The Journal of Systems and Software 208 (2024) 111884

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A study on classifying Stack Overflow questions based on difficulty by utilizing contextual features[☆]

Maliha Noushin Raida, Zannatun Naim Sristy [*], Nawshin Ulfat, Sheikh Moonwara Anjum Monisha, Md. Jubair Ibna Mostafa, Md. Nazmul Haque

Software Engineering Lab (SELab), Islamic University of Technology (IUT), Gazipur, 1704, Bangladesh
Department of Computer Science and Engineering, University of Technology (IUT), Gazipur, 1704, Bangladesh

ARTICLE INFO

Keywords:
Stack Overflow
Question difficulty
Text analysis

ABSTRACT

Technical question-answering sites like Stack Overflow are gaining enormous attention from practitioners of specialized fields looking to exchange their programming knowledge. They ask questions on different topics with varying degrees of complexity and difficulty. All practitioners do not have the same level of expertise on those topics to respond to such questions. However, the current approach used by Stack Overflow mostly filters questions based on topics alone and does not take difficulty into account. For this reason, a large percentage of questions fail to attract the attention of appropriate users, resulting in questions having no answer or a significant delay in response time. To address these limitations, we incorporate three models, TF-IDF, LDA, and Doc2Vec, to extract semantic and context-dependent features that can measure the difficulty of questions. Each of these models is paired with different classifiers along with other features to classify the questions based on difficulty. Extensive experiments on three different datasets exhibit the effectiveness of our models, and Doc2Vec outperforms the other models. We also identified that the contextual features are correlated with question difficulty, and one subset of features outperforms others. The proposed approach can be beneficial for building an automatic tagger based on question difficulty.

1. Introduction

Developers frequently use community-driven Question and Answering (Q&A) sites like Stack Overflow (SO) to solve inquiries related to programming. Every day, over 6000 new questions are posted on SO, and approximately 10 million users follow the site (Stack Exchange, 2009). The users, from beginners to experts, participate in constructive knowledge exchanges on this site, forming a dynamic programming community. Anyone can ask questions about various topics to fix their issues, and other users can respond or offer their thoughts. To make this procedure more user-friendly, SO offers several filtering and preference choices such as Interesting,[1] Bounties,[2] Watched Tags,[3] Ignore Tags for suggesting appropriate ones. However, with quantitative analysis on the live server,[4] we found that it takes around 16 days to get an answer while the standard deviation varies up to 113 days. Another major concern is the growth of knowledge shared in SO, as 30% of the total questions remain unanswered for one hour and have a response time of more than a day (Bhat et al., 2014).

Researchers have addressed this issue from different angles, such as: looking into the causes of unanswered questions and identifying various factors that lead to questions becoming unanswered. For example, Wang et al. (2018) conducted an empirical study on Stack Exchange websites to figure out the causes of the slow response times from the Q&A systems. They suggested that SO should improve their incentive systems for more active and faster answers and also consider the difficulty of answering the questions. To solve the unanswered questions, Mondal et al. (2021) explored the factors that contribute to unanswered questions and suggested models that predict potential unanswered questions.

Considering the aforementioned issues, researchers are now looking at the problem from a different perspective by assessing the difficulty of

☆ Editor: Raffaela Mirandola.
* Corresponding author at: Software Engineering Lab (SELab), Islamic University of Technology (IUT), Gazipur, 1704, Bangladesh.
E-mail addresses: malihanoushin@iut-dhaka.edu (M.N. Raida), zannatunnaim@iut-dhaka.edu (Z.N. Sristy), nawshinulfat@iut-dhaka.edu (N. Ulfat), moonwaraanjum@iut-dhaka.edu (S.M.A. Monisha), jubair@iut-dhaka.edu (Md.J.I. Mostafa), nazmul.haque@iut-dhaka.edu (Md.N. Haque).

1 https://stackoverflow.com/?tab=interesting
2 https://stackoverflow.com/?tab=bounties
3 https://stackoverflow.help/en/articles/5611335-watch-or-ignore-tags
4 https://data.stackexchange.com/stackoverflow/queries

https://doi.org/10.1016/j.jss.2023.111884
Received 3 January 2023; Received in revised form 22 August 2023; Accepted 24 October 2023
Available online 7 November 2023

a question based on various features and factors related to the question. Several approaches (Burel and He, 2013; Lin et al., 2014) have been proposed to understand the difficulty of the question. They considered the user's profile (number of questions/answers, reputation), question features (views, upvotes, and downvotes), and answer features (difference between the date a question was asked and the date the answer was given, the number of comments). However, the previous approaches did not consider the contents (textual information and code snippet) of the question as a feature. In response to this limitation, Neung and Twittie (Viriyadamrongkij and Senivongse, 2017a) proposed a "concept hierarchy" method that accounted for the vocabulary-based difficulty to measure the question's difficulty but acknowledged its limited applicability to generalization.

To overcome the constraints of the vocabulary approach in measuring question difficulty, Hassan et al. (2018a) used Term Frequency-Inverse Document Frequency (TF-IDF) based supervised learning technique, exploiting various feature lists. They introduced two feature lists ***Pre-Hoc***, the features related to the question and the questioner that are also available before receiving any response, and ***Post-Hoc***, which can be identified as the features that can be retrieved at a later stage when questions may have views, comments, and answers. Although they ignored the code snippets in the questions and any semantic or contextual features because of the nature of TF-IDF. Overall, existing literature hardly discusses the performance of different question difficulty estimation methods with a multitude of features (e.g., ***Pre-Hoc***, ***Post-Hoc***).

In this research, to address the preceding limitations, we considered the semantic and contextual features of the questions as well as the user's background. We categorized these features into three categories, introducing ***A Priori***, the features that can be extracted from the post only (includes text, code, and external links, etc.), which are available immediately following the posting of a question; reforming ***Pre-Hoc*** that additionally retrieve the questioner's information with A Priori (e.g., user reputation, badges, etc.) and ***Post-Hoc*** that includes response related features on top of ***Pre-Hoc*** (e.g., view count, question score, etc.) Utilizing these features, we proposed a variety of supervised models to estimate the difficulty of SO questions. The main objective of our models is to extract prominent features and classify the difficulty level of questions. We further extended our research to explore the relationship between the features and difficulty level.

To accomplish our objective with the proposed models, we first manually categorized the 738 randomly selected SO questions on Java that would serve as an addition to the 507 questions (Hassan et al., 2018a) that already existed but were small and lacking topic-independent data. The labeling was done into three classes - basic, intermediate, and advanced - based on their difficulties. The questions were labeled by four annotators while mentioning the reason for categorizing them in a particular class. Moreover, the major voting approach was followed for the final label. The labeled dataset is generic as they are independent of any particular topic.

We implemented three supervised learning models (Tf-Idf, Topic Modeling, and Doc2Vec) with different classifiers (Random Forest, XG-Boost, ADA-Boost, SVM) and evaluated them with different performance metrics. In the experimentation, Doc2Vec with XG-Boost outperformed other question difficulty classification models with the minimum number of features. To know the efficacy of ***Pre-Hoc*** and ***Post-Hoc*** feature sets, it is found that the robustness of ***Pre-Hoc*** features is relatively better for estimating the question difficulty level by using any classification models. Additionally, we identified correlation between characteristics and question difficulty; some features (e.g.,question size, LOC, accept rate) are positively related to question difficulty, while others (e.g.,number of views, answer count) are inversely related.

Thus, the main contributions of this paper are:

- We proposed to utilize contextual features and user profile information to estimate the difficulty of SO questions and evaluated the performance of supervised models (e.g.,Tf-Idf, Topic Modeling, and Doc2Vec) to classify the questions.
- We improved the existing ***Pre-Hoc*** and ***Post-Hoc*** features to capture contextual information and to make them more comprehensive. Moreover,we introduced ***A Priori*** feature set that can classify the difficulty of the newly asked questions with unknown questioners.
- We analyzed the relationship between the difficulty level of each question and different features i.e. question length, line of code, user reputation, badges, and different intervals.

