Expert as a Service: Software Expert Recommendation via Knowledge Domain Embeddings in Stack Overflow

Chaoran Huang *, Lina Yao*, Xianzhi Wang†, Boualem Benatallah*, Quan Z Sheng‡
*UNSW Sydney, Australia
chaoran.huang@unsw.edu.au
†Singapore Management University, Singapore
xzwang@smu.edu.sg
‡Macquarie University, Australia
michael.sheng@mq.edu.au

***Abstract*—Question answering (Q&A) communities have gained momentum recently as an effective means of knowledge sharing over the crowds, where many users are experts in the real-world and can make quality contributions in certain domains or technologies. Although the massive user-generated Q&A data present a valuable source of human knowledge, a related challenging issue is how to find those expert users effectively. In this paper, we propose a framework for finding such experts in a collaborative network. Accredited with recent works on distributed word representations, we are able to summarize text chunks from the semantics perspective and infer knowledge domains by clustering pre-trained word vectors. In particular, we exploit a graph-based clustering method for knowledge domain extraction and discern the shared latent factors using matrix factorization techniques. The proposed clustering method features requiring no post-processing of clustering indicators and the matrix factorization method is combined with the semantic similarity of the historical answers to conduct expertise ranking of users given a query. We use Stack Overflow, a website with a large group of users and a large number of posts on topics related to computer programming, to evaluate the proposed approach and conduct extensively experiments to show the effectiveness of our approach.**

***Keywords*-Knowledge discovery; Stack Overflow; Expertise finding; Question answering; Expert as a Service**

I. Introduction

Nowadays, Question and Answering (Q&A) websites represent the most popular platforms or collaborative networks for knowledge sharing. These websites allow online users to raise and answer questions, which form a huge amount of user-generated content. Besides disseminating knowledge over the web, such content provides important information about the expertise of users that can be used by various applications. We describe a typical scenario of such applications in human resource management as follows.

Alice is an HR officer responsible for recruiting new employees in a software company. After screening hundreds of résumé, she selected 30 of them for further review. However, even examining the 30 applicants could cost considerable efforts. Fortunately, she finds many applicants have included collaborative network accounts such as GitHub[1] or Stack Overflow[2] accounts in their résumés. Usually, Alice needs to browse their profile pages to have a better understanding of applicants' skills, Alternatively, Alice can use our system to help make decisions based on applicants' online information. In this case, she only needs to make inquiries about user profiles by describing the desired jobs using some keywords or sentences in the system. Our system will then list the users related to certain websites and show all detailed information of applicants. The information is a briefed version of users' profile pages that contain basic built-in metrics such as user reputation, badges numbers, answers numbers, question numbers of users, people reaches, profile descriptions, recent posts, as well as expertise scores of users in different areas. Based on such information, Alice can easily locate and gain a reliable insight to the skills of applicants. The above system is a just a showcase of many important applications that can leverage user expertise information underlying the Q&A websites, and upon which, we propose to deliver **Expert as a Service**.

Generally, the quality of answers highly depends on contributors' skills and experience, thus the quality of answers are not guaranteed. Take the Stack Overflow(SO) for example, it is one of the most successful Q&A websites in Stack Exchange Networks[3]. As a popular online community, it includes over 5 million users who post questions and offer answers on the topic of programming. For this reason, Stack Overflow introduces a voting system that allows each user to vote for or against answers. The question is closed once an answer is accepted by the questioner. Similar to all online communities, activities in SO highly depend on user participation and contribution. According to statistics provided by the website, 73% of 11 million questions on SO has been answered and closed, and around 3 million questions are still waiting for answers. Besides, among those answered questions, only 55.37% of them have accepted an-

[1] github.com
[2] stackoverflow.com
[3] stackexchange.com

swers[4]. Given this huge amount of user-generated contents, it is highly possible to mine expertise information of users, and to recommend experts to answer the unsolved questions based on it. The expert recommendation not only helps improve user participation but also reduces the unanswered questions. Both results will attract more users to engage in the community and speed up the completion of domain knowledge.

Usually, a question may be unsolved as a result of being unnoticed by or unattractive to the majority [1]. The tagging system of SO helps users to find their interested questions by allowing questions raisers to assign up to 5 tags to each question. Thus, potential answerers can browse questions according to their tags of interest instead. However, although experts finding can be conducted based on tags [2], [1], tagging systems themselves are a passive mechanism. In addition, the tags in SO rely on human creation, which is often unreliable, ambiguous, or even distracting to people who are unfamiliar with certain terms, not to mention a user could use improper tags. Other existing research for expert recommendation are based on the establishment of user profiles, which measures user expertise in certain domains [3], [4], [5], [6]. Although experts can be recommended according to the specific tags, tags themselves have inherent limitations. For example, an expert of a general tag may not necessarily cover the knowledge required to answer a specific question. An example is that an expert in computer programming may be unable to answer questions related to certain programming languages that he/she is not familiar with.

In view of the above challenges, we propose a novel approach for recommending software experts, as a service. Given some query text chuck in the form of either keywords or sentences stating a certain question, the approach enables to find experts of certain domains. Unlike previous approaches based on user performance, our approach builds on a conceptual understanding of the content of previous answer posts. We first train word embedding based on user-generated text data (mainly content of posts) and then derive expertise domains by clustering word representations. Finally, we group the top existing answers and map the answers to specific domains to recommend experts. Our contributions in this paper is a novel framework for expert finding in the software domain, wherein users' explicit and implicit knowledge domains are discerned via graph-based clustering and latent factorization. Both of them are combined with a software-specific knowledge base to better discover the most capable experts for software related questions.

[4]Stack Overflow public dump on 10 March 2016, where 6,120,191 questions among total 11,053,469 have accepted answers; Available at http://archive.org/download/stackexchange.

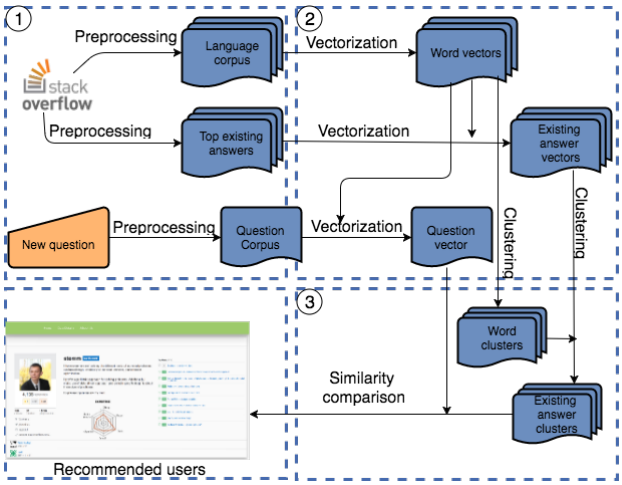


Figure 1. Workflow of Expert Recommendation, the main component of an Expert as a Service architecture.

II. Our Approach

The goal of this paper is to find users with relevant background to answer new or untouched questions. Our approach mines existing posts by conceptual modeling and understanding of the textual content of posts. The approach includes three main stages, namely:

1) **Post Representation:** This stage is aimed at transfer text-based posts into computing-friendly vectors. To achieve this, we first preprocess posts, filtering out poor quality posts, redundant and irrelevant information in posts, so that the post are ready to be for word representation learning. Lastly, based on learned word vectors, each post can be converted into a vector representation.

2) **Expertise Domain Extraction:** As we aforementioned that, tag information indicating expertise is not reliable and trustworthy sometimes. At this stage, we apply a graph-based clustering algorithm to automatically extract domains over posts, to divide posts into different domain expertise. Besides, this process can also help to avoid devoting huge amount of time to computing score of unrelated posts.

3) **Expert Recommendation:** The last stage is related to the input query (a post) by a questioner, which we apply the same procedure in the second stage to vectorize the input and then map it to one of the extracted expertise domains. Hence a list of answerers who gave highly voted answers will be recommended to the questioner.

The framework is shown in Fig. 1, and we will introduce each of the three stages in the following subsections.

A. Post Representation

1) Post Preprocessing: Before learning representations of posts in Stack Overflow, one critical step is to preprocess the posts. The goal of data preprocessing is to eliminate the redundant information as well as regularize the formats of data.

As the first step, we remove the symbols that are apparently irrelevant to the main text using regular expressions and the annotations used only to formatting text in web pages. Meanwhile, we retain the common words and grammatical or structural words and reserve the comments and tags. We choose to handle the irrelevant symbols and format annotations later because these words usually contain important semantic information that affects the other text and we still need such information to obtain as accurate word vectors as possible in the following step. Tags are also useful as they often help determine the types of programming language related to the question. Note that SO is a programmer-oriented website, where code examples and demonstrations are often provided as supplements to the description of questions or solutions. Although codes themselves can be difficult to handle, we still kept comments which are often meaningful text as well.

To be able to determine the users who can most possibly answer the input question well, we follow by considering the questions and the answers that are likely to give good inferences toward users with good expertise. In particular, we select questions along with their top-voted answers, i.e., the answers with the most numbers of up-votes. By performing such selection, we can reduce the size of input by ruling out the irrelevant or redundant information and noises without losing the necessary information. The final step of preprocessing is to eliminate the remaining irrelevant words to further enhances our speed and accuracy of future processing. Despite complicated effective methods available, we simply use a dictionary of software-related terms to filter out the irrelevant words in the remaining text cause the filtering is not this paper's main focus,

2) Word Representation: Word embedding is a set of techniques for natural language processing, which represent words or phrases by generating and using some lower dimensional vectors. Word embedding has recently gained success in many applications which further boost its application. Currently, most of the existing studies use distributed word representations[7]. A popular state-of-the-art technique for distributed word presentation training is Word2Vec [8]. This technique starts by using random vectors to present words in the vocabulary, and then goes through the corpus to examine surrounding words of every word within a certain window. In this way, words far enough from a specific word will less likely to be relevant. This approach differs with distributional word representation in that it directly predicts instead of counting and calculating distributions. Word2Vec can learn raw texts and produce word representation vectors efficiently by using a neural probabilistic language model built on a two-layer neural network. Same as many other neural networks, Word2Vec uses a Softmax function to calculate probabilities based on the activation status of neurons in the network.

3) Post Representation: Since this study focuses on text chunks rather than single word, it is essential to represent posts as well. Although often short in length, a post, regardless question or answer, can be seen as a type of document. A traditional way of modeling documents is using term frequency [9]. Specifically, given a document (post) $d \in D$ we establish a set of distinct terms $T = \{t_1, t_2, ..., t_n\}$ that occur in D. Then the document d can be represented as a vector of dimension n, where each element corresponds to terms in document d and its value is frequency of the term, denoted as $tf(d, t)$. Thus, we have a vector representation of document d:

$$\vec{t_d} = (tf(d, t_1), tf(d, t_2), ..., tf(d, t_n)) \tag{1}$$

This term frequency-based vector stands on the assumption that the more important a word is, the more frequent it should be. Yet, this assumption is not always true, as the highly frequent words in English are often articles or conjunctions, which do not have much practical meaning. Although more complex weights such as TF-IDF can be computed instead of term frequency, those models are not sensitive to semantic similarities of different words.

To better represent posts, we take advantage of word representations to naturally contains semantic information of words. Based on the pre-trained language model $T = \{T_1, T_2, ..., T_n\}$, which is representations of the terms in a documents set D, given any document d with its term frequency vector $\vec{t_d}$, we can summarize this document using the weighted average sum of word vectors, i.e.,

$$\vec{t_d'} = \sum T_i \times tf(d, t_j) \tag{2}$$

where terms t_j correspond to word vector T_i.

After the quantitative vectorized representation of posts, we use a vector distance metric, i.e., cosine similarity, to compute the similarities among posts in terms of the angle of two vectors. Given two documents $\vec{t_1}$ and $\vec{t_2}$, their cosine similarity is computed by using the following equations.

$$d_{cosine} = \frac{\vec{t_1} \cdot \vec{t_2}}{|\vec{t_1}| \times |\vec{t_2}|} \tag{3}$$

B. Expertise Domain Extraction

Given the huge amount of information (e.g., millions of instances) to be processed, it is not feasible to calculate similarities between inputs due to the possible tremendous amount of time consumed. In this paper, we group our data into different clusters to save the computing time. Traditionally clustering is performed directly on documents, e.g., one

ALGORITHM 1: Our algorithm to graph construction for clustering algorithm, it takes pre-trained word vectors W as inputs and returns a graph G

Input: W
Output: G
Algorithm $GraphConstruct(W)$
for $e = (u, v) \in G$ **do**
  $d \leftarrow distance(u, v)$
  if $d > 0$ **then**
    $\Omega(e) \leftarrow d$
  else $\{d \leq 0\}$
    $\Omega(e) \leftarrow 0$
  end if
end for
return G

popular clustering technology based on tagging a sample of documents manually as ground truth to infer the rest [10]. The disadvantage lies in the difficulty of handling the variability of tags for certain items, spamming tags, as well as ambiguous tags. Another mostly used way of clustering data is term based [11], but this method has similar drawbacks to the naïve term frequency based approach in representing documents, i.e., it treats synonyms and words of different forms as different terms. In comparison, recent research shows that semantic-based approaches are more faster and easier to use [12], [13], [14], where word vectors [15] and document representations [16] are useful for clustering.

Based on the above analysis, we follow the similar idea as [15] for expertise domain extraction based on the output of word vectors from preprocessing. We first cluster word vectors based on their semantic similarities by applying CLR graph-based clustering. The words in each cluster share a particular concept, which we call a topic. As CLR clustering requires a graph as input, we construct a full connected, weighted graph with a filtered word representations vocabulary, as shown in Algorithm 1. We initialize edges with zero weights and then update the weights with cosine similarities of their nodes. Note that the clusters can also be seen as special kind of documents when each cluster is presented with their element words and each term only occurs once. Our previous approach for computing document representation can be applied to calculate the centroids of clusters. This approach meets Kalogeratos' approach of generating global context vector [15].

We further present the Constrained Laplacian Rank Algorithm (CLR)-based clustering method to extract expertise domains in Algorithm 2. It is a novel parameter free spectral method for graph-based clustering and requires no further processing about cluster indicators [17]. Different from the popular way of using Gaussian kernel function [18], this method takes advantage of the application of sparse

ALGORITHM 2: Our algorithm to cluster posts, it takes pre-trained word vectors W, vectorized posts P and clustering number n as input, and returns them in clusters

Input: P, n
Output: C
Algorithm $PostClustering(P, n)$
$C \leftarrow Cluster(n, W)$
for $c \in C$ and $c_{centroid} \in C_{centroid}$ **do**
  $c_{centroid} \leftarrow mean(c)$
end for
for $p \in P$ **do**
  for all i corresponding to $C_i \in C$ such that minimize $d_{cos}(p, C^i_{centroid})$ **do**
    add p into C_i
  end for
end for
return C

representation of matrices [19], [20] to compute the similarity matrix. The sparse representation of matrices can be approximated by solving ℓ_1 minimization problems, which are solvable via standard linear programming [21].

Specifically, given data $X_{d\times n} = (x_1, x_2, ..., x_n)$ of the size d and n dimensions, the CLR-based clustering aims at computing a representation vector β, satisfying that a new observation can be approximated by:

$$y \approx X\beta \tag{4}$$

Instead of finding a sparse solution, the penalty version of its ℓ_1 minimization is used as:

$$\min_{\beta} \|X\beta - y\|_2^2 + \lambda\|\beta\|_1 \tag{5}$$

where λ is a parameter greater than 0.

Assume that we want to compute the similarities among the number k vector and others. We note $X - x_k$ as X'_k, meaning β is the vector of similarities and x_k is the y shows in the above equation. Suppose the majority of similarities are positive values, the computation turns to minimize the sum of residue error. By considering that data X can be identified by a constant shift vector t, the shift invariant similarities can be obtained by:

$$\|(X'_k + t1^T)\beta - (x_k + t)\|_2^2 = \|X\beta - y\|_2^2 \tag{6}$$

By setting the constraint as $\beta^T = 1$, the problem can be simplified as:

$$\min_{\beta} \|X\beta - y\|_2^2 \text{ such that } \beta > 0, \beta^T = 1 \tag{7}$$

ALGORITHM 3: Our algorithm to infer recommended users, it takes vectorized input question q, clustered Posts C with its centroid set $C_{centroid}$ as input, and returns a list of limited top recommended users A

Input: $q, C, C_{centroid}$
Output: A
Algorithm $UserRecommend(q, C, C_{centroid})$
for $c_{centroid} \in C_{centroid}$ **do**
  $d_{cq} \leftarrow d_{cos}(q, c_{centroid})$
  put d_{cq} in distance set D_{cq}
end for
$C_q \leftarrow C$ such that the corresponding $d = \min(D_{cq})$
for $ainC_q$ **do**
  compute $d_{aq} \leftarrow d_{cos}(a, q)$
  put d_{aq}, into D_{aq}
end for
initialize list A
$A' \leftarrow rank(D_{aq}, \ell)$ {$rank(S, k)$ returns top k result in set S}
$A = getAuthors(A')$ {$getAuthors(P)$ returns authors of Posts in P}
return A

C. Expert Recommendation

Once clustering existing posts is completed, new questions can be accepted for the expert recommendation. Therefore, the task of this step is to infer the users best qualified to answer the input question. Our recommendation approach finding experts in certain knowledge domains. Specifically, the input question is first mapped to one of those domains obtained by Algorithm 2. This helps reduce the number of instances in the search stage while still gives us a reasonable chance of finding past some answers that are semantically similar to the input question. Since the remaining data may still be large in quantity, as the second step, we apply matrix factorization techniques to further reduce the time spent on ranking and recommending experts.

Note that, the accepted answer may be good enough for the questioner, yet may not be the most valuable one(most of the time, it is the best answer indeed). Sometimes, the unaccepted answers containing precious knowledge, especially when the answers are too general and not suitable for the specific question. Based on this consideration, we take the up-vote of answers as an indicator of the quality of answers and keep top k up-voted answers with threshold t, including the accepted one, to establish the recommendation model. Basically, SO has a comprehensive voting system, where users can vote a post up or down and a score of a post is the difference of up and down votes. Despite sometimes an individual may be not professional in some domain area and give a wrong decision, we can still claim that such voting information is reliable as SO has a large group of users.

To use the voting information effectively, we extend our evaluation of user expertise by factorizing the post-user interaction matrix of up-voting scores. The obtained latent user performance factors can later be combined with our previously computed scores to perform recommendations. Matrix factorization is a latent factors model, which does well in handling sparse data and has been widely adopted by modern recommendation systems [22], [7]. MF usually starts with a given sparse user-item matrix and de-composite it into the user latent factors multiplying with item ones. Here we factorize the post-contributor matrix of answer scores, where a score is the difference of up-votes and down-votes assigned to posts.

Given matrix $V_{N\times M} = WH$, where V is the post-contributor matrix that contains voting information from M users in N questions, we apply the Non-negative Matrix Factorization(NMF) technique and define the loss function as:

$$\pounds_{loss} = \underset{W,H}{\operatorname{argmin}} \frac{1}{2}\|X - WH\|_F^2 = \frac{1}{2}\sum_{i,j}(X_{i,j} - WH_{i,j})^2 \tag{8}$$

where $\|\cdot\|_F^2$ is the Frobenius norm of the matrix.

With an Elastic Net regulator, which combines ℓ-1 and ℓ-2 norms, along with parameter ρ controls ℓ-1 ratio and α regulates ℓ-2 intensity, we have this regulation function:

$$\pounds_{reg} = \alpha\rho\|W\|_1 + \alpha\rho\|H\|_1 + \frac{\alpha(1-\rho)}{2}\|W\|_F^2 + \frac{\alpha(1-\rho)}{2}\|H\|_F^2 \tag{9}$$

Now, the objective function turns into:

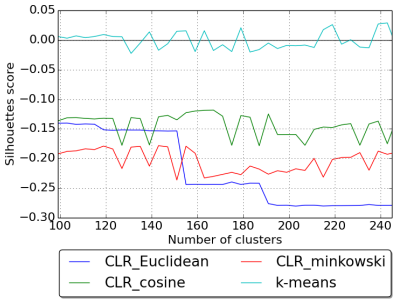
$$J = \pounds_{reg} + \pounds_{loss} \tag{10}$$

Having the matrix learned, we can now combine it with similarity scores, which indicates the semantic distance of posts to the domain of the input question. As the learned matrix is still sparse, using it as a weight would not be practical. For this reason, we sum up similarity scores with parameter λ multiplying the learned voting information. Again, by comparing the vectorized input question q with clusters set C, we can define the scope to search. Algorithm 3 shows the calculation of documents distance within the cluster on their representations. The algorithm finally recommends top-ℓ answerers, who gave the top similar answers set A, as experts.

