Get Me The Best: Predicting Best Answerers in Community Question Answering Sites

Rohan Tondulkar
IIT Hyderabad
India
cs17mtech11028@iith.ac.in

Manisha Dubey
IIT Hyderabad
India
cs17resch11003@iith.ac.in

Maunendra Sankar Desarkar
IIT Hyderabad
India
maunendra@iith.ac.in

ABSTRACT

There has been a massive rise in the use of Community Question and Answering (CQA) forums to get solutions to various technical and non-technical queries. One common problem faced in CQA is the small number of experts, which leaves many questions unanswered. This paper addresses the challenging problem of predicting the best answerer for a new question and thereby recommending the best expert for the same. Although there are work in the literature that aim to find possible answerers for questions posted in CQA, very few algorithms exist for finding the best answerer whose answer will satisfy the information need of the original Poster. For finding answerers, existing approaches mostly use features based on content and tags associated with the questions. There are few approaches that additionally consider the users' history. In this paper, we propose an approach that considers a comprehensive set of features including but not limited to text representation, tag based similarity as well as multiple user-based features that target users' availability, agility as well as expertise for predicting the best answerer for a given question. We also include features that give incentives to users who answer less but more important questions over those who answer a lot of questions of less importance. A learning to rank algorithm is used to find the weight of each feature. Experiments conducted on a real dataset from Stack Exchange show the efficacy of the proposed method in terms of multiple evaluation metrics for accuracy, robustness and real time performance.

CCS CONCEPTS

• **Information systems → Expert search**; *Document topic models*; *Content analysis and feature selection*; *Personalization*; *Similarity measures*; *Learning to rank*; *Top-k retrieval in databases*; *Question answering*; *Recommender systems*; Language models; Retrieval efficiency;

KEYWORDS

Expert Recommendation;Community Question Answering;Learning to Rank

RecSys '18, October 2–7, 2018, Vancouver, BC, Canada
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5901-6/18/10…$15.00
https://doi.org/10.1145/3240323.3240346

ACM Reference Format:
Rohan Tondulkar, Manisha Dubey, and Maunendra Sankar Desarkar. 2018. Get Me The Best: Predicting Best Answerers in Community Question Answering Sites. In *Twelfth ACM Conference on Recommender Systems (RecSys '18), October 2–7, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240323.3240346

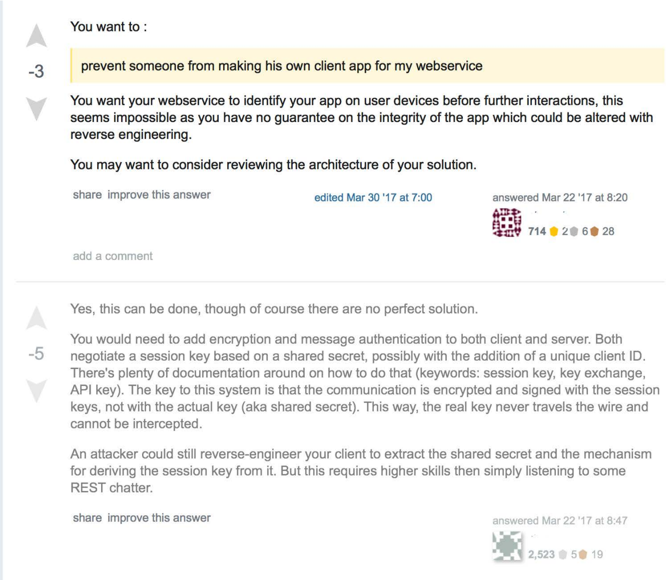
1 INTRODUCTION

In recent years, the use of Community Question Answering (CQA) forums has seen a significant rise. One of the reasons for this rise is the variety of such forums (eg. Stack Exchange, Quora, Yahoo Answers, etc) available to users now. These websites provide the functionality to users to ask questions and give answers to the questions based on their expertise. Some of these forums also allow users some additional features like giving upvotes/downvotes to questions and answers based on their perceived correctness. It also allows question posters to mark one of the answers as the accepted answer. On the Stack Exchange forums, more than 10 million users have registered and around 17 million questions have been asked so far. Also daily more than 11 million pages of these forums are visited. These statistics clearly convey the popularity and importance of CQA forums.

In spite of their recent success, lot of questions remain unanswered in CQA websites. A stat[1] from the from top 5 most active Stack Exchange forums shows that around 30% of the questions in those forums remain unanswered. This problem arises mainly due to two reasons - (a) question posters do not know who the best experts are for their questions and (b) experts are not directed to the questions that match their expertise. The problem is amplified by the fact that approximately only 10% of the users ever answer a question and very few of them are experts. Less than 1% of total users have posted more than one accepted answer. The clear lack of experts raises the demand for an efficient system to optimize the use of existing experts.

One way to tackle this problem is to recommend experts for a given question who can answer that particular question. There are two common approaches in which this problem is addressed in literature. Given a question, the first approach tries to find questions similar to it and recommends users who gave *good* answers on those questions. The second approach profiles the users based on their previous answers and recommends users whose profiles are best suited for the given question. Most of the work following this second approach has been done on the easier problem of predicting any answerer for a given question. These predictions can then be used to recommend experts for a new question. It has been observed in Stack Exchange that not all answers are good answers and some answers get negative scores as shown in Figure 1. Thus, predicting

[1] https://stackexchange.com/sites?view=list



-3

You want to :

prevent someone from making his own client app for my webservice

You want your webservice to identify your app on user devices before further interactions, this seems impossible as you have no guarantee on the integrity of the app which could be altered with reverse engineering.