The rest of the paper is organized as follows: Section 5 discusses the motivation of our study with an example. Section 2 discusses the related works. Section 3 describes the overall methodology of our study. The results are discussed in Section 4. The overall implications of our study are projected in Section 6. In Section 7, we mentioned all the threats and validated each of them. Lastly, in Section 8, we concluded our paper and mentioned potential future opportunities.

2. Related works

SO serves as a widely-used platform for users seeking programming solutions. However, as the volume of questions grows, it draws significant attention from the research community. It has been analyzed from different perspectives by researchers, including trending topics, question types, user engagement, etc., for the improvement of the current system. While some studies were based on content analysis, others were focused on user expertise assessment. Additionally, recommendation systems are another domain where a handful of researchers have put their efforts. Table 1 provides an overview of previous studies conducted in the aforementioned domains. In the subsequent subsections, a comprehensive description of each domain is presented. Finally, we presented a comparative analysis that points out how our work is different from previous works.

2.1. Content analysis

In content analysis, researchers are mostly concerned about identifying topics from questions, answers, and comments on SO posts. Allamanis and Sutton (2013) focused on categorizing questions by associating programming concepts and some identifiers using topic modeling. In a similar vein, Barua et al. (2014) employed Latent Dirichlet Allocation (LDA) to identify the main topics discussed by developers on SO, revealing 40 topics and analyzing changes in topic interests over time.

Shifting focus to API documentation, Treude and Robillard (2016) proposed a supervised approach called SISE to extract valuable insights from SO. Nonetheless, all of them ignored the connection between text descriptions and code snippets. Unlike the others, Ahmed and Bagherzadeh (2018) utilized LDA to measure the difficulty and popularity of concurrency topics on SO. However, their hypothesis on evaluating question difficulty based on the percentage of unaccepted answers and response times may not always be reliable.

In this regard, Lin et al. (2014) proposed the KG-DRank algorithm, which adjusted expertise based on the knowledge gap and achieved successful prediction of question difficulty. Whereas, Thukral et al. (2019) introduced a graph network-based method using textual descriptions, temporal effects, and user information. But the performance degraded with scattered topics, and the models were susceptible to the cold-start problems.

From a slightly different perspective, Amancio et al. (2021) conducted a thorough evaluation of ranking algorithms and identified Coordinate Ascent and LambdaMart as top performers for answer ranking and predicting answer recency. Yet, their study did not incorporate semantic analysis to determine answer relevance. Wang et al. (2013)

Table 1
Literature review at a glance.

Domain	Paper	Approach	Literature gap
Content Analysis	Ahmed and Bagherzadeh (2018), Allamanis and Sutton (2013), Amancio et al. (2021), Barua et al. (2014), Treude and Robillard (2016)	SO question and/or answer content analysis using Topic Modeling to extract textual features and gain insights on different research questions.	Focused on only topic trends and disregarded the code snippets attached to the content.
	Lin et al. (2014), Hassan et al. (2018b), Sun et al. (2018), Thukral et al. (2019), Viriyadamrongkij and Senivongse (2017b), Wang et al. (2013), Wang et al. (2014)	Using textual features and/or user metadata, questions from CQA sites are evaluated to determine their level of difficulty.	These studies often have a number of limitations in their research, including a small biased dataset, cold start issue handling, difficulty generalizing, and disregarding context with code snippets.
User Profiling	Sun et al. (2018), Diyanati et al. (2020), Yang et al. (2013)	User metadata and historical information such as answers and comments to determine the user expertise	Most of these works had been afflicted by cold start issues from the perspective of new users, and they ignored contextual elements when analyzing past user questions, answers, and comments.
Recommendation System	Li and King (2010), Wang et al. (2016), Wang et al. (2019)	Frameworks for routing the questions to appropriate users features like interest, previous answers, comments and/or activeness	Studies had issues with cold starts for new questions and/or new users. To route appropriate questions, authors overlooked question quality as well.

however proposed a competition-based model that leveraged pairwise comparisons, achieving accuracy in estimating question difficulty but facing limitations with cold-start problems due to its dependency on answerer skill. Wang et al. (2014) addressed these limitations by developing the Regularized Competition Model (RCM), which combined question-user comparisons and textual descriptions. But this approach was more suitable for non-technical questions as it ignored question context and code.

Later on, Viriyadamrongkij and Senivongse (2017b) used concept hierarchy and Q&A community features to determine question difficulty. Despite good performances, building concept hierarchies was space-inefficient, and relying on questioner information was not useful for anonymous or new users. Lastly, Hassan et al. (2018b) focused on textual and contextual features, and used a supervised model to obtain question difficulty levels. But the dataset was small and biased towards basic topics.

These studies collectively contribute to understanding and addressing the challenges in estimating question difficulty on SO. Yet, limitations remain in terms of semantic analysis, dependency on user participation and addressing cold-start problems.

2.2. User profiling

Researchers focus on user engagement by analyzing user expertise, activeness, and way of interaction i.e., comments, and answers to questions which leads to the study of user profiling. In this domain, Yang et al. (2013) introduced a Topic Expertise Model (TEM) combined with a joint model called CQARank, achieving high performance in recommending expert users and answers. Whereas, Diyanati et al. (2020) focused on determining user expertise based on question, answer and comment scores. But high dependency on Q&A voting history to determine user expertise limited their efficiency for new, infrequent users.

However, Sun et al. (2018) presented a competition graph-based framework with the user gain expertise (EGA) as the central element and also the textual features for addressing cold-start problem. Yet, reliance on textual comparisons limited its accuracy.

These studies provide insights into user expertise and question difficulty estimation. However, limitations persist regarding the cold-start problem, textual comparison, and scoring methods.

2.3. Recommendation system

In the domain of question routing Li and King (2010) proposed a Question Routing (QR) framework that directs questions to highly ranked answerers based on their past performance. The four-phase framework included performance profiling, expertise estimation, availability estimation, and answerer ranking. In another research, Wang et al. (2019) proposed the IEA approach, which leveraged user topical interest, topical expertise, and activeness derived from their questions, answers, and comments.

In the context of personalized recommendation, Wang et al. (2016) combined topic modeling (Twitter-LDA) (Negara et al., 2019) and link structure analysis (NEWHITS) to recommend appropriate experts for new questions. Although both of the aforementioned methods achieved high performance, did not effectively handle cold-start scenarios.

These papers help in understanding the underlying factors influencing the development of effective models for estimating question difficulty. However, challenges related to new or infrequent users remain unresolved.

2.4. How our study differs from others

The study on difficulty estimation on Q&A site has the challenge of the cold-start problem, like many of the aforementioned works Lin et al. (2014), Wang et al. (2013), Viriyadamrongkij and Senivongse (2017b) faced. These studies were not able to address the cold start problem in terms of any new question to be posted by any new user on the Q&A site. Most of these models used user engagement metadata to measure the difficulty of the questions, and a lack of information for new users was a hurdle in the path.

However, some of the recent studies took this problem into consideration, like Thukral et al. (2019) tried to solve the problem using K-nearest neighbor to find the similarity between a newly posted question and other questions of the same tags, but this model suffered in terms of their results in F1 score scoring about 56% or less for different K values. Then Hassan et al. (2018b) proposed their supervised algorithm to estimate the difficulty of the questions in SO. They also considered newly posted questions with their Pre-Hoc feature list, but their whole experiment was based on only three topics in Java, string, inheritance, and thread.