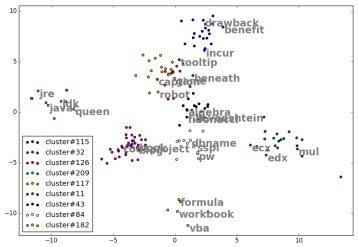
III. Experiments

A. Dataset

By training on extracted natural language corpus from the post title, body, and comments, we obtain word vectors contain a vocabulary of size 1,346,955. This vocabulary, unavoidably, included some useless words such as "ping-test" or user names. Those words may have limited influence

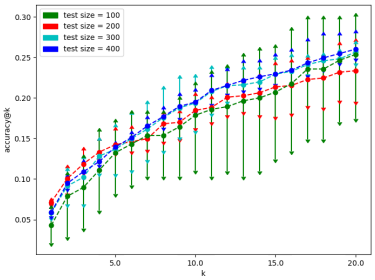


(a)

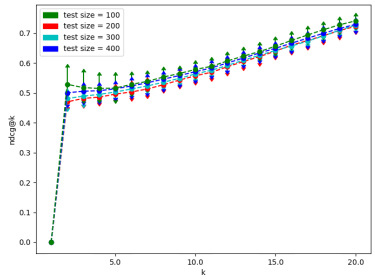


(b)

Figure 2. (a) Silhouettes score for different cluster number, suffix followed by CLR is the distance metrics used in Algorithm 1; (b) Example of selected word clusters, all points with same colour shown above belong to one cluster

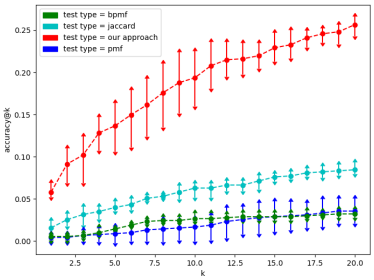


(a)

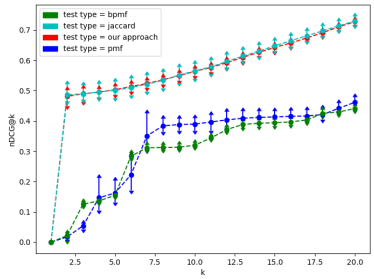


(b)

Figure 3. Accuracy and nDCG at top-k of 3-fold tests, with different test size, $\lambda = 0.5$



(a)



(b)

Figure 4. Accuracy and nDCG comparison at top-20 of 3-fold 300-query tests, with PMF, BPMF and Jaccard $\lambda = 0.5$

on other vectors, but still indeed cost our time and space on further computation tasks. To address this issue, we applied the software domain oriented database from Tian's work [23] to remove the non-software-related words. This reduced our vocabulary size significantly to 5,336. Also, within the totally 5,916,073 data source questions, we selected and kept only their top-5 voted answers of 11,832 questions for the training, which grantees the quality of our input data, increase our processing efficiency, as well as make it more feasible to conduct 3-fold tests using randomly chosen 100, 200, 300 and 400 queries.

B. Results Analysis and Evaluation

Since the number of clustering is variable and can affect the accuracy of recommendation, We use the Silhouettes Score is a measurement to evaluate the clustering quality. This score takes both the similarities between an element and a cluster and its dissimilarity to other clusters. Also, we tried different λ, the weight we use to combine our expertise score with the supplemental information learned from matrix factorization, with a 3-fold test.

Here, we calculated Silhouettes scores on both CLR clustering and the k-means clustering. Figure 2(a) shows the score at some clustering numbers and Figure 2(b) is an example of clustered words (topic). Because of the even distribution of our word vectors, we noticed that a smaller cluster number may have a relatively high score, yet we still need a reasonable number of clusters for grouping posts and doing the recommendation.

As the result shown in Figure 2(a), cluster number 243 by k-means clustering gives a relatively high Silhouettes score, at 0.028879744, and this also helps to produce better performance with tighter prediction, which hence becomes our final choice.

As for evaluating recommendation methods, both accuracy and quality are vital. We use the precision at N [1] for accuracy measurement and nDCG for quality assessment, which is computed based on an ordered list of the actual answers by the score as the evaluation metrics. Besides, we used the traditional Jaccard similarity to find potential experts on the same query sets as the baseline test. We further evaluate the stability of our approach by testing the approach with different test sizes of 100, 200, 300 and 400 queries. Figure 3 shows the accuracy and nDCG of those tests. Despite slight fluctuations with query sizes in both accuracy and nDCG, it achieves generally stable results in recommending the top-20 answerers.

In this paper, we compare our approach the popular methods of recommendation, namely Probabilistic Matrix Factorization(or PMF) and Bayesian PMF (or BPMF) [24], an extension version with Gaussian-Wishart priors placed on the user and item hyperparameters. Both methods enhance

the scalability of traditional matrix factorization and the performance on sparse data. Besides, we also compare with Jaccard similarity to measure the semantic correlations given a query post.

While comparing to the traditional approaches, our method performed much better(see Fiugure 4). Matrix Factorization methods appear not suitable for our very sparse data. Although there is the only slight improvement on nDCG when comparing to the semantic-based method(Jaccard-based approach), our accuracy is evidently better. For example, the $accuary@10$ of our approach is approximately 18%, which is about 2 times better than baseline tests.

IV. Related Works

Expert finding has been a long-standing topic of information management and has become hot when online communities became popular Largely due to the limit of computing performance, earlier works often focus on link analysis [25], [26] on users, while recent research is more diverse. For example, Chiang et al. [3] focus on graph-based question answering. They assume the reliability of user's browsing log and the authors claim that user generated contents are not so reliable as well as there is language dependence problem caused by some graph-based Q&A recommendation models. As users often browse other pages besides Q&A pages in forums, they apply the Continuous-time Markov model to generate QA Latent Browsing Graph and to overcome data sparsity; for the graph, they propose Latent Browsing Rank and Latent Browsing Rank Recommendation as the importance score and recommendation module.

Stack Overflow is a very popular object of study in recent years, as well as the expert recommendation problem on this website. The analysis of Hanrahan and his group sheds light on both the difficulty of questions and user expertise [5]. Their research relies on question-answer events to evaluate question difficulty, and then combine reputation stack Overflow provides. Zhang's z-score [4] differentiate up-votes and down-votes the users give to determine user expertise. Riahi et al. [6] focus on ranking users by building their profiles. They measure the relationships between users and questions and output the highly- ranked user based on interests. They also compared Latent Dirichlet Allocation (LDA) and Segmented Topic Model (STM) for clustering in their experiments.

Past work on answering provider recommendation is often only based on user performance, which in our perspective of view can be done better, as we can obtain relevance on textual as well and this is thanks to recent research on language processing. Dong's approach also classifies users according to topics and compare topics with questions for recommendation purposes [2]. However, it relies on user tags and uses user authority as the metrics for selecting users. User tags may not be so reliable as the tags themselves rely on user expertise. Guo et al. [1] systematically study existing solutions and develop their topic-based methods based on a new thought: either by investigating the answers to similar questions or by studying a user's performance based on his/her history. Despite similar in the assumption that questions and answers share same topic space, their user-centric model is fundamentally different from our answer-centric view.

V. Conclusion

We address in this paper the low participation issue in Stack Overflow. We have proposed a novel framework to recommend potential contributors when can answer new or unsolved questions best. Particularly, we use embedding techniques to generate word representations and to cluster those representations into topics. We then project new question onto a topic, compute its similarities to existing answers within the topic, and combine with post-user matrix learned via matrix factorization. The final output is a list of recommended contributors. We have tested different clustering methods and conducted experiments with three existing approaches for comparison. The results demonstrate the advantages of the proposed approach.

References

[1] J. Guo, S. Xu, S. Bao, and Y. Yu, "Tapping on the potential of q&a community by recommending answer providers," in *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, 2008, pp. 921–930.

[2] H. Dong, J. Wang, H. Lin, B. Xu, and Z. Yang, "Predicting best answerers for new questions: An approach leveraging distributed representations of words in community question answering," in *Frontier of Computer Science and Technology (FCST), 2015 Ninth International Conference on*. IEEE, 2015, pp. 13–18.

[3] M.-F. Chiang, W.-C. Peng, and S. Y. Philip, "Exploring latent browsing graph for question answering recommendation," *World Wide Web*, vol. 15, no. 5-6, pp. 603–630, 2012.

[4] J. Zhang, M. S. Ackerman, and L. Adamic, "Expertise networks in online communities: structure and algorithms," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 221–230.

[5] B. V. Hanrahan, G. Convertino, and L. Nelson, "Modeling problem difficulty and expertise in stackoverflow," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work Companion*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 91–94. [Online]. Available: http://doi.acm.org/10.1145/2141512.2141550

[6] F. Riahi, Z. Zolaktaf, M. Shafiei, and E. Milios, "Finding expert users in community question answering," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12 Companion. New York, NY, USA: ACM, 2012, pp. 791–798. [Online]. Available: http://doi.acm.org/10.1145/2187980.2188202

[7] D. Liang, J. Altosaar, L. Charlin, and D. M. Blei, "Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence," in *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 2016, pp. 59–66.

[8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[9] D. L. Lee, H. Chuang, and K. Seamons, "Document ranking and the vector-space model," *Software, IEEE*, vol. 14, no. 2, pp. 67–75, 1997.

[10] G. Begelman, P. Keller, F. Smadja *et al.*, "Automated tag clustering: Improving search and exploration in the tag space," in *Collaborative Web Tagging Workshop at WWW2006*, 2006.

[11] F. Beil, M. Ester, and X. Xu, "Frequent term-based text clustering," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 436–442.

[12] S. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive learning algorithms and representations for text categorization," in *Proceedings of the Seventh International Conference on Information and Knowledge Management*, ser. CIKM '98. New York, NY, USA: ACM, 1998, pp. 148–155. [Online]. Available: http://doi.acm.org/10.1145/288627.288651

[13] Y. Li, S. M. Chung, and J. D. Holt, "Text document clustering based on frequent word meaning sequences," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 381–404, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.datak.2007.08.001

[14] X. Hu, X. Zhang, C. Lu, E. K. Park, and X. Zhou, "Exploiting wikipedia as external knowledge for document clustering," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 389–396.

[15] A. Kalogeratos and A. Likas, "Text document clustering using global term context vectors," *Knowledge and information systems*, vol. 31, no. 3, pp. 455–474, 2012.

[16] R. Forsati, M. Mahdavi, M. Shamsfard, and M. R. Meybodi, "Efficient stochastic algorithms for document clustering," *Information Sciences*, vol. 220, pp. 269–291, 2013.

[17] F. Nie, X. Wang, M. I. Jordan, and H. Huang, "The constrained laplacian rank algorithm for graph-based clustering," 2016.

[18] J. Huang, F. Nie, and H. Huang, "A new simplex sparse learning model to measure data similarity for clustering," in *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 3569–3575.

[19] D. L. Donoho, "Neighborly polytopes and sparse solution of underdetermined linear equations," 2004.

[20] E. J. Candes, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on pure and applied mathematics*, vol. 59, no. 8, pp. 1207–1223, 2006.

[21] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000. [Online]. Available: http://science.sciencemag.org/content/290/5500/2323

[22] Y. Koren, R. Bell, C. Volinsky *et al.*, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[23] Y. Tian, D. Lo, and J. Lawall, "Sewordsim: Software-specific word similarity database," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 568–571.

[24] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization." in *Nips*, vol. 1, no. 1, 2007, pp. 2–1.

[25] P. Jurczyk and E. Agichtein, "Hits on question answer portals: exploration of link analysis for author ranking," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 845–846.

[26] J. Wang, Z. Chen, L. Tao, W.-Y. Ma, and L. Wenyin, "Ranking user's relevance to a topic through link analysis on web logs," in *Proceedings of the 4th international workshop on Web information and data management*. ACM, 2002, pp. 49–54.