You may want to consider reviewing the architecture of your solution.

share improve this answer edited Mar 30 '17 at 7:00 answered Mar 22 '17 at 8:20

714 ● 2 ● 6 ● 28

add a comment

-5

Yes, this can be done, though of course there are no perfect solution.

You would need to add encryption and message authentication to both client and server. Both negotiate a session key based on a shared secret, possibly with the addition of a unique client ID. There's plenty of documentation around on how to do that (keywords: session key, key exchange, API key). The key to this system is that the communication is encrypted and signed with the session keys, not with the actual key (aka shared secret). This way, the real key never travels the wire and cannot be intercepted.

An attacker could still reverse-engineer your client to extract the shared secret and the mechanism for deriving the session key from it. But this requires higher skills then simply listening to some REST chatter.

share improve this answer answered Mar 22 '17 at 8:47

2,523 ● 5 ● 19

Figure 1: An example with two answers getting negative votes. This shows that predicting any answerer is not enough as there may be some *bad* answers for a question.

any answerer isn't enough. Less work has been done on the more difficult problem of predicting the best answerer for a question and recommending better experts for a question. The best answerer is the user whose answer is marked as the accepted answer by the question poster. This is decided by the poster based on whether the answer exactly matches his requirement in the question. This makes modeling this problem a very challenging task.

In this paper we propose an effective approach for predicting the best answerer for a given question. This is done by ranking the *expected answerers* for a question with the aim of ranking *possible best answerers* higher. Most of the existing work focuses on using text and tags of questions previously answered by users and very less on using features based on users' history. We consider a comprehensive set of features which includes user-based features denoting the expertise, agility, availability of user along with several tag and text based features. We also include features that incentivize users who answer less but more important questions over those who answer more but less important questions. A learning to rank algorithm, LambdaMART, is used to learn the weights for our feature set. We also show our model can be used real time to recommend experts on new questions.

Contributions. Our contributions can be summarized as follows:

- We use a diverse set of features that considers question related details as well as variety of details related to users' history.
- Every feature contains specific importance and contributes positively to the performance.
- We combine the features based on certain weights which are derived using learning to rank algorithm.
- We show improvement in results in this domain of finding best answerer for a given question.
- We also build a real time prediction model to predict the best answerer for any new question.

2 LITERATURE REVIEW

In this section, we review the work done previously in the domain of Community Question Answering. We segregate the work in this domain into a few categories namely (a) Finding quality questions and answers, and (b) Expert recommendation.

2.1 Finding Quality Questions and Answers

In the past few years, many community question answering platforms like Stack Overflow[1], Quora[2] have emerged as large knowledge bases including questions, their answers and other information associated with them along with the user details. Due to large popularity of such platforms, these systems have been subject to various research problems. One such research deals with the problem of finding quality answers while minimizing the waiting time for the question poster. One way to address such a problem is by finding answers for the existing similar questions. To model answer-finding system, [1] has used five different ways: TF-IDF, adaptive TF-IDF, query-expansion, statistical translation and latent-variable models. In other work [7], authors have used 13 different non-textual features like users' dis-recommendations, click-counts to predict quality of answers. They have used kernel density estimation and maximum entropy approach to handle various types of non-textual features.

Some work has been done in the area of similar question recommendation as well. In [9], authors have tried to find semantically similar questions to estimate probabilities for a translation-based retrieval model. In the work proposed in [4], the authors have worked on constrained question recommendation task under the constraints of load balancing and expertise matching. They have designed a context-aware matrix factorization including contextual information, then built a max cost flow model to fit the constraints. Heterogeneous Network Integration Learning (HNIL) is used by [17] where they have modeled the question textual content using RNN. Along with this, they have used user embeddings to represent question and rank similarities with previous questions. In [16], proposed model incorporates word-to-word translation probabilities learned through exploiting different sources of information for finding a good answer to user's question. A new dependency based model (DM) for the question retrieval in community based question answering is proposed by [10]. In another work done in [13], authors have tried to find useful answers which they call as promising answers.

The above line of work tries to address the problems of finding best answers or similar questions given an input question. However, in this paper, we are looking at the problem of finding the best answerers for a given question. Although these two problem statements are quite different, many of the approaches mentioned above need to model the questions efficiently. We found the representation of the questions, and their corresponding insights to be quite useful in some cases, and hence we discuss about this line of research in some detail.

[1] https://stackoverflow.com/
[2] https://www.quora.com/

2.2 Expert Recommendation

Several researchers have worked on the problem of finding the best experts for a given question, However, the term *expert* has been defined in various ways in different papers. In [2], authors have considered any answerer to be the expert for the question asked. They have modeled user-tag expertise matrix using probabilistic matrix factorization. Some basic models like query likelihood model, relevance model and cluster-based language model to find experts for a given question have been used by [15]. In the work proposed in [15], authors have used topic sensitive page ranking algorithm where they have performed topic distillation by creating user-topic model. They have also considered link structure and topical similarity between users for expert recommendation.