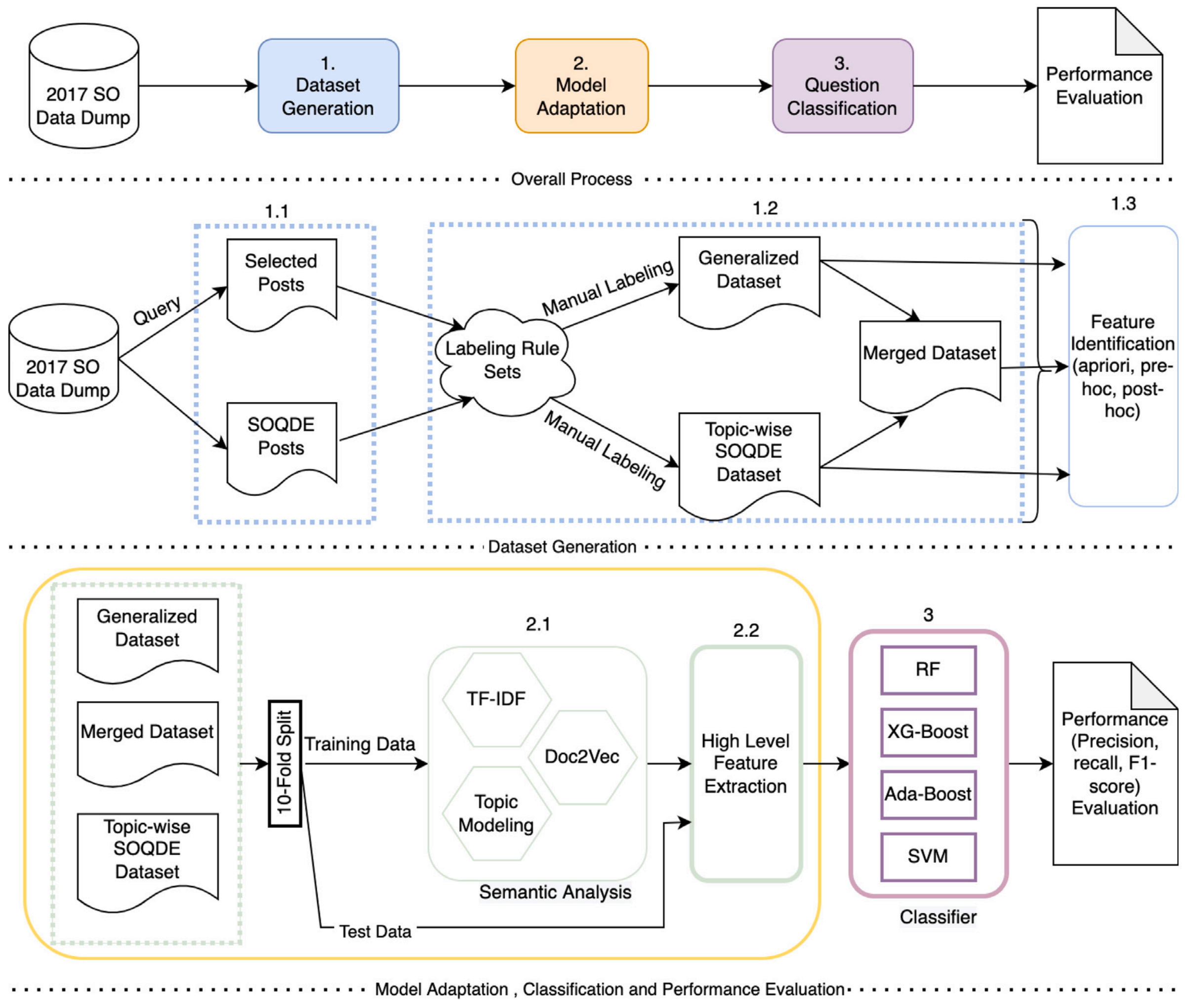


Fig. 1. Overall process of difficulty-based question classification, expanding the process of dataset generation, and model adaptation.

Similarly, Lin et al. (2014) proposed a difficult ranking algorithm, but the performance of the model fell behind when the topics in the question varied vastly. Also, Wang et al. (2014), Viriyadamrongkij and Senivongse (2017b), Sun et al. (2018) analyzed the content of the questions. Yet, these proposed models did not experiment with the contextual features that might impact the difficulty of the questions and only took the textual features into account. While content analyzing, Sun et al. (2018), Thukral et al. (2019), Wang et al. (2014) proposed methods did not consider the whole content, including the code, links which are commonly added to SO question and might impact the difficulty of any question.

Our proposed study took these concerns into account, and the experiment was built around Java, one of the most diverse languages in the industry according to the Jet Brains Survey 2020 (Jet Brains, 2020). To tackle the cold-start problem, we introduced A-priori features that can be extracted from any new question content, including the text, tags, code, and links. The result of the proposed methods exceeded the previous model, even with a minimum number of features.

3. Methodology

The overall approach consists of three parts, namely Dataset Generation, Model Adaption, and Question Classification, as shown in Fig. 1. Dataset Generation step extracts and prepares SO questions, which will be used as input in the Model Adaptation step. This generation step engages Data Extraction, Data Labeling, and Feature Identification steps described in the subsections of 3.1. In Section 3.2, Model Adaptation uses preprocessing step 3.2.1 with the Semantic Analysis and Feature Extraction to extract latent high level features, which are described in Sections 3.2.2 and 3.2.3 respectively. In Section 3.3, Classification uses various classifiers to classify questions by learning features of the Model Adaptation part. Lastly, the Performance Evaluation step utilizes different performance metrics (accuracy, precision, recall, F1-score, AUROC) to evaluate the effectiveness of different models along with the classifiers, which are discussed in Section 3.4.

3.1. Dataset generation

This section discusses the whole dataset generation process from the SO for the question classification models. Firstly, a stable curated data dump of SO is selected, and based on some informative criteria, a specific portion of data is extracted, which is discussed in Section 3.1.1. Secondly, the researchers execute the labeling process, which is thoroughly described in the Data Labeling Section 3.1.2. And lastly, in Section 3.1.3, various features are identified which could be relevant to understanding difficulty wise question classification.

3.1.1. Data extraction

As SO contains a diverse area of programming knowledge, including technology, domain, and language, it is not feasible to cover all aspects of programming knowledge for difficulty assessment. So, we confined ourselves to focusing only on Java related questions and answers and extracted those datasets from SO. Java is one of the popular languages that developers are using for building enterprise solutions (Stack

Overflow, 2020). The SO 2020 survey illustrates the fact that about 40.2% (Stack Overflow, 2020) of the developers are using Java, putting it in the fifth position. 38.4% of respondents who are professional developers have chosen Java as their programming language. This information undoubtedly makes Java a beneficial language for learners. A significant amount of research has been conducted on Java to identify the key traits of the development process (de Castro et al., 2016; Jiang, 2020). Throughout the world, the Java programming language is broadly used, starting from websites, system software, data storage, IT infrastructure, and data science, according to a survey done in 2020 by Jetbrains (Jet Brains, 2020). Hence Java is chosen as a representative of SO questions.

To capture the overall scenario of the posts generally answered in Stack Overflow, a stable and timely distributed dataset is needed to be extracted. A significant number of research works leverage such data for the research purpose, including API documentation from SO posts (Treude and Robillard, 2016), and connecting to Integrated Development Environment (IDE) (Bacchelli et al., 2012), comprehensive research on the legacy data of Java community (Blanco et al., 2020) and so on.

Following the data consistency, we used the SO data dump of December 2017,[5] which was stored in the 2008 SQL Server database for running queries locally and extracting SO posts. The whole data dump consists of posts from 2012 to 2017, which is informative for expressing a deep-rooted effect of users' and posts' characteristics. Here, incorporating more recent data is not applicable for two reasons. First, we used only textual information of a question to extract **A Priori** and **Pre-Hoc** features. Thus, recent questions do not add any value. Second, more recent data are not stable enough for extracting Post-hoc features as the number of views, answers, comments, and the overall score may increase gradually. Moreover, we compared our approach with the SOQDE approach (using Random Forest classifier with TF-IDF textual model), which is also based on the same data dump of Java programming language.

We developed a series of queries constraining on Java tag, question score, owner id, answer count, and such to get a subset of a relevant dataset that would satisfy the requirements for a certain question to be valid and generic. The queries of Listing 1, 2 and 3 are three of those queries.

Here, the SQL in Listing 1 represents a query to find the questions having a "java" tag in the year 2017 SO dataset, whereas Listing 3 finds the questions which have a "java" tag and the first answer was found after 30 days with a positive score. Listing 2 query was used to calculate the difference between the creation of every question and its first answer.

```sql
select Id, ViewCount,Title,AnswerCount,Tags,Score from
    Posts
where PostTypeId=1
and AcceptedAnswerId Is not null
and AnswerCount>0
and Score >10
and YEAR(CreationDate)= 2017
and Tags like '%<java>%'
order by AnswerCount desc;
```

Listing 1: Query to find java questions in the year of 2017.

Out of 2000 queried questions, we randomly picked a total of 750 questions for the next step of manual labeling (described in the following Section 3.1.2). During the manual inspection, we had to exclude 12 questions because of not having all the features needed for model training. Each post is extracted with post id, post title, and post body. To keep the labeling process unbiased, we did not extract any other information related to Q/A threads or users' history. However, after completion of labeling, contextual and user history-related features are identified and extracted for each post in Section 3.1.3.

```sql
select x.Id as QuestionId,y.Id as AnswerId
,x.CreationDate as QuestionDate,y.CreationDate as
    AnswerDate,
DATEDIFF(Day,x.CreationDate,y.CreationDate) as Interval
from (select * from Posts where PostTypeId=1) x LEFT
    JOIN
(
   SELECT *, ROW_NUMBER()
   OVER(PARTITION BY ParentId ORDER BY CreationDate)
    AS RowNo
   FROM Posts
   where PostTypeId=2
) y
on  x.Id=y.ParentId and y.RowNo=1
where y.Id is not NULL;
```

Listing 2: Query to calculate the difference between the creation of every question and its first answer.

```sql
select x.Id as QuestionId,x.Title,x.Score,y.Id as
    AnswerId,x.CreationDate as QuestionDate,y.
    CreationDate as AnswerDate, DATEDIFF(Day,x.
    CreationDate,y.CreationDate) as Interval
from (select * from Posts where PostTypeId=1) x LEFT
    JOIN
(
   SELECT *, ROW_NUMBER() OVER(PARTITION BY ParentId
    ORDER BY CreationDate) AS RowNo
   FROM Posts
  where PostTypeId=2
) y
on  x.Id=y.ParentId and y.RowNo=1
where y.Id is not NULL
and x.Tags like '%<java>%'
and DATEDIFF(Day,x.CreationDate,y.CreationDate)>30
and x.Score >0
Order by Interval desc;
```

Listing 3: Query to find java questions answered 30 days later having positive scoring.

3.1.2. Data labeling

Before initiating the manual data labeling process, it was necessary to establish a set of rules that would guide the annotators. The initial ruleset was obtained from SOQDE (Hassan et al., 2018a), which categorized the rules into Basic, Intermediate, and Advanced levels. However, the chosen ruleset often contained general and incomplete rules. This ambiguity could lead to confusion and raise questions among the annotators. So, the given rules were broken down into more granular degrees for better understanding and a clearer detection of class in the labeling process. In the Table 2, the first column indicates the labels to be given to the posts, the second column depicts the ruleset from the existing literature, the third column represents the breakdown of each of the main rules, and the last column exhibits an example that falls under each of the granular rules.

After the ruleset was made, it was provided to the annotators to guide them. They examined the title and entire body of each post to manually categorize the questions.

The initial annotation was carried out by the first four authors, and expert validation was sought to mitigate any confusion with the labels. Majority voting was employed, with each post being labeled by all four authors. To prevent topic-based biases, the posts were randomly assigned to the annotators. After the authors completed their labeling, Cohen's Kappa (Cohen, 1960) or Fleiss's Kappa (Fleiss, 1971) can be used to assess the level of agreement among them. However, Fleiss's Kappa is more applicable to measure the agreement for more than 2 annotators. So we computed the value and obtained the result of Fleiss's kappa, $\kappa = 0.723$,[6] indicating a high level of agreement. If at least three out of the four authors agreed on the difficulty level of a question, that difficulty is assigned to the question. The same approach

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

[6] https://doi.org/10.6084/m9.figshare.23059934.v1

was applied when two authors agreed on one difficulty label while the other two authors selected different options. In cases where there was an equal split of votes (two-two), the experts were consulted to collectively decide on the final label.

By following the aforementioned process, we successfully labeled both the dataset obtained from SOQDE (Hassan et al., 2018a) and our own dataset, resulting in a total of 1245 labeled questions. Table 3 presents the three datasets along with the respective counts of questions categorized under each difficulty level. The first dataset is named after the original paper and focuses exclusively on three specific Java topics: threads, inheritance, and strings. The second dataset is a generalized version encompassing various topics based on our labeling. The last dataset combines the previously mentioned two sets, accumulating their contents.

3.1.3. *Feature identification*

The labeled dataset only consists of three features: post identification number, post body, and post title. However, it is difficult to make a decision from this limited number of features. Hence, analyzing existing literature and performing manual inspection motivated us to extract more features. From the existing literature, Hassan et al. (2018a), various features, including the question body, response time, question score, view count, etc., are identified and considered as the prominent features (Table 4). In addition, after manual inspection, some contextual information, like post owner history and answer-related features described in Table 4, are also considered.

In total, 18 features represented in Table 4 were utilized to determine the difficulty of a question. These three types of features cover both cold-start situations with only question text and warm-start situations with contextual information for difficulty assessment. **A Priori** feature set consists of only the question content, irrespective of the user history or metadata. Thus, any new question posted by a new user with no history in the system can be taken into consideration. All other features set, **Pre-hoc** and **Post-hoc** are taken into account for the warm start that will be applicable for the vast amount of questions that consist of information about user history and engagement. Fig. 2 presents an actual SO question-answering thread along with the identified features.

For the “Processed Body” feature, after extracting the code snippets, the considered post body was appended to the title and tags. The code snippets are the section that is written between the anchor tag of `<code></code>`. So, each textual and code section could be featured separately for document analysis models. The user profile details were considered for every questioner and answerer, and features like reputation, accept rate, and badges were scraped from the data dump at that timestamp. We also considered details extracted from the post body, like Line of Code (LOC), code snippets, URLs, and image counts.

Extracted code snippet kept for further calculation of the line of code using Pygout,[7] a python command-line tool that counts only physical lines of source code. For each snippet in a particular post, we measured the LOC metrics. The line number of codes is combined if a post has more than one code snippet. Moreover, the feature of $URL + image$ count refers to the number of hyperlinks in a certain post that might be used to clarify the posted question, including images, live codes, or questioning concepts. These features, along with the dataset, were used as input in the following Model Adaptation step 3.2.

3.2. *Model adaptation*

To predict a question’s difficulty class, we preprocessed the post body, including the title and tags (removing code snippets), to harvest the textual features and assemble them with the features coming from Table 4. This section will describe all document analyzing models that can have a satisfactory effect on the difficulty classification for the documented questions. The overall process is described in Fig. 1 (2.1, 2.2). Here, we first preprocessed the data related to the question 3.2.1. Then, the contextual information is extracted using semantic analysis 3.2.2. Lastly, We extracted the semantically high-level features and projected them into our dataset to make a distinguishable version.

3.2.1. *Preprocessing*

After the dataset preparation, we preprocess the new post body, excluding the code snippets. This preprocessed data is used in the following Semantic Analysis 3.2.2.

- **Title, Tags, and Textual Body Composition**:
  The title and tags from a question describe the post’s main aspect and straightforward concept. After the title and tags are included in the body, the new body would represent the whole post narrative, which would help us discover a further semantic association between body, title, and tags.
- **Tokenization**:
  We tokenized each post into smaller units (words) to extract meaningful terms and their occurrences. And the word tokenizer used for the SO dataset was from the Gensim (Rehurek and Sojka, 2011) library’s simple_preprocess function that includes a lower casing, removing any accent marks from the sentence and storing only words with a minimum length of 3 to a list of tokens.
- **Stop Words Removal**:
  Generating the list of tokens, the elimination of the stopwords, like articles, pronouns, and prepositions, was performed using the most commonly used NLTK (Bird et al., 2009) library for its documented English stopwords appending it to the Stanford CoreNLP stopwords list.
- **Stemming & Lemmatization**:
  The examination of each word to convert it to its original form was also executed using the NLTK library’s well-defined Snowball stemming function. And before lemmatization, we used SpaCy (Honnibal and Montani, 2017), an open-source NLP library, with the `en_code_web_sm` model for tagging the words and allowing only ‘NOUN’, ‘ADJ’, ‘VERB’, ‘ADV’ to be part of the further calculation. Now, lemmatization would take place to the words to dictionary form of words with the same SpaCy library.
- **Bag of Word (BOW)**:
  The list of filtered tokens would be converted to a dictionary, having each of the words mapped to a unique identity number. And for this purpose, we used the Gensim library’s corpora package. Using this dictionary, we created the BOW for each sentence.
- **Frequency Limitation**:
  We limited word frequency to eliminate the outliers not to be added to the models in further steps, which may affect the results. To identify the frequency limit, we needed to plot the frequency of each word in all the documents cumulatively. And decided to exclude words that appear less than 30 times in general after plotting the total frequency of each word appearing in the documents (i.e., posts) under consideration. We found a very small number of words having an appearance less than the set limit. And each of these words was added to the stop words and eliminated from each of the documents. For example, some of the eliminated words were related to numbers which had a rare probability of appearing in other questions in the same manner.

3.2.2. *Semantic analysis*

In this part, we experimented with three well-known models: the TF-IDF (Hassan et al., 2018a), a numeric statistical measure; Latent Dirichlet Allocation (LDA) (Allamanis and Sutton, 2013; Wang et al., 2016) and the documentation vectorization technique, Doc2Vec (Le and Mikolov, 2014).

[7] https://pygount.readthedocs.io/en/latest/

Table 2
Labeling rule set for measuring question difficulty.

Difficulty class	General rule set	Granular breakdown	Example question
Basic	Questions on simple built-in functions/API documentation/beginner level knowledge	Regular Built-in-function	How can I pad a String in Java?
		Simple Operator/Expression	Check if at least two out of three booleans are true
		API documentation	Add a dependency in Maven
		Beginner level Theory Question	In Java, difference between package private, public, protected, and private
		Basic OOP problem	How to determine an object's class (in Java)?
	Questions related to comparison between concepts and functions of various languages	Analysis of various languages' functions	Difference between StringBuilder and StringBuffer
		Beginner level difference related query	The difference between the Runnable and Callable interfaces in Java
		Simple problem solving in other languages	What is the JavaScript version of sleep()?
	Questions about simple problem-solving or random topic	Simple problem solving	How to create a method to return 1 if input is provided 0 and 0 if provided 1 without using conditions?
		Random query	Is String.length() invoked for a final String?
	Questions with simple exception, error and other problem	Solve for nullpointer exception	Boolean.valueOf() produces NullPointer-Exception sometimes
		Simple Error Handling	getSupportFragmentManager() shows a compile time error
		Configuration problem solved by documentation	maven build failed: Unable to locate the Javac Compiler in: jre or jdk issue
Intermediate	Questions that require a relatively deeper understanding of the language to answer, for example Why type questions	Advance features of a language that require deeper understanding	How do I use a PriorityQueue?
		Need more knowledge about the algorithms	Consistency of hashCode() on a Java string
		Multiple questions in a single post	How to launch a java program with precisely controlled execution time?
		Need knowledge on Advanced Programming topics	Logback to log different messages to two files
		Difference between two packages	Joda Time and Java8 Time difference
	Questions where the questioner knows about the answer/solution but wants to know more efficient one	Looking for contextually suitable solution despite having a solution	Copying TIMESTAMP to DATETIME on MySQL with Hibernate
		Analyzing various solutions for execution time	Java Timer vs ExecutorService?
	Questions related to time complexity, memory usage or other different resource usages of a system/solution	Efficient way	Fastest way to determine if an integer's square root is an integer
		Performance, optimization, accuracy	Memory allocation problems with android application
		Memory related	Freeing memory wrapped with NewDirectByteBuffer
	Questions need conceptual reasoning of programming construct/design principle	Reverse programming	How do I "decompile" Java class files?
		Underlying philosophy of any programming construct	Why use getters and setters/accessors?
		Design pattern	What is Alternative to Singleton
		Feasibility study	Setting up Java + svn + Eclipse+ Tomcat, development environment with docker
		Question about built-in documentation in details	Java 9: How to find every new method added
		Required Testing Related Knowledge	What is the proper way to setup and seed a database with artificial data for integration testing

(*continued on next page*)

TF-IDF considers each question as a document and each of the unique words in that document as terms to compute document-wise term distribution. To do so, we used the preprocessed data from Section 3.2.1 and calculated the TF-IDF using Eq. (1).

$$TF\text{-}IDF_{i,j} = TF_{i,j} \times log(\frac{N}{df_i}) \tag{1}$$

Here, $TF_{i,j}$ denotes the frequency of the unique word, j in the question/post, i. N is the total number of questions. The value df_i represents the number of questions that contain the word i. The result of the TF-IDF was used to get a vector representation of features, which had more than 1600 features for each question.

Table 2 (*continued*).

Difficulty class	General rule set	Granular breakdown	Example question
Advanced	Questions that deal with hard/critical problems where solution needs in-depth programming knowledge or conceptual/logical thinking	Solution needs in-depth programming knowledge or conceptual thinking	JPMS and cannot understand its dynamism, Can it be done in Java 9 module system
		Despite of large analysis, several unsolved issues	Incompatible types, equality constraints and method not found during Java 9 Migration
		Improvement of existing answers	How to implement retry policies while sending data to another application?
	Questions that require advanced in-depth knowledge of internal language structure	In-depth knowledge of internal language structure	Spring RedisConnectionFactory with transaction not returning connection to Pool and then blocks when exhausted
		In-depth knowledge of packages	Publish a bom from a multi-module-project
		In-depth knowledge on Garbage Collection	Latencies issues which G1GC
	Questions that deals with infrequently used functions	Deals with infrequently/rarely used framework/API/functions	Using Ebean for persistence
		Deals with depricated framework/functions/API	How do I properly map a ‘MagImageScalingCallback‘ using JNA?
	Question that requires in-depth knowledge about software architecture & SDLC	In-depth knowledge of software architecture	JPA Clean Architecture
		In-depth knowledge of software maintenance	How do I upgrade to jlink (JDK 9+) from Java Web Start (JDK 8) for an auto-updating application?
		In-depth testing and security knowledge	RESTful Authentication via Spring
	Related to production environment	Efficiency related question	Why is AES encryption/decryption more than 3x slower on Android 24+?
		Deployment related question	GWT Deployment on Tomcat 5.5
	Question that deals with large data set and diversified topics	Data mining/ Deep learning/Artificial Intelligence	Deeplearning4j - using an RNN/LSTM for audio signal processing
		Need in-depth knowledge of multiple topics	Unable to identify source of java.lang.ClassNotFoundException BaseDexClassLoader

Table 3
Dataset wise class distribution.

Dataset name	Total no.	Class distribution		
	of samples	Basic	Intermediate	Advanced
SOQDE	507	375	104	28
Generalized	738	360	305	73
Merged	1245	735	409	101

Since TF-IDF cannot be capable of capturing a composite representation of information in terms of topic, we considered topic modeling, a popularly used model for representing relevant topics from the question. Specifically, we used LDA (Blei et al., 2003) to find the abstract topics from any question using Eq. (2).

$$P(\beta, \theta, z, w) = \left(\prod_{i=1}^{K} p(\beta_i|\eta)\right)\left(\prod_{d=1}^{D} p(\theta_d|\alpha)\right.$$
$$\left.\prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:K}, z_{d,n})\right) \quad (2)$$

Here, β, θ, and z denote the distribution of words (1st part of the equation) for K number of topics, the topic proportion of a question (2nd part of the equation) and topic assignment of a word in a question (last part of the equation), respectively. For yielding an LDA model, we applied Gensim's LDA model, which took the created corpus, id to word mapped dictionary from the preprocessing step, and lastly, the number of topics for finding from each of the questions. The topic number is a hyperparameter we had the chance to choose for our model. We experimented by setting topic numbers from 20 to 40, and we found that our model performs the best using the number of topics set to 23.

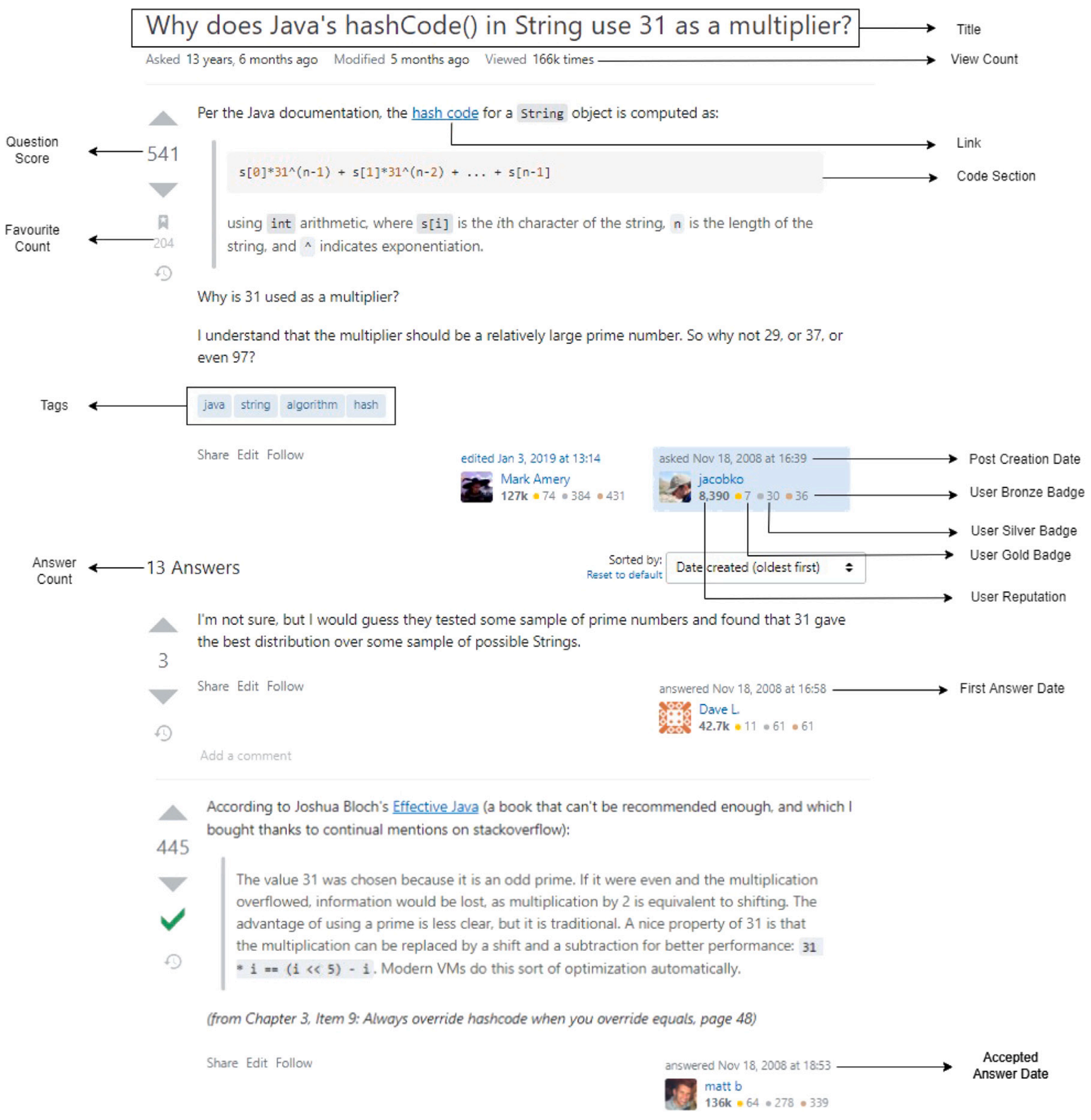
LDA discards some contextual information from the question with its BOW approach, as it prefers to represent the statistical relationship of occurrences. Consequently, it may lose some possible good representation and semantic information of the question because of the consideration of word order. Doc2Vec, a modified version of Word2Vec, allows learning the real semantic information representing the whole question as a vector. The architecture of the Doc2Vec is shown in Fig. 3.

We built it with Gensim's Doc2Vec model with parameters of vector size as a variable and min_count set to 2, which would remove words having a frequency less than two and epoch number over the whole dataset. Given the training set, the first task was to construct a vocabulary dictionary from the stream of questions. Then the vocabulary dictionary and the preprocessed corpus were passed to train the Doc2Vec model. For any question, this model can infer the vector representation of that question. Users can set vector length, so we tried vector sizes from 20 to 40, and the best score of accuracy was given by the vector size 36. The accompanying code has been provided for further reference.[8]

3.2.3. *High level feature extraction*

As it is inconvenient to classify the difficulty of a question from the low-level features, we transform these low-level features into high-level semantically rich features which have a higher classification, recognition, and segmentation capabilities. Generally, high-level features were built on top of the low-level feature, like a scratch from an image. It contains the information from the raw text (e.g., a different topic from

[8] https://doi.org/10.6084/m9.figshare.19726621.v1


Why does Java's hashCode() in String use 31 as a multiplier?

Asked 13 years, 6 months ago Modified 5 months ago Viewed 166k times

541

204

Per the Java documentation, the hash code for a `String` object is computed as:

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

using `int` arithmetic, where `s[i]` is the *i*th character of the string, `n` is the length of the string, and `^` indicates exponentiation.

Why is 31 used as a multiplier?

I understand that the multiplier should be a relatively large prime number. So why not 29, or 37, or even 97?

java string algorithm hash

Share Edit Follow

edited Jan 3, 2019 at 13:14 Mark Amery 127k ● 74 ● 384 ● 431

asked Nov 18, 2008 at 16:39 jacobko 8,390 ● 7 ● 30 ● 36

13 Answers

Sorted by: Reset to default Date created (oldest first)

3

I'm not sure, but I would guess they tested some sample of prime numbers and found that 31 gave the best distribution over some sample of possible Strings.

Share Edit Follow

answered Nov 18, 2008 at 16:58 Dave L. 42.7k ● 11 ● 61 ● 61

Add a comment

445

According to Joshua Bloch's Effective Java (a book that can't be recommended enough, and which I bought thanks to continual mentions on stackoverflow):

The value 31 was chosen because it is an odd prime. If it were even and the multiplication overflowed, information would be lost, as multiplication by 2 is equivalent to shifting. The advantage of using a prime is less clear, but it is traditional. A nice property of 31 is that the multiplication can be replaced by a shift and a subtraction for better performance: `31 * i == (i << 5) - i`. Modern VMs do this sort of optimization automatically.

(from Chapter 3, Item 9: Always override hashcode when you override equals, page 48)

Share Edit Follow

answered Nov 18, 2008 at 18:53 matt b 136k ● 64 ● 278 ● 339

Fig. 2. Stack Overflow question & answer thread indicating various features.

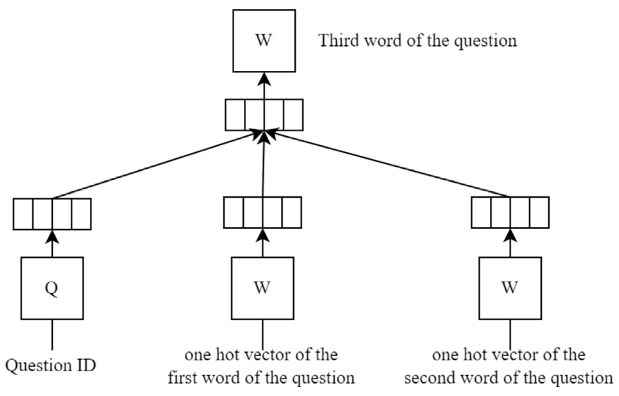

W
Third word of the question
Q
W
W
Question ID
one hot vector of the first word of the question
one hot vector of the second word of the question

Fig. 3. Architecture of Doc2Vec.

a large document, a contextual summary of a document) that is easy to understand and recognize.

To extract the high-level, semantically rich features from the low-level dataset, after getting the high-level features from the Semantic Analysis 3.2.2 subsection, we projected these features into our dataset to make a transformed version of our dataset, which is shown in Fig. 1 (2.2). This transformed dataset had high-level features with their corresponding transformed values. This transformed dataset was used for classification, described in the next Section 3.3.

3.3. Question classification

As all textual analysis models provided the features representing the post body in vectorized format, we incorporated the extracted textual features with the features divided into ***A Priori***, ***Pre-Hoc*** and ***Post-Hoc*** categories. We applied different multi-class classifiers to the dataset to determine the best model for extracting textual features and their correlation with the other features. The most used classifier for text classification (Onan et al., 2016), and handling complex high dimensional feature space (Bloehdorn and Hotho, 2006) while being robust to overfitting e.g., Random Forest (Ho, 1995), XGBoost (Chen and Guestrin, 2016), Adaboost (Schapire, 2013), and lastly, SVM (Cortes and Vapnik, 1995).

Table 4
The features and their definition with associated feature list.

Feature name	Definition	Included in
Processed Body	Post full textual body excluding code snippets and the html tags like <p>, <code>,<href>	A Priori,Pre-hoc,Post-hoc
Tags	Post tags, decided at the time of posting, e.g. <java><oop><multithread>	A Priori,Pre-hoc,Post-hoc
Title	Post title, decided by questioner	A Priori,Pre-hoc,Post-hoc
Question_Length	Length of the whole Processed Body	A Priori,Pre-hoc,Post-hoc
Url+Image_Count	Number links the post	A Priori,Pre-hoc,Post-hoc
LOC	Line of Code, counting only physical lines of source code in snippet extracted from post body Summing up all the LOCs from a certain post	A Priori,Pre-hoc,Post-hoc
User_Reputation	User Reputation Point given by Stack Overflow activities like answering, questioning	Pre-hoc,Post-hoc
User_Bronze_Badge	Number of awards for basic use of the site	Pre-hoc,Post-hoc
User_Gold_Badge	Number of awards for important contributions from members of the community	Pre-hoc,Post-hoc
User_Silver_Badge	Number of awards for being experienced users who regularly use Stack Overflow	Pre-hoc,Post-hoc
Accept_Rate	The percentage of answers accepted based on the questions asked by the user.	Pre-hoc,Post-hoc
View_Count	Number of time a certain question is viewed by users	Post-hoc
Favorite_Count	Number of times a certain question is saved as favorite	Post-hoc
Up_vote_Count	Number of up votes on a certain question for being useful and appropriate	Post-hoc
Answer Count	Number of answers in a certain question–answer thread	Post-hoc
Question_Score	The total number of upvotes it received minus the total number of downvotes it received in a question	Post-hoc
First_Answer_Interval	Interval in days between question creation date to first answer creation	Post-hoc
Accepted_Answer_Interval	Interval in days between question creation date to accepted answer creation	Post-hoc

To perform classification on the dataset, we executed K-fold cross-validation where K = 10, using the Sklearn (Pedregosa et al., 2011) library of Python. And the whole models with the classifier were run ten times to train and test. The aforementioned classification models were tuned by a trial-error process using the parameters to get a better classifier. Here we discuss all the classifiers with the necessary variables that were set to conduct our experiment.

- **Random Forest:** It is an ensemble machine learning classification technique having multiple decision trees. We set 15 decision trees with a depth of 8 for each of the trees, and the criterion for trees was chosen to be entropy.
- **XGBoost:** It is a parallel tree-boosting classification algorithm. We used the boosting rounds of 40 with a learning rate of 0.05. And the maximum tree depth was set to 8 as before.
- **Adaboost:** It is an iterative ensemble classification technique that combines multiple classifiers to increase the accuracy of classifiers. In our experiment, we used 1000 classifiers and a learning rate of 0.05.
- **SVM:** It is a supervised classification technique with the objective of finding a hyperplane in an n-dimensional space that distinctly classifies the questions based on their difficulties. It was built for the one-vs-rest ('ovr') decision function of shape and enabled the probability estimation to fit the training data by measuring performance metrics.

We calculated five performance metrics for each fold, such as accuracy, precision, recall, F-1 score, and AUROC using Sklearn's metrics package. After completing all folds, the average for each metric was calculated to compare the text analyzing models' performances.

3.4. Performance evaluation

To assess the performance of different adaption models and classifiers, we considered five performance metrics: accuracy, precision, recall, F-1 score, and Area Under the Curve & Receiver Operating Characteristic curve (AUROC). Accuracy is the ratio of the samples that are predicted as true from the total samples. It is used to measure how close a given question is to its actual difficulty. The following equation is used to measure it.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3}$$

Here, *TP* and *TN* are the numbers of positive and negative samples that are correctly classified. *FP* is the number of negative-class samples misclassified as the positive class, and *FN* is the number of positive-class samples misclassified as the negative class. In contrast, accuracy is a measurement of both positive and negative samples, and precision measures only the positive samples. Precision measures the relevancy of results by computing $\frac{TP}{TP+FP}$ while recall measures truly relevant results by computing $\frac{TP}{TP+FN}$.

Having uneven class distribution, we computed F1-score and AUROC, well-known metrics for class imbalance problems. We computed F1-score to capture the weighted average of correctly identifying difficulty and total identified difficulty using the following equation.

$$F1\ Score = \frac{2 * (Recall * Precision)}{(Recall + Precision)} \tag{4}$$

Along with F1-score, $AUROC$ represents the degree or measure of separability between classes, and it can be used in both balanced and imbalanced datasets, especially imbalanced datasets. ROC is a probability curve of a classifier at various thresholds. It plots a curve based on the true positive rate (*TPR*) and false positive rate (*FPR*) represented in Eqs. (5) and (6).

$$TPR = \frac{TP}{TP + FN} \tag{5}$$

$$FPR = \frac{FP}{FP + TN} \tag{6}$$

In these equations, *TP*, *TN*, *FP* and *FN* denote the same meaning of Eq. (3). To compute the points in a ROC curve, $AUROC$ computes an aggregate measure of various thresholds. Using these metrics, we analyzed the performance of the various models and classifiers in Sections 4.1 and 4.2. The value of these metrics lies within the range of 0 to 1. A score of 0 indicates the poor performance of the classifier, and a score of 1 indicates good performance.

Table 5
Performance metrics of Tf-Idf model, TM model and Doc2Vec model with different classifiers using A Priori Features of Question.

A priori features of question						
Classifier	Model	Accuracy	Precision	Recall	F1-score	AUROC
RandomForest	Tf-Idf	0.615	0.545	0.615	0.509	0.698
	TM	0.624	0.569	0.624	0.549	0.702
	Doc2Vec	0.643	0.608	0.643	0.594	0.713
XG-Boost	Tf-Idf	0.653	0.61	0.653	0.622	0.746
	TM	0.62	0.582	0.62	0.579	0.68
	Doc2Vec	**0.656**	**0.625**	**0.656**	**0.626**	**0.746**
Ada-Boost	Tf-Idf	0.643	0.577	0.643	0.59	0.712
	TM	0.632	0.586	0.632	0.581	0.612
	Doc2Vec	0.659	0.633	0.659	0.625	0.674
SVM	Tf-Idf	0.63	0.548	0.63	0.56	0.71
	TM	0.63	0.548	0.63	0.56	0.71
	Doc2Vec	0.63	0.548	0.63	0.56	0.722

Table 6
Performance metrics of Tf-Idf model, TM model and Doc2Vec model with different classifiers using Pre-hoc Features.

Pre-hoc features						
Classifier	Model	Accuracy	Precision	Recall	F1-score	AUROC
Random Forest	Tf-Idf	0.626	0.553	0.626	0.525	0.711
	TM	0.623	0.55	0.623	0.549	0.673
	Doc2Vec	0.655	0.628	0.655	0.605	0.746
XG-Boost	Tf-Idf	0.646	0.602	0.645	0.615	0.741
	TM	0.622	0.587	0.622	0.586	0.679
	Doc2Vec	**0.657**	**0.64**	**0.657**	**0.629**	**0.744**
Ada-Boost	Tf-Idf	0.644	0.577	0.644	0.591	0.71
	TM	0.621	0.582	0.621	0.576	0.61
	Doc2Vec	0.663	0.64	0.663	0.63	0.673
SVM	Tf-Idf	0.586	0.423	0.586	0.446	0.62
	TM	0.586	0.423	0.586	0.446	0.637
	Doc2Vec	0.586	0.456	0.586	0.448	0.635

Table 7
Performance metrics of Tf-Idf model, TM model and Doc2Vec model with different classifiers using Post-hoc Features.

Post-hoc features						
Classifier	Model	Accuracy	Precision	Recall	F1-score	AUROC
Random Forest	Tf-Idf	0.618	0.554	0.618	0.52	0.743
	TM	0.647	0.613	0.647	0.597	0.721
	Doc2Vec	0.648	0.62	0.648	0.603	0.719
XG-Boost	Tf-Idf	0.663	0.637	0.663	0.638	0.762
	TM	0.644	0.61	0.644	0.617	0.734
	Doc2Vec	**0.659**	**0.636**	**0.659**	**0.632**	**0.738**
Ada-Boost	Tf-Idf	0.652	0.59	0.652	0.608	0.729
	TM	0.654	0.629	0.654	0.628	0.658
	Doc2Vec	0.667	0.643	0.67	0.646	0.692
SVM	Tf-Idf	0.594	0.487	0.594	0.454	0.677
	TM	0.594	0.482	0.594	0.454	0.676
	Doc2Vec	0.594	0.482	0.594	0.454	0.68

4. Results & discussion

In this section, we investigated and answered three research questions.

- **RQ 1:** Which model performs well to define question difficulty level?
- **RQ 2:** How do ***Pre-Hoc*** features perform in comparison to ***Post-Hoc*** features to classify questions based on difficulty?
- **RQ 3:** How do different features correlate with the difficulty level of a question?

Firstly, we compared the three text analysis models along with four classifiers for each type of feature set separately in Section 4.1. Secondly, in Section 4.2, the efficiency of the ***Pre-Hoc*** feature set was analyzed in contrast to the ***Post-Hoc*** feature set for different datasets. Finally, the relationship between features and the question difficulty level is explored in Section 4.3 to provide insights into the characteristics of questions of various complexities.

4.1. Performance of question classification models

Tables 5–7 summarized the comparative results of different classification models on the merged dataset using three types of features (***A Priori***, ***Pre-Hoc*** and ***Post-Hoc*** features) and the values in boldface represent the best performing method for a particular classifier.

For ***A Priori*** features, the performance of three models using different classifiers is shown in Table 5. As we can see, for almost every classifier, the Doc2Vec model outperforms the other two models. This is because topic modeling finds the topics (that incorporate some words) from a document. Whereas Tf-Idf finds the most important words of the document to find the key concept of the document. Since the neighboring words and word sequencing are not taken into account by the last two models, these two models sometimes miss some relevant insights of the question. On the other hand, Doc2Vec considers the concept of a word in the surrounding of other words in a document which helps to draw more insightful conclusions to estimate the question's complexity based on tags or topics in accordance with the question scenario.

Among the classifiers, AdaBoost has the highest accuracy because of its sequentially growing learnability. But it has relatively lower coverage due to overfitting and longer time requirement from an algorithmic perspective. However, XGBoost additionally employs parameter regularization with sequential adaptation, which greatly lowers overfitting and improves the classifier as a whole, making it a better option. XGBoost with the Doc2Vec model can classify the question based on difficulty with an accuracy of 0.656, F1-score 0.626 and AUROC 0.746.

Tables 6 and 7 show the performance metrics of textual models for all the classifiers with ***Pre-Hoc*** features and ***Post-Hoc*** features. For ***Pre-Hoc*** features, the Doc2Vec model using XGBoost classifier provides better performance, with accuracy, F1-score and AUROC of 0.657, 0.629 and 0.744 respectively. But surprisingly ***Post-Hoc*** feature set, the performance of the Tf-Idf model clearly surpasses that of both the TM and Doc2Vec models.

However, the datasets of the Tf-Idf model were split after calculating the Tf-Idf score for overall data to keep the feature set constant. As a result, this model gains the advantage of learning the test set earlier, which is not the case for any classification or filtering system and renders it ineffective as a question classifier in real life. Hence, we can overlook the Tf-Idf model, and in comparison to the TM model, we can infer that the Doc2Vec model performs better, with an accuracy of 0.659, an F1-score of 0.632, and an AUROC of 0.738.

Finally, as the feature set grows, the overall performance of all question classification models improves. Unlike the TM model, where adding features boosts performance significantly, the performance improvement of Doc2Vec with less important features is somewhat slower as it is more context-dependent.

Answer to RQ1. In general, the Doc2Vec model with XGBoost classifier surpasses all other models in classifying questions based on difficulty, with an accuracy of 0.657 and AUROC of 0.738. It can also filter questions with a high degree of accuracy for any unknown or new user with no prior information.

4.2. Comparative analysis of ***Pre-Hoc*** *and* ***Post-Hoc*** *features*

To understand the change in performance of ***Pre-Hoc*** and ***Post-Hoc*** features, we analyzed the performances of different models on topic-wise, generalized, and merged datasets. The performance comparison

Table 8
Performance comparison of Pre-hoc and Post-hoc features for TM model using different data sets.

TM model						
Metrics	Existing dataset (Topic wise)		Our dataset (Random)		Merged dataset	
	Pre-hoc	Post-hoc	Pre-hoc	Post-hoc	Pre-hoc	Post-hoc
Accuracy	0.733	0.747	0.58	0.597	0.622	0.644
Precision	0.662	0.674	0.578	0.587	0.587	0.61
Recall	0.733	0.747	0.58	0.597	0.622	0.644
F1-Score	0.683	0.696	0.567	0.585	0.586	0.617
AUROC	0.632	0.703	0.692	0.72	0.679	0.734

Table 9
Performance comparison of Pre-hoc and Post-hoc features for Doc2Vec model using different data sets.

Doc2vec model						
Metrics	Existing dataset (Topic wise)		Our dataset (Random)		Merged dataset	
	Pre-hoc	Post-hoc	Pre-hoc	Post-hoc	Pre-hoc	Post-hoc
Accuracy	0.739	0.732	0.598	0.629	0.657	0.659
Precision	0.668	0.648	0.592	0.625	0.64	0.636
Recall	0.739	0.732	0.598	0.629	0.657	0.659
F1-Score	0.689	0.675	0.578	0.617	0.629	0.632
AUROC	0.693	0.725	0.721	0.746	0.744	0.738

is important since the goal of our study is to forecast a question's difficulty level and filter them accordingly. As a result, it reduces the response time and unanswered questions. ***Pre-Hoc*** features are chosen such that they are available whenever a new question emerges, making them better for filtering purposes, whereas ***Post-Hoc*** features are only available after the question has been resolved. Since the main purpose of this comparison is to analyze the performