefox offer add-ons, portals like iGoogle offer plug-in “gadgets”, and FaceBook offers “apps” that users can install. A user’s interest in a module is sufficient to indicate a potential interest in alternative, substitute modules (perhaps of higher quality) and complementary modules that work with the one currently being considered. While it is possible to imagine systematic taste differences (one user prefers text interfaces; another GUIs), personalization based just on which module page the user is currently visiting is likely to be quite effective.

Because modules are typically accompanied by short descriptive text, content-based filters can compare the descriptions of modules to assess similarity, but the descriptions provide little clue as to quality. Collaborative data, indicators of other people’s assessments of items, can be used to assess both similarity and quality. Those indicators can be explicit, based on, for example, five-star or up/down ratings. They may also be implicit, based on purchases, link citations [3], link clickthrough [6], or time spent reading [8]. In the case of software modules, we have several potential additional implicit indicators: user downloads, installations on running systems, and mentions of modules in public discussion forums.

Terveen et al mined Usenet conversations for mentions of URLs [13]. Drenner et al explored the automatic detection of mentions of movies in forum discussions, and the suggestion of related conversations on movie pages [4]. In our previous work [14], we applied the idea to software modules, introduced the idea of a conversation double pivot from pages to conversations and back to other modules mentioned in the same discussions, and showed that it could be used to generate at least some recommendations for most Drupal and Perl modules. The double pivot technique is closely related to the technique of item-item collaborative filtering [11], where two items are related if they are highly rated (or purchased) by the same people. It is also analogous to co-citation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

RecSys '09, October 23–25, 2009, New York, New York, USA.
Copyright 2009 ACM 978-1-60558-435-5/09/10...\$10.00.

analysis in bibliometrics, where two papers are related if they are cited in the same other paper [12].

This paper reports on a tuning and assessment process for conversation double pivots. We deployed them on drupal.org, the official portal for the Drupal open source community. Drupal is a popular Web Content Management software platform that has more than 4,000 contributed modules. Drupal.org receives about 600,000 pageviews from 100,000 visitors on a typical day; it has an average of 200 new incoming conversation threads daily. Figure 1 shows pivots enabled on a typical drupal.org module page. Area B of Figure 1 shows the related module recommendations to the module on the current page.

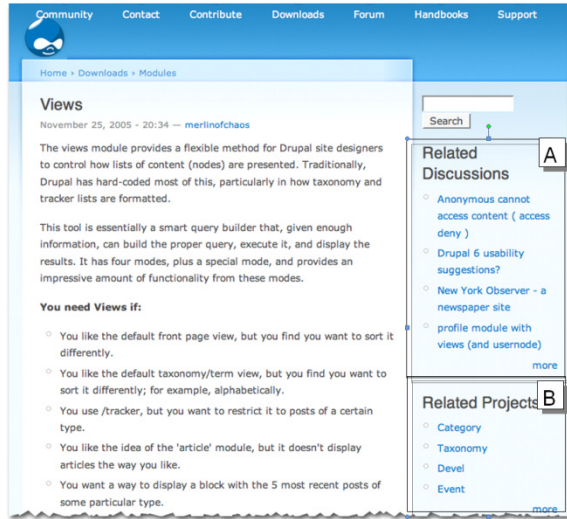


Figure 1. Drupal.org module page with (A) pivot to related conversations and (B) double pivot to other modules

We conducted a four-stage assessment and tuning process. First, holding fixed the algorithm for detecting mentions of a module in a conversation, we conducted a bake-off comparing four alternative algorithms for using the detected module co-mentions to produce recommendations, correcting for popularity, recency, and uniqueness. Next, we assessed the utility of the conversation pivot and double pivot blocks by comparing the clickthrough rate for the winning version of the recommender from stage 1 to an alternative block of similar size that was already deployed elsewhere on the drupal.org site. Third, we adjusted the algorithm for detecting mentions of a module by adding a number of “aliases” that people often use as shorthand to refer to modules, which nearly tripled the number of mentions detected, while introducing only a modest number of false positives. In Stage 3 we assessed the change in performance resulting from the changed detection algorithm. Finally, we compared the performance of the tuned recommender based on conversation double pivots to four other recommender algorithms: one based on co-popularity of installation of modules and three variations of a content similarity algorithm that compared the texts describing the modules. Figure 2 summarizes the stages of tuning and assessment.

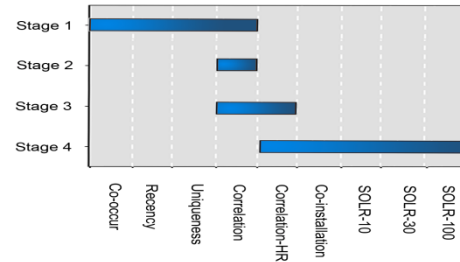


Figure 2. Summary of tuning and assessment process

2. CLICKTHROUGH ASSESSMENT

To assess conversation double pivots as a recommender system for software modules, we deployed the alternative algorithms on the live drupal.org site and measured the actual clickthrough rates. Higher clickthrough rates were considered indicators of better recommendations. Usage was tracked with Google Analytics, which naturally filters out visits from most automated crawlers and keeps track of user sessions. Unique views for a particular module page were counted, as the number of distinct sessions in which the page for that module was displayed. For each module, each session in which the user clicked on at least one of the recommended links to other modules in the “Related Projects” block counted as a recommended module clickthrough; similarly, one or more clicks in the related discussions block counted as a single conversation clickthrough for that module.

We also attempted to subjectively code the recommended modules as either non-relevant, relevant substitute modules, or as relevant complementary modules. We found, however, that complement vs. substitute judgments require significant domain knowledge about a large number of Drupal modules, and that even the relevance/non-relevance criteria depend considerably on the task scenario that one imagines for a user of the recommendations. We were unable to achieve sufficient inter-coder reliability and do not report the relevance assessments here.

3. ALGORITHMS AND TESTS

3.1 Alternative Double Pivot Algorithms

In the first stage of tuning, we implemented and tested four variations of an item-item recommender system [4] that all use the same conversation co-mention data but process it slightly differently. Each algorithm computes a square item-item similarity matrix whose rows and columns are the sets of software modules. Items with the highest similarity scores are recommended.

The conversation mention data that all algorithms work from is a matrix M with one row for each software module and one column for each conversation thread. We treat the thread as the unit of analysis rather than the individual message because we thought that when two modules are mentioned anywhere in a thread, even if not in the same message, it indicates a relationship that would be useful in recommending modules. $M_{it} = 1$ if thread t mentions item i ; otherwise $M_{it} = 0$. Table 1 is a fictitious small table of items and threads that will be used to illustrate the algorithms. The actual matrix contains 1,645 rows for the software modules and more than 47,000 columns for threads in the forums on the drupal.org site as of November 16, 2008.

Table 1. A matrix M of module mentions

		Conversation Threads			
		T1	T2	T3	T4
Software Module	S1	0	1	0	1
	S2	1	0	1	0
	S3	1	0	1	1

3.1.1 The Co-occur algorithm

The first version of the similarity function, as reported previously [14], simply counts co-occurrences. That is:

$$Co - occur_{ij} = \sum_t M_{it}M_{jt}$$

Table 2 shows the matrix Co-occur that would be generated from the detected item mentions from Table 1. Module S1 is not related to S2 because they are never referenced in the same thread, but is somewhat related to S3 because one thread mentions both S1 and S3.

Table 2. Co-occur algorithm similarity scores

		Software Module		
		S1	S2	S3
Software Module	S1	2	0	1
	S2	0	2	2
	S3	1	2	3

3.1.2 The Correlation algorithm

One problem with using the Co-occur similarity score is over-recommendation of popular modules, what we might call the ‘‘Harry Potter syndrome’’. Since so many people have purchased Harry Potter, a naive implementation of ‘‘People Who Bought This Book Also Bought...’’ at Amazon would cause Harry Potter to be recommended on almost every other book page. This is problematic because people are often aware already of the most common books or software modules and do not need them to be recommended. There is an additional problem when item similarity is based on co-mention in conversations. When two less popular modules are mentioned in a conversation thread, we speculate that there is often an interesting relationship between them, that they are either complements or substitutes for each other. When a popular module is mentioned in a conversation, however, we speculate that it is more often an incidental reference that does not necessarily indicate a strong relationship between it and the other modules mentioned in the thread.

Our second similarity function corrects for the Harry Potter syndrome by assigning scores based on whether the target occurs more frequently in conversations that mention the source item than in other conversations. The scoring function is simply the Pearson correlation coefficient [11] between two rows of the mention matrix M.

Recall that, for any matrix X with I rows and T columns, corr(X) is an IxI matrix giving the correlation coefficients:

$$corr(X)_{ij} = \frac{\sum_t (X_{it} - \bar{X}_i)(X_{jt} - \bar{X}_j)}{\sqrt{\sum_t (X_{it} - \bar{X}_i)^2} \sqrt{\sum_t (X_{jt} - \bar{X}_j)^2}}$$

Thus, for our second similarity algorithm, we have Correlation_{ij} = corr(M)_{ij}. Table 3 shows an example, using the module detection data from Table 1. Note that S2 and S3 are positively correlated, but S1 and S3 are negatively related even though they have one co-mention.

H1a: Correlation produces better recommendations than Co-occur.

Table 3. Correlation algorithm similarity scores

		Software Module		
		S1	S2	S3
Software Module	S1	1	-1	-0.58
	S2	-1	1	0.58
	S3	-0.58	0.58	1

3.1.3 The Recency algorithm

Our third similarity function takes into account the recency of conversations. The idea is to highlight ‘‘hotspot’’ modules that received a lot of attention in recent discussions. Modules not discussed recently could be due to abandonment, and thus should be given less weight in recommendations. It also corrects the potential problem that newly created modules would get overshadowed by older modules that have accumulated mentions over time.

The score function of the Recency algorithm uses the same correlation coefficient function as earlier, but on a different mention matrix M’, where:

$$M'_{it} = \max \left[0, \left(1 - \frac{age(t)}{600} \right) \right] \text{ if } M_{it} = 1; \text{ or } 0 \text{ otherwise}$$

That is, we apply a degradation function on the original mention matrix M that decreases the value of mentions linearly from 1 to 0 in 600 days. Age(t) is the number of days since the last time

Table 4. Conversation mentions matrix M’ corrected for recency

		Conversation Threads			
		T1	T2	T3	T4
Software Module	S1	0	0.9	0	0.5
	S2	1	0	0	0
	S3	1	0	0	0.5

Table 5. Recency algorithm similarity scores

		Software Module		
		S1	S2	S3
Software Module	S1	1	-0.54	-0.44
	S2	-0.54	1	0.87
	S3	-0.44	0.87	1

thread t was updated. Table 4 shows the original conversation mentions matrix M after correction for recency, where T1 through T4 were last updated today, 60 days ago, 2 years ago, and 300 days ago respectively.

Thus, for our third similarity algorithm, we have $Recency_{ij} = corr(M'')_{ij}$. Table 5 shows the double pivot under M'' , where S2 and S3 have a higher relevance score than in Correlation because T4's mention of S1 and S3 together became weaker after 300 days.

H1b: Recency produces better recommendations than Correlation.

3.1.4 The Uniqueness algorithm

Our last similarity function takes into account the “uniqueness” of module mentions in conversations. Intuitively, when a conversation mentions too many modules, those modules are not likely to have as strong relevancy to each other as those from a conversation mentioning only a few modules, which are somewhat “uniquely” related. The score function also uses the same correlation coefficient function, but modifies the mention matrix M , where:

$$M''_{it} = \frac{1}{\sum_j M_{jt}} \text{ if } M_{it} = 1; \text{ or } 0 \text{ otherwise}$$

That is, the weight of each module mention is normalized by the total number of mentions in the same conversation. Table 6 shows the original conversation mentions matrix M corrected for uniqueness. Thus, for our last similarity algorithm, we have $Uniqueness_{ij} = corr(M'')_{ij}$. Table 7 shows the double pivot under M'' .

Table 6. Conversation mentions matrix M'' corrected for uniqueness

		Conversation Threads			
		T1	T2	T3	T4
Software Module	S1	0	1	0	0.5
	S2	0.5	0	0.5	0
	S3	0.5	0	0.5	0.5

Table 7. Uniqueness algorithm similarity scores

		Software Module		
		S1	S2	S3
Software Module	S1	1	-0.9	-0.87
	S2	-0.9	1	0.58
	S3	-0.87	0.58	1

H1c: Uniqueness produces better recommendations than Correlation.

3.1.5 Overlaps of result sets

The four alternative double pivot algorithms generated quite different recommendation result sets, even though they used the same conversation co-mentions. For the top-100 most popular modules, we calculated average overlap of the top-five

recommended modules from the four algorithms. As shown in Table 8, all the algorithms produced distinct results.

Table 8. Alternative double pivot results overlap

	Co-occur	Corr	Recency	Uniq
Co-occur.	--	27.2%	29.4%	26.2%
Corr.		--	48.8%	55%
Recency			--	47.6%

3.1.6 Deployment

The algorithms were deployed on the live drupal.org site. We concurrently tested clickthrough for all four algorithms from Nov. 16 to Dec. 3, 2008. Each pageview was randomly assigned to display results from one of the four alternatives.

3.2 Comparison to “Recent Conversations”

In stage 2, we tested to see whether conversation pivots were a better use of screen real estate than other possible uses of the same space. The version we tested was the Correlation algorithm (a winner of the Stage 1 bakeoff, as will be described in the results section). The comparison baseline was the “recent conversations” block that simply displayed links to the ten most recent threads in the drupal.org forums. It was already displayed on the drupal.org homepage, but not on module pages.

H2: Conversation pivot and double pivot blocks will receive more clickthroughs than the “recent conversations” block.

3.2.1 Deployment

First, we measured clickthrough on the pivot blocks for 36 days between Dec. 4, 2008 and Jan. 8, 2009, using the Correlation algorithm. Then, we stopped displaying pivots blocks and instead displayed the “recent conversations” block on the module pages for the next 11 days between Jan. 9 and Jan. 19, 2009. Finally, we switched back to displaying the pivot block for another 17 days between Jan. 19 and Feb. 5, 2009.

3.3 Better Detection of Module Mentions

The conversation mention matrix M is constructed by a detection program that automatically infers references to modules as they are naturally expressed in messages, albeit with some error. The initial detection algorithm was crafted to minimize false positives, as we initially thought that false positives were more harmful than false negatives for conversation double pivot recommendations. The algorithm detected a mention if a module's complete title appeared exactly in the message, immediately preceded or followed by the word “module”. In addition, if a known alias for a module appeared exactly in a message, a mention was detected without requiring the adjacent word “module”. For example, most authors refer to the Drupal module titled “Content Construction Kit (CCK)” as “CCK” rather than using the full title. Using domain knowledge, we manually built a small list of such commonly used aliases.

We ran the detection program on the complete drupal.org forums history as of Feb.5, 2009. Of 588,246 messages, 75,266 contained at least one module reference. 1,645 of the 4,199 modules were cited in at least one thread and those had a median of 4 references. The total number of detected module references was 89,216.

To measure precision and recall of the detection program, one of the authors manually coded module mentions in all 4200

messages posted on drupal.org forums between Nov. 20 and Nov. 30, 2008.¹ Those 4200 messages referenced a total of 472 modules, and each module had an average of 3 mentions.

Using the 4200 coded messages, we calculated precision and recall as 85.6% and 27.1% respectively. Recall was very low because of the conservative detection algorithm. For example, the 34th most popular module according to download statistics, “Printer, e-mail and PDF versions”, was detected only six times in forum conversations, because it had no alias in our initial list, and people rarely referred to it using the long title.

This raised the question of whether improved recall for the module mention detection algorithm, at the expense of some precision, would improve the effectiveness of the double pivot module recommendations. To increase recall, we used a few more aggressive string matching heuristics, such as:

- If a module title has more than one word, then match on the module title without requiring the word “module”.
- If a module name is a complex word such as “ImageCache”, then match on the module title without requiring the word “module”.
- Detect references to the modules’ drupal.org URLs and short-codes.
- If a module’s title has two words, such as “Organic Groups”, match on “organic_groups” and “organicgroups” as well.
- Use additional aliases we found, such as “AF” for “Advanced Forum”.

Applying those new heuristics to the detection program, we found 2,603 modules, a 58% increase, cited in at least one thread. The total number of module mentions detected increased to 216,325, raising recall from 27.1% to 84.1%. Precision dropped only slightly, from 85.6% to 84.4%.

We ran the Correlation double pivot algorithm again using the new detection program. We called it Correlation-HR because it differed from the original Correlation algorithm only by using the high recall module detections. The top-five recommended items lists for Correlation and Correlation-HR had only a 27.4% overlap: they generated quite different results.

H3. Correlation-HR produces better recommendations than the original Correlation.

3.3.1 Deployment

The pivots block continued to be displayed on the drupal.org website from Feb 5. through April 24, but the Google Analytics tracking of clickthroughs was turned off during that period. We first tracked the original Correlation double pivot from April 24-29, 2009, in case there had been a change in clickthroughs from the previous tracking period. Then we switched to Correlation-HR from April 29 - May.4. Because we found that clickthrough rates on weekends were different from rates on weekdays, we made sure that both periods included three weekdays and a weekend. We also continued to track Correlation-HR from May 5-26.

¹ To check reliability, the other author randomly coded 100 messages. The two raters agreed if they applied the same set of module labels. The agreement was 92%, Cohen’s kappa was 0.85

3.4 Alternative Algorithms

Our last stage compared the Correlation-HR algorithm to other recommender approaches. The first alternative approach was based on co-installation, with the assumption that modules installed on the same site were somewhat related. Drupal.org keeps track of module installation statistics on live sites, if the hosting sites choose to share it. One daily snapshot of such installation data on Feb. 4, 2009 consisted of 225,962 sites with an average of 13 modules installed on each site.

The module installation matrix N has $N_{is}=1$ if module i is installed on site s , and $N_{is}=0$ otherwise. The scoring function was simply the correlation coefficient between two rows of N . Thus, we have $\text{Corr-install}_{ij}=\text{corr}(N)_{ij}$.

We speculate that discussion co-mentions might indicate stronger relevance than co-installations because a Drupal site might install many unrelated modules. Moreover, we observe that substitute modules that offer similar functionality should rarely be installed together, and so Corr-install recommendations will miss the recommending of substitutes.

H4a: Correlation-HR produces better recommendations than Corr-install.

The second alternative recommender used a Vector Space Model [10] content similarity algorithm to compare the texts describing the modules. We used the open source tool “ApacheSolr More Like This”² (SOLR) to generate the recommended items list using module title texts and module description texts, sometimes up to several paragraphs, provided by the module authors. SOLR gives higher weight to terms appearing in the title than the description.

SOLR has an option to use only a limited number of terms, those with the highest tf-idf weights. That strips off unimportant noisy terms that might harm the quality of the similarity results. We used three instantiations of the SOLR algorithm with the terms limits set to 10, 30, and 100, and named them as SOLR-10, SOLR-30, and SOLR-100 respectively. Table 9 shows that the three instantiations generated quite different result sets.

Table 9. Result set overlaps of double pivot and other approaches

	Corr.-HR	Corr-install	SOLR-10	SOLR-30	SOLR-100
Corr.-HR	--	20.6%	15%	17.4%	12.2%
Corr-install		--	8%	10%	6.8%
SOLR-10			--	40.6%	21.6%
SOLR-30				--	42.2%

We observed that not all Drupal modules had good text descriptions and hypothesized that that would hamper the performance of the text-based algorithms.

H4b. Correlation-HR produces better recommendations than SOLR-10, SOLR-30 and SOLR-100.

² <http://lucene.apache.org/solr/>

3.4.1 Deployment

The alternative algorithms and Correlation-HR were deployed on the drupal.org site from May 27 to July 22. Each pageview was randomly assigned to display results from one of the five alternative algorithms.

4. RESULTS

4.1 Stage 1 Comparisons: Alternative Double Pivot Algorithms

Table 10 shows that in the stage 1 bakeoff, Correlation and Uniqueness had the highest clickthrough rates. Except for Correlation compared to Uniqueness, all other differences were statistically significant (pairwise comparisons of population proportions, $p < .05$).

Thus, the results support H1a: all three correlation coefficient based algorithms performed better than Co-occur, in both clickthroughs and relevance outcomes. Surprisingly, however, the corrections for recency and uniqueness did not improve performance as predicted by H1b and H1c. We declared Correlation the winner of the stage 1 bakeoff since it had a slightly higher clickthrough than Uniqueness, although the difference was tiny.

Table 10. Stage 1 clickthrough results: Nov. 16-Dec. 3

Algorithm	Pageviews	Clickthrough
Co-occur	238,709	0.09%
Recency	238,359	0.11%
Uniqueness	239,590	0.13%
Correlation	238,879	0.13%

4.2 Stage 2 Comparison: Clickthrough to Pivots vs. “Recent Conversations”

Figure 3 shows the daily clickthrough for the testing period of the stage 2 comparison. With the pivots block displayed, clickthrough was 0.18% for the 5 recommended conversation items, 0.28% for the 5 recommended modules, and 0.45% for the overall 10 links. During the 11-day comparison period with the “New Forum Posts” block displayed instead, the overall clickthrough dropped to 0.14%. When the pivots block was turned back on, the overall clickthrough partially recovered to 0.34%, consisting of 0.14% from recommended conversations and 0.20% from recommended modules. The differences between overall clickthrough rates before and after each switch were statistically significant. (pairwise comparisons of population proportions; $p < .05$). H2 was supported.

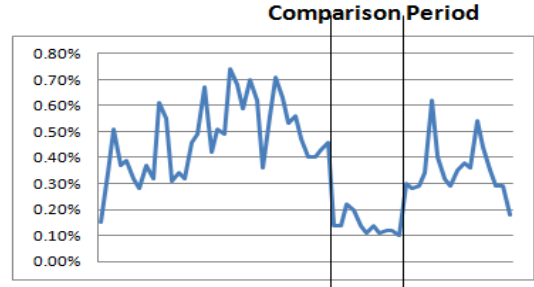


Figure 3. Daily clickthrough between Dec.4, 2008 and Feb.5, 2009.

4.3 Stage 3: Effects of High Recall

The third stage compared the Correlation algorithm with and without the higher recall detector of module mentions. Clickthrough with the original Correlation algorithm in the 5 days from April 24-29, 2009 was 0.64%. In the five days after switching to Correlation-HR, clickthrough jumped to 1.07%, more than a 50% increase. Clickthroughs remained relatively stable for the next 21 days, using Correlation-HR.

Table 11. Stage 3 clickthrough results

Dates	Algorithm	Pageviews	Clickthrough
April 24 - 29	Correlation	289,866	0.64%
April 29 – May 4	Corr-HR	315,301	1.07%
May 5 – May 26	Corr-HR	1,481,350	1.05%

4.4 Stage 4: Comparison to Other Recommender Approaches

The Correlation-HR algorithm was not able to fill all five of its recommendation slots on pages for modules that were not co-mentioned with at least five other modules in the Drupal forums. Figure 4 shows the percentage of possible recommendation slots that the Correlation-HR algorithm produced. Modules are ranked by popularity. For each decile, the bar shows the percentage of the possible recommendation slots (five for each of the source modules) that were filled by the algorithm. Thus, the most popular tenth of the modules had nearly 90% of the possible

Figure 4. Recommendation Slots filled by Corr-HR

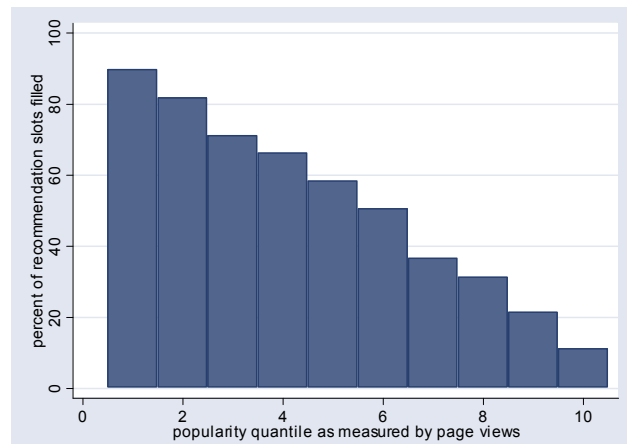


Table 12. Stage 4 clickthrough results: all modules

Algorithm	Pageviews	Clickthrough
Corr-HR	884,834	1.12%
Corr-install	920,099	0.80%
SOLR-10	949,283	1.22%
SOLR-30	949,225	1.15%
SOLR-100	950,394	1.04%

recommendation slots filled, while the least popular tenth had only 11% of the possible recommendation slots filled.

Correlation-HR had significantly higher clickthrough than Corr-install, as shown in Table 12 (1.12% > 0.80%, pairwise comparisons of population proportions; $p < .05$). H4a was supported. Contrary to H4b, we found that Correlation-HR had significantly lower clickthrough than SOLR-10 and SOLR-30.

Note that even though pageviews were randomly assigned to the five algorithms, Correlation-HR had fewer registered pageviews. This occurred because when Correlation-HR made no module recommendations, our code did not generate a Google Analytics event tracking code. In order to correct for that, we estimate the number of pageviews for such modules with algorithm Correlation-HR as equal to the number of pageviews for those modules with SOLR-10. With that correction, the clickthrough rate on Correlation-HR recommendations goes down to 1.04% overall, about the same as SOLR-100.

In a post-hoc analysis, however, we found that Corr-HR had a higher clickthrough rate than any of the SOLR algorithms on the more popular modules. The top 206 modules garnered half of all page views, with the remaining 4,006 getting the other half. On these modules, Corr-HR had a clickthrough rate of 0.91%, while SOLR-10 had 0.82%, as shown in Table 13. Interestingly, all the algorithms had higher clickthrough rates on recommendations from the pages for the less popular modules than for the more popular modules. Even Correlation-HR, while it recommended fewer items, had a higher clickthrough rate from pages for less popular modules than more popular modules.

Table 13. Stage 4 clickthrough results by module popularity

	Algorithm	Pageviews	Clickthrough
More popular modules	Corr-HR	* 477,430	0.91%
	SOLR-10	475,227	0.82%
Less popular modules	Corr-HR	* 476,403	1.18%
	SOLR-10	474,056	1.63%

* Pageviews for Corr-HR include estimated pageviews for modules where pageviews were not tracked because no recommendations were made.

5. DISCUSSION AND FUTURE WORK

We explored how the co-mention of modules in conversations can be detected and used as a resource for software module recommendation. We found that the correlation coefficient

algorithm worked better than merely counting co-occurrences. We attribute that to the fact that it corrects for the Harry Potter syndrome, over-recommendation of popular modules.

The additional corrections for discussion recency did not work as well as expected. We speculate that it leads to unstable results: a random recent conversation on two remotely related modules might override strong relevancies established over a long period of time. Our results suggest that reduction of variance from larger numbers of conversations is more valuable in this domain, even if the extra conversations are old.

We also found that detection of more module mentions in conversations dramatically improved the quality of recommendations, even though the detection algorithm then had more false positives. Contrary to our original argument in [14] that precision of the detection algorithm is most critical, we found that high recall was vital to the double pivot algorithm's performance.

Improving recall has another side effect, too: it picks out more mentions of obscure, unpopular modules. If an obscure module is mentioned only once or twice, always together with a popular module, the correlation algorithm will cause the obscure module to be displayed at the top of the recommendation list for the popular module. Thus, improving recall could lead to over-recommendation of unpopular modules. In post-hoc analysis of Stage 4, however, we did not find a difference in probability of clickthrough to recommendations of more or less popular algorithms.

Clickthrough rates increased more than tenfold from the beginning to the end of the tuning process. Overall, this suggests that clickthroughs were indeed sensitive to the quality of recommendations and that developers of recommenders should be prepared to tune their recommenders over time.

A more than threefold increase in clickthroughs on recommendations occurred in the three months between Stage 2 and Stage 3, a time during which the recommendation block was installed and the algorithm was unchanged, but clickthroughs were not tracked. One possible explanation is that users became accustomed to the presence of the recommendations block and looked at it more often. This explanation is consistent with the apparent slow rise in clickthroughs in the first, long part of Figure 3 during December, 2008. When recommendations were relatively stable over time, people may have begun to use them as re-finding aids: someone who forgot the name of a module but remembered the page from which it was recommended might click through again from the remembered page. Another possible explanation is that other changes were made to the display of module pages, which could have affected the population of visitors to those pages and their receptivity to module recommendations. A final possibility is that we changed the Google Analytics tracking functions used between stage 2 and stage 3—while Google's documentation indicates that the numbers reported in stages 2 and 3 should be comparable, it is possible that there was some underlying change that we are not aware of.

Conversation double pivots were more effective than the recommender based on co-installation. Perhaps this was because most sites only installed a few popular modules, and those popular modules were calculated as related even though they were not useful recommendations. Even so, co-installation can still be

valuable in the sense that it can distinguish substitutes from complements: if two modules are related but rarely found in co-installation, then they are almost certainly substitutes. We suspect that recommending substitutes would be especially valuable to users: a common use case would be searching for a module that performs some function, finding one, and navigating from there to substitutes that are of higher quality.

Conversation double pivots performed better than the text-based matching algorithm on pages for the more popular items where conversation mentions were plentiful. The best text-matching algorithm, however, performed better on recommendations from the less popular module pages, and better overall.

Somewhat surprisingly, users followed the recommendation links much more often from less popular module pages. There are two plausible explanations. One is that more users who visit the popular pages already know they want those modules, and are just visiting to actually download them, while more visitors to less popular modules are still exploring to find modules that will meet their needs. Another is that, while exploring, more users find that the popular modules are just what they need and stop exploring, while visitors to pages for less popular modules find that they need to keep looking.

There are other small fixes that could be applied to refine the recommenders. For example, we could remove the extremely popular and unpopular modules from all result sets, because they are not likely to be high quality recommendations anyway (unpopular items because they may be of low quality; popular items because everyone knows about them already). Also, rather than displaying the top five items, we could randomly display five of the top ten recommendations, in order to provide more variety.

Overall, the results suggest that there is great value in tuning module recommendations. They also suggest that a hybrid approach may be better than any one algorithm alone. Conversation co-mentions may be better for modules that are mentioned a lot, and text matching better for other modules. Perhaps the best approach, in the end, would be a semi-automatic recommender: use the automatic algorithms only to suggest a list of recommended candidates, and then have the users pick, collectively, the related modules to recommend. This is the direction we are heading to in our future work.

Our final reflection goes back to the basic question: what module recommendations are really useful for different users and usage scenarios? Novice users might want to see more popular modules; veterans might be more adventurous and prefer those remotely related modules that might inspire them for new applications. Some users might like to see related modules, but others might prefer to use the five recommended links as shortcuts to the most popular modules. This question needs to be addressed in future work.

To conclude, we assessed conversation co-mentions as resources for software module recommendation, and found they were indeed useful. We hope to see more applications in other domains that mine recommendations from conversation.

6. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under award IIS-0812042. We thank Kieran Lal, Gerhard

Killesreiter, and other drupal.org infrastructure team members for their effort deploying pivots on the production server. We thank Michael Hess, Nancy Douyan and Nathan Oostendorp for their advice and help.

7. REFERENCES

- [1] Bennett, J., and Lanning, J. 2007. The Netflix Prize. In *Proceedings of KDD Cup and Workshop, 2007*.
- [2] Breese, J., Heckerman, D., and Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of 14th Conference on the Uncertainty in Artificial Intelligence*
- [3] Brin, S., and Page, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7), 107-117
- [4] Drenner, S., Harper, M., Frankowski, D., Riedl, J., and Terveen, L. 2006. Insert movie reference here: a system to bridge conversation and item-oriented web sites. In *Proceedings of the SIGCHI Conference*.
- [5] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. 2004. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* 22, 1 (Jan. 2004), 5-53.
- [6] Joachims, T., Granka, L., Pan, B., Hembrooke, H., and Gay, G. 2005. Accurately interpreting clickthrough data as implicit feedback. In *Proc. of the 28th ACM SIGIR Conf.*
- [7] Linden, G., Smith, B., and York, J. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (Jan. 2003), 76-80
- [8] Morita, M. and Shinoda, Y. 1994. Information filtering based on user behavior analysis and best match text retrieval. In *Proceedings of the 17th ACM SIGIR Conference on Research and Development in information Retrieval*.
- [9] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*.
- [10] Salton, G., Wong, A., and Yang, C. S. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11 (Nov. 1975), 613-620.
- [11] Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international Conference on World Wide Web*.
- [12] Small, H. 1973. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science* 24, 4.
- [13] Terveen, L., Hill, W., Amento, B., McDonald, D., and Creter, J. 1997. PHOAKS: a system for sharing recommendations. *Commun. ACM* 40, 3 (Mar. 1997), 59-62
- [14] Zhou, D., Oostendorp, N., Hess, M., and Resnick, P. 2008. Conversation pivots and double pivots. In *Proc. of the 26th SIGCHI Conf. on Human Factors in Computing Systems*.