Few approaches have considered answerer with maximum number of votes to be accepted answer, and the corresponding user has been considered as the expert for the question asked. A probabilistic generative model with GMM hybrid to jointly model topics and expertise by integrating textual content model and link structure analysis has been proposed by [11]. Less work has been done considering accepted answerer to be best answerer. One such work is presented in [5], where word-based models and topic-based models have been explored for expert recommendation. The techniques used in this paper are TF-IDF and language modeling which use smoothed distribution to estimate likelihood of query. Topic based models like segmented topic modeling and latent dirichlet allocation provide additional representation level. Probabilistic latent semantic indexing is used to model users' interest from his previously asked questions in [12]. Also, they have introduced a novel metric for evaluation of such recommendation systems. In [6], authors have used word2vec for textual representation. Also, they have used user authority and activity as prior probability for user.

3 PROBLEM DEFINITION

Let q be a new question and U be the set of users in the forum. We aim to find the user $u_i \in U$ who is the best answerer for question q.

We tackle the above problem as a ranking problem. Given a question q and the set of users U, we want to output a partial ranking R of k usres ($|R| = k < |U|$). If rank of u_i < rank of u_j in R, it indicates that the algorithm is more confident about u_i being the best answerer for the question over u_j. We want to learn a scoring function $\sigma(u|q)$ that denotes how likely it is for a user u to give best answer for the question q. If $\sigma(u_i|q) > \sigma(u_j|q)$, then u_i is placed before u_j in the output ranking R.

4 METHODOLOGY

We now explain our methodology in detail. To model our recommendation system, we have used a wide set of features which are then fed to the learning to rank (LTR) framework. We explain each of them in detail in the following sections.

4.1 Feature Set

We have considered a comprehensive set of features to include various aspects of answering a question. The features used in our work can be classified into four main categories:

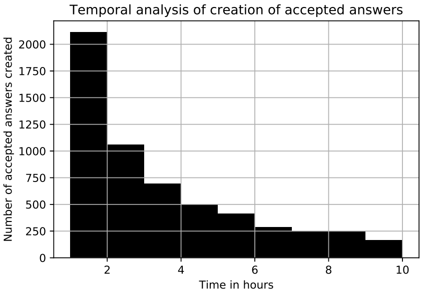


Figure 2: Plot for the number of accepted answers created against time in hours, x-axis represents hours passed since the question was posted

- **Content** In order to leverage the information contained in the questions and answers posted for those questions, we have used two features, namely, **tag-based similarity** (TagSimilarity) and **textual similarity** (TextSimilarity). These features measure semantic relation between a user and question.
  - **TagSimilarity:** For finding tag-based similarity, tags used on the questions answered by user were considered. For each user, number of questions a user has answered with a particular tag was considered. Following similarity measure was used for tag-based similarity between a question and a user:

$$TagSimilarity = (Number\,of\,matched\,tags\,with\,count \geq 1)\cdot \\ (Sum\,of\,counts\,of\,matched\,tags\,) \quad (1)$$

  - **TextSimilarity:** We use Doc2Vec [14] ,which is an unsupervised algorithm, to learn fixed-length vector representations for questions. For vector representation of each user, we have considered the added textual content of each question answered by him. Cosine similarity is calculated on these representations to model similarity between a user and a question.
- **User's Agility** To take advantage of the temporal relations in acceptance of an answer we take user activity into consideration. Importance of time can be emphasized from Figure 2 which shows that maximum number of accepted answers are created within 2 hours of posting of question. In fact that 70% of accepted answers are written within 5 hours of their creation. To include user agility we have considered two features: **Days to previous answer** (DaysToPreviousAns) and **answering frequency in days** (AnsweringFreq).
  - **DaysToPreviousAns:** Days to previous answer represents number of days since a user had last posted an answer with respect to the posting time of the particular question.
  - **AnsweringFreq:** Answering frequency is the average number of days taken by user to answer a question.
- **User's expertise** To quantify expertise of an user, StackExchange assigns a reputation score to the users. This score is calculated based on the votes received on the questions and answers posted by the user.We use this feature in our method and call refer to it as **RepScore**.
- **Removal of bias** In crowd generated platforms like Stack Overflow, a relatively small subset of users are responsible

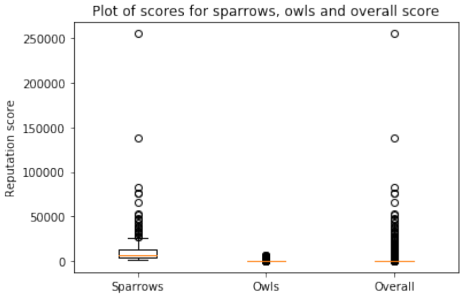


Figure 3: Plot representing reputation score of sparrows and owls where sparrows are considered to be those answerers who have answered more than 10 questions

for majority of contributions. The users who are highly active on such platforms are called as *sparrows*. Also, there is a category of users called *owls* who, despite being expert, answer to only those questions which are perceived as difficult and important to the community [8]. However, due to built-in incentivization systems, standard expert identification systems often misclassify the active users. This anomaly in such systems can be observed through Figure 3 which shows huge difference in reputation score for sparrows and owls. Here, we have considered all those users who have posted less than any 10 answers (need not to be best answer) as owls and more than that have been considered as sparrows. The threshold value of 10 is chosen randomly. The plot shows that difference in reputation score for sparrows and owls. On the other hand, behavior of owls is similar to average users.

In order to characterize the users' contribution in terms of quality of answer, we use features like **average question quality** (AvgQuesQuality), **average question popularity** (AvgQuesPop), **smoothened number of best answers** (ModifiedBestAns) and **mean reciprocal rank** (MRR).

- **AvgQuesQuality:** Average question quality for a user is the average score of all questions which are answered by that user.
- **AvgQuesPop:** Average question popularity for a user is calculated as the average of view counts of all the questions which are answered by that user.
- **MRR:** Mean reciprocal rank of any user is the average of reciprocal of the rank for all questions answered by that user. The rank for a question is based on the votes received for the answers on that question.
- **ModifiedBestAns:** Since owls are characterized by the fact that they provide less answers but only good ones, we have smoothened number of best answers using the following formula:

$$ModifiedBestAns = \frac{Number\ of\ best\ answers\ by\ the\ expert}{Total\ number\ of\ answers\ by\ the\ expert + \lambda} \tag{2}$$

where λ is a constant whose value can be decided empirically.

Table 1: Table consisting of all the features along with their interpretation

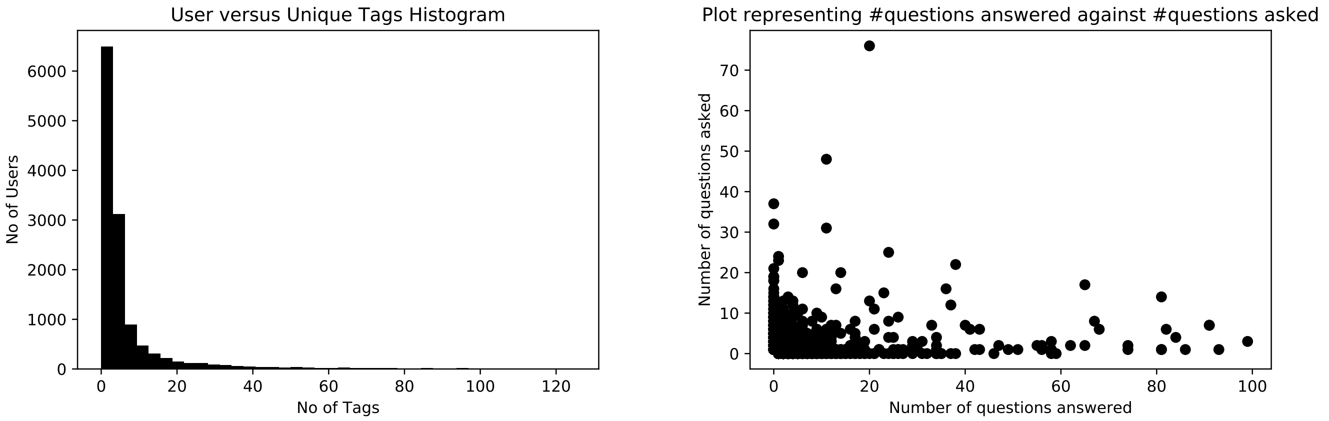
Features	Description
TagSimilarity	Similarity between tags of question and user profile
TextSimilarity	Similarity between Doc2Vec representation of question and user profile
DaysToPrevAns	Number of days since a user had last posted an answer before the time of current question
AnsweringFreq	Average of number of days taken by user to answer a question
RepScore	Reputation assigned to every user by Stack Exchange
AvgQuesQuality	Average score of all question answered by an user
AvgQuesPop	Average view count of the questions answered by an user
ModifiedBestAns	Smoothed number of best answers
MRR	Average of vote based reciprocal rank for all questions answered by an user

4.2 Learning to Rank (LTR)

Learning to Rank is a machine learning based framework to come up with a desired scoring function for ranking problems. We have employed supervised pairwise learning-to-rank in our work. Our problem statement for learning to rank can be stated as follows:

> Let $U = \{U_1, U_2, ...U_{|U|}\}$ be the set of users and $Q = \{Q_1, Q_2, ...Q_{|Q|}\}$ be the set of questions. The data is made up of $|Q||U|$ query-user pairs which can be written as $(\{(q_i, u_j)\})_{j=1,2,\cdots|U|}^{i=1,2..|Q|}$. Each pair (q_i, u_j) is represented by feature vector $x_{i,j} \in \mathbb{R}^n$ such that $x_{i,j} = (x^{(1)}_{i,j}, x^{(2)}_{i,j}, x^{(3)}_{i,j}, ..., x^{(n)}_{i,j})$ where n is the number of features used by ranking function, $x^{(t)}_{i,j}$ denotes the value of t-th feature. We want to learn a scoring function $\sigma(x_{i,j})$ such that for a question q_i, the function assigns higher values to possible best answerers and lower values to the other users.

For implementation, we have used the LambdaMART algorithm [3]. LambdaMART combines LambdaRank and MART (Multiple Additive Regression Trees) approaches. In this technique, ranking is transformed into pairwise regression or classification problem. A preference network is developed to evaluate the pairwise order between two documents. The network learns the preference function to find rank of new data. While MART uses gradient boosted decision trees for prediction tasks, LambdaMART uses gradient boosted decision trees using a cost function derived from LambdaRank for solving a ranking task. LambdaRank takes input as all possible combinations of (Questions, Users) pairs along with their respective set of features. This algorithm outputs score for each pair in test dataset which is used to provide ranking to users. In this way our approach can be used to find candidate users for best answerer prediction for a given test question.



(a) Analysis of number of unique tags answered by users

(b) Analysis of relation between number of questions answered and number of questions asked

Figure 4: Analyzing user behavior on the Information Security forum of Stack Exchange

5 EXPERIMENTS

5.1 Stack Exchange

We perform our experiments on the Stack Exchange community question answering dataset. It currently has 172 different forums based on technical as well as non-technical domains with over 100 million monthly unique visitors. Users can ask questions and associate up to 5 unique tags with each question. Other users who are interested in those topics can give answers to these questions. Also, questions and answers can be voted up/down by other users in the community. The question poster also gets to select one answer as the accepted answer. Each user gets a reputation score which is calculated based on the votes on their questions and answers.

5.2 Dataset

The data dump of all communities under Stack Exchange is available for research on the Internet Archive[3] under the Creative Commons License. It contains the anonymized dump of all user contributed content since 2009 and is updated every 3 months. We consider the *Information Security* forum which is a relatively new community to analyze the modern real data. We use the entire data available up to August 2017. It contains a total of 119,094 users, 41,706 questions posted and 1.1K unique tags which constitute the sub-topics in this community. For each question, the best answer is defined as the one that is selected as the accepted answer by the question poster.

We prepare the dataset with the following considerations: (1) Select questions where the poster has selected an accepted answer. (2) Select answerers who have answered at least two best answers. This ensures that we have at least one question of each user in the training set for building the user profiles. We have a total of 1339 such users. (3) Consider all the posts where the above selected users have answered including those where they are not marked as best answerer. We have a total of 37,522 questions in our dataset. (4) Only the questions where selected users are best answerers are used in training and predictions using LTR. (5) Tags on all the questions used in our dataset are considered which equals to 987 unique tags.

We see that, in most of the existing work, very stringent criteria are put in place for filtering questions and users on the lines of only selecting questions with more than 10 answers on them or selecting users who have answered at least 50 questions. In comparison, our criteria are very generic as we have only one filtering criteria to select users who have given at least 2 best answers. This is taken only as a requirement to ensure that at least one post of each best answerer is available in the training set while creating user profiles.

Analysis of the Dataset and Insights: We analyze our filtered dataset to see the pattern between number of questions asked and number of questions answered by these users. From Figure 4b, we can see that users who give high number of answers indeed ask lesser questions, proving their status as experts. We find that 63% of the times the accepted answer matches the answer which gets highest number of up votes from the community. The problem of lack of expertise is highlighted by the fact that only 9.8% of the users have ever answered a question and 63% of the answerers have given answer to only one question. We also observe that most answerers are experts in very niche sub-topics. This can be explained using Figure 4a which shows that most answerers give answers on very few unique tags.

The exact datasets of previous work on Stack Exchange forums are not made available which makes it difficult to reproduce the results. We have made our train and test dataset available with most of the features extracted on GitHub[4] for further work.

5.3 Experimental Details

Now we provide the experimental details so that it becomes easier to reproduce our results. After the generation of dataset following the steps mentioned in Section 5.2, it is split into training and testing datasets. The training involves two tasks: generating user profiles for all users and learning the weights of the features using LTR. Only those questions where the selected users are best answerers are selected for LTR training and testing. This is done because the learning of weights is performed in a way to discriminate the best answerers from other users. For LambdaMART we need the

[3] https://archive.org/details/stackexchange

[4] https://github.com/Rohan191/stack-exchange-expert-prediction

supervised information of which user is the best answerer for a given question. We maintain a ratio of 75:25 while splitting these questions into training and testing sets. The questions selected for LTR training as well as the ones where the selected users are not best answerers are used for generating the user profiles. We can include the extra questions (where the best answerer information is not available) for this purpose because the user features can be constructed without any supervised information.

The LTR expects as input the feature set for all question-user pairs along with their relevance labels during training. Relevance label is 1 if the user is best answerer and 0 otherwise. Since we have 12785 questions and 1339 users, the input to LTR becomes very huge. To reduce time of LTR runs, we sampled a candidate set of 150 users per question based on tag-similarity score of users for every question. This covers 89% of best answerers and the missing ones are added to complete the candidate training set. The number 150 was considered as the ideal value after experiments showed it optimized time and had minimum impact on prediction results of LTR. For testing, all the question-user pairs for 4945 questions and 1339 users were considered as any of the users can be the best answerer. All the features are then extracted based on their type. Some of the features like TagSimilarity, TextSimilarity and DaysToPrevAns are dependent on questions and have to be calculated separately for each question-user pair in the candidate training set and test set. Other features which are the user-related features like RepScore, ModifiedBestAns, MRR, AvgQuestionQuality and AvgQuesPop are calculated independently of the train or test questions and remain same for both of them. The value of lambda for ModifiedBestAns was set to 10 based on the performance on different values of lambda. We found that higher values of lambda had negative impact on the performance. The feature set for the candidate training set along with the relevance values is passed to LambdaMART[5] Learning to Rank algorithm for training. We perform hyper-parameter tuning to get best performance. Our ideal parameters for LTR training are given in Table 2. The feature set for test data is passed for prediction to get scores on each question-user pair. The users are ranked based on these scores for every question and then performance is evaluated.

Table 2: Parameter values for LambdaMART training

Parameter	Value
Metric	Expected Reciprocal Rank (ERR)
k	10 (for calculating metric)
Learning rate	0.02
n_estimators	1000
max_features	0.5
max_leaf_nodes	10

5.4 Methods Compared

We compare the performance of our proposed method with that of few other methods mentioned below. Some of these methods are taken from literature. We implemented these methods based on the information provided in the corresponding papers. The remaining ones can be considered as variants of our proposed method, where we experiment with different feature subsets from our original set of features.

Methods from Literature:

- **TF-IDF:** This uses the TF-IDF model presented in [5].
- **LDA:** This uses the Latent Dirichlet Allocation (LDA) model proposed in [5].
- **Word2Vec:** This is the approach proposed in [6] which includes word2vec model with priors on user activity and expertise.
- **Tag PNMF:** This is the tag based approach using Probabilistic Non-negative Matrix Factorization (PNMF) mentioned in [2]. We have used the nimfa[6] python library to apply PNMF.

Feature subsets and variants of the proposed method:

- **Features Sum:** Sum of all features is used as a score to rank users instead of LTR. This is done to check whether learning the weights for each feature is needed.
- **Without Tag Model:** This method considers all features except TagSimilarity. LTR is used to learn the feature weights.
- **Without Text Model:** This model uses all features except TextSimilarity feature. LTR is used to learn the feature weights.
- **Only Tag Model:** We see how well the TagSimilarity feature performs individually when used as the score to rank users.

5.5 Evaluation Metrics

We use the following metrics to compare the performances of the algorithms.

Accuracy at N (Acc@N). Accuracy@N is defined as the percentage of test questions for which best answerers are predicted when top N ranked users are selected.

$$Acc@N = \frac{\sum_{i=1}^{T} \sum_{j=1}^{N} \mathbb{I}(U_{i,j} == B_i)}{T}$$

where T is the number of questions in test dataset, N is the number of top ranked users selected, $U_{i,j}$ is the user at rank j for question i in the test dataset, B_i is the user marked as best answerer for question i in the test dataset, and $\mathbb{I}(p)$ is an indicator function which evaluates to 1 if the predicte p is true, and 0 otherwise. We calculate the values of Accuracy@N for $N = 1, 5, 10, 20, 50$.

Mean Reciprocal Rank at N (MRR@N). If we select top N ranked users for every question then the reciprocal of rank at which the best answerer is placed is the reciprocal rank for that question. The reciprocal rank is 0 if the best answerer is not in top N ranked users. The MRR@N is the average reciprocal rank over all questions in the test dataset.

$$MRR@N = \frac{\sum_{i=1}^{T} \sum_{j=1}^{N} \frac{1}{j} \mathbb{I}(U_{i,j} == B_i)}{T}$$

where all the variables are same as mentioned above. We calculate the values of MRR@N for $N = 1, 5, 10, 20, 50$.

For both these measures, higher values indicate better performances. Since each test question has exactly one best answerer, Precision score would have been same as Accuracy. Also, Since the relevance labels are binary (best answerer or not), MRR will be

[5] https://github.com/jma127/pyltr

[6] http://nimfa.biolab.si

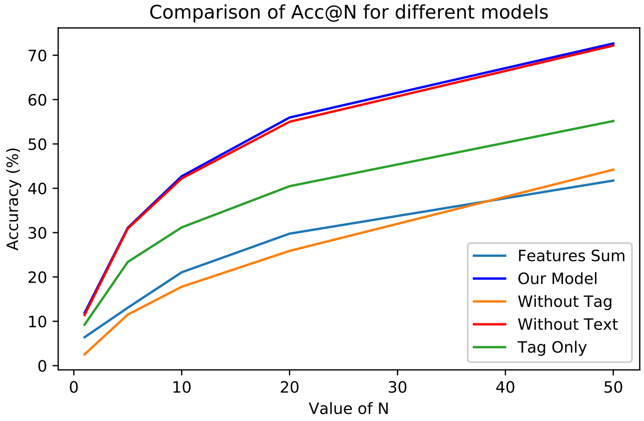


Figure 5: Plot of Acc@N for models based on different feature combinations using our approach

correlated with NDCG. Hence we do not use Precision of NDCG separately as evaluation metrics.

5.6 Results and Discussion

In Table 3 we show *Acc@N* results for our proposed model and other models mentioned in Section 5.4. We can clearly see that our model outperforms the models from existing work. The basic model in our approach, *Features Sum* model, performs better than the models from literature. We also observe that all the text based models give very poor performance for this problem on our dataset. The tag-based approach [2] performs comparatively better than the text based approaches. This shows models dominated by text-based features [5, 6] show poor performance for this problem.

Table 3: Acc@N comparisons between our benchmark model and models proposed in literature. Best results are indicated in bold

	Accuracy				
Model	**@1**	**@5**	**@10**	**@20**	**@50**
Our Model	**11.910**	**31.080**	**42.710**	**55.960**	**72.660**
TF-IDF [5]	0.040	0.404	0.788	1.638	3.940
LDA [5]	0.120	0.520	0.860	1.710	3.860
Word2Vec [6]	0.200	0.670	1.110	1.840	3.100
Tag-based [2]	2.930	9.140	11.950	15.400	22.120
Features Sum	6.410	13.060	21.050	29.770	41.740
Without Tag	2.550	11.510	17.790	25.880	44.200
Without Text	11.440	30.920	42.220	55.005	72.194
Tag only	9.240	23.380	31.180	40.470	55.170

The *MRR@N* results of all the models mentioned in Section 5.4 are shown in Table 4. Also the plot of *Acc@N* for different variants of our models is shown in Figure 5. We can see that all our models perform better than the models in literature. The Features Sum model, which takes the sum of all features gives poorer results results compared to our final model which learns the weights using LTR. This shows that learning the weights contributes immensely to the model performance. The final proposed model is clearly the best performing model which gives a value of 0.204 for *MRR@10*. When we remove the TagSimilarity feature in Without Tag model, we see a big drop in performance. We remove TextSimilarity for the *Without Text* model and it does not impact the performance heavily. The *MRR@10* for Tag Only Model which considers TagSimilarity as the only feature is 0.155. We can infer that TagSimilarity is a very important feature in our model and TextSimilarity has less impact on the overall performance.

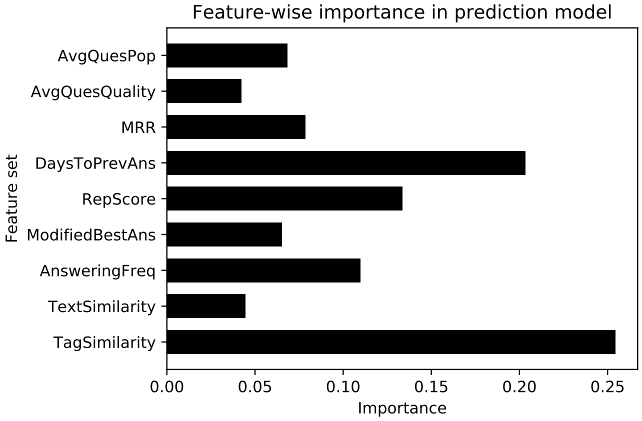


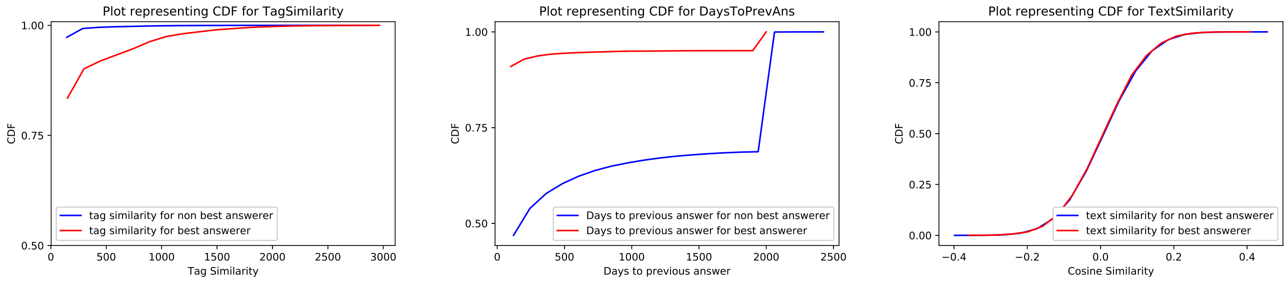
Figure 6: Capture the importance of each feature towards prediction

Table 4: MRR@N comparisons on models using different feature combinations to understand their importance

	Mean Reciprocal Rank (MRR)				
Model	**@1**	**@5**	**@10**	**@20**	**@50**
Our Model	**0.119**	**0.188**	**0.204**	**0.213**	**0.218**
TF-IDF [5]	0.000	0.002	0.002	0.002	0.003
LDA [5]	0.001	0.002	0.003	0.003	0.004
Word2Vec [6]	0.002	0.004	0.004	0.005	0.005
Tag-based [2]	0.029	0.052	0.056	0.058	0.060
Features Sum	0.064	0.092	0.102	0.109	0.113
Without Tag	0.026	0.055	0.064	0.070	0.075
Without Text	0.115	0.183	0.199	0.207	0.213
Tag only	0.092	0.145	0.155	0.161	0.166

These observations are also validated by Figure 6 which shows the importance of each feature of our model as given after LTR training. Along with TagSimilarity, DaysToPreviousAns which tells the availability of user, makes important contribution to the performance of the model. The RepScore and AnsweringFreq contribute next highest. The rest of the features which are used to remove the bias also make contributions which are higher than TextSimilarity.

In an attempt to provide an explanation for the performance of some features, we plot Cumulative Distribution Function (CDF) for 3 features as shown in Figure 7 below. These plots calculate CDF based on the probability of the feature values in each bin for best answerers and non-best answerers for every question in training



(a) CDF plot of TagSimilarity

(b) CDF plot of DaysToPrevAns

(c) CDF plot of TextSimilarity

Figure 7: CDF plots on various features for best answerers and non-best answerers to explain their performance

dataset. We can see a clear distinction in CDF of TagSimilarity and DaysToPreviousAns compared to that of TextSimilarity. For TagSimilarity, it tells us that more percentage of best answerers have high TagSimilarity score compared to non-best answerers as shown in Fig 7a. non-best answerers have high percentage of users with low scores of tag-based feature. Similarly, for DaysToPrevAns, we can see in Figure 7b that higher number of best answerers have low values of this feature and it's vice versa for non-best answerers. For TextSimilarity, we observe from Figure 7c that the CDFs for best answerer and non-best answerers almost completely overlap. Thus, it explains the feature importance we mentioned above that TagSimilarity and DaysToPrevAns are amongst the most important features and Text Similarity adds less value to the performance.

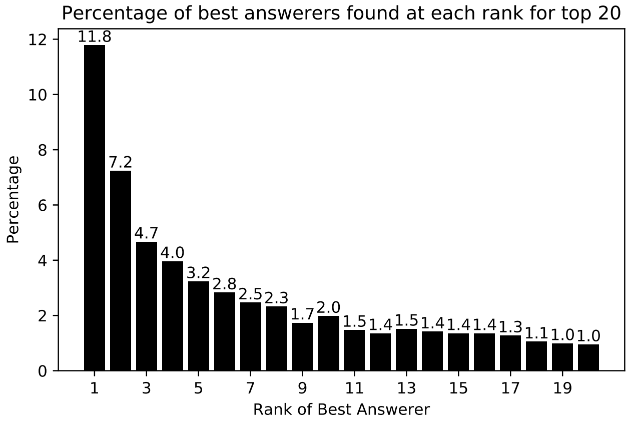


Figure 8: Percentage of best answerers predicted at each rank

The percentage of best answerers predicted at each rank is shown in Figure 8 when top 20 users are predicted per question. We can see that most best answerers are found in top 5 ranks. This also corresponds with our *MRR@5* value being close to 0.2. It shows that our model does a good job in recommending best answerers at higher ranks.

Real time prediction. We also build a real time model for recommendation of experts on Stack Exchange. Our model takes an average of 0.45 seconds to give ranked list of users for new questions on which no features are already calculated. The real time input of new questions should consist of the following details - question content, tags associated with the question and time of posting the question. Based on only these three details, the model calculates all the features for this question and returns a ranked list of top K experts. Thus, we show that our model works efficiently and can be used for real time predictions on new questions. We make this model publicly available on GitHub [7]. This experiment was performed on Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz processor with 14 cores with 64 GB RAM available.

6 CONCLUSION

In this paper we sought to build an effective model to solve the problem of predicting best answerers for Community Question Answering forums. We do that by using a extensive set of features that uses information based on tags, text content, user expertise, user agility, user availability, etc. We learn the weights for these features using Learning to Rank algorithm. We show that our model performs better than various existing works in this domain. We also prove that text based features perform badly on this problem on our dataset. We also demonstrate that tag-based similarity and user availability perform better than other features. We show efficiency of our model by its real time prediction which recommends the best answerers for a new question in 0.45 seconds.

7 FUTURE WORK

In the future we would like to apply this approach on bigger datasets like the main Stack Overflow forum. An interesting area of research would be to see if the learnings from Information Security dataset can be transfered on different datasets, especially the non-technical CQA forums. We also plan to exploit the features based on the history of the question poster. This can help us in extracting information on whether a question poster is biased towards a few experts while selecting their answer as accepted answer. We also plan to explore the impact of location based proximity on responding to answers faster.

REFERENCES

[1] Berger A., Caruana R., Cohn D.and Freitag D., and Mittal V. 2000. Bridging the lexical chasm: statistical approaches to answer-finding. In *23rd annual international ACM SIGIR conference on Research and development in information retrieval.* ACM.

[2] Yang B. and Manandhar S. 2014. Tag-based expert recommendation in community question answering. In *Advances in Social Networks Analysis and Mining (ASONAM).* IEEE/ACM.

[3] Burges C.J. 2010. From ranknet to lambdarank to lambdamart: An overview. Learning. ACM, 475–482.

[7] https://github.com/Rohan191/stack-exchange-expert-prediction

[4] Yang D., Adamson D., and Rose C.P. 2014. Question Recommendation with Constraints for Massive Open Online Courses. In *8th ACM Conference on Recommender systems*. ACM.

[5] Riahi F., Zolaktaf Z., Shafiei M., and Milios E. 2012. Finding expert users in community question answering. In *21st International Conference on World Wide Web*. ACM.

[6] Dong H., Wang J., Lin H., Xu B., and Yang Z. 2005. 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)*. IEEE.

[7] Jeon J., Croft W.B., Lee J.H., and Park S. 2006. A Framework to Predict the Quality of Answers with NonTextual Features. In *29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM.

[8] Yang J., Tao K., Bozzon A., and Houben G.J. 2014. Sparrows and owls: Characterisation of expert behaviour in stackoverflow. In *International Conference on User Modeling, Adaptation, and Personalization*. Springer, Cham., 266–277.

[9] J. Jeo, Croft W.B., and Lee. 2005. Finding Similar Questions in Large Question and Answer Archives. In *14th ACM international conference on Information and knowledge management*. ACM.

[10] Bae K. and Ko Y. 2017. Improving Question Retrieval in cQA Services Using a Dependency Parser. In *IEICE TRANSACTIONS on Information and Systems*. 807–810.

[11] Yang L., Qiu M., Gottipati S., Zhu F., Jiang J., Sun H., and Chen Z. 2013. Cqarank: jointly model topics and expertise in community question answering. In *Proceedings of the 22nd ACM international conference on Information and Knowledge Management*. ACM, 99–108.

[12] Qu M., Qiu G., He X., Zhang C., Wu H., Bu J., and Chen. 2009. Probabilistic Question Recommendation for Question Answering Communities. In *18th international conference on World wide web*. ACM.

[13] Roy P.K., Ahmad Z., J.P. Alryalat Singh, M.A.A. Rana, N.P., and Dwivedi Y.K. 2018. Finding and Ranking High-Quality Answers in Community Question Answering Sites. In *Global Journal of Flexible Systems Management*. 53–68.

[14] Le Q. and Mikolov T. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*. 1188–1196.

[15] Liu X., Croft W.B., and Koll M. 2005. Finding Experts in Community-Based Question-Answering Services. In *14th ACM international conference on Information and knowledge management*. ACM.

[16] Xue X., Jeon J., and Croft W.B. 2008. Retrieval models for question and answer archives. In *31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 475–482.

[17] Chen Z.and Zhang C.and Zhao Z. and Cai D. 2016. Question Retrieval for Community-based Question Answering via Heterogeneous Network Integration Learning. arXiv:1611.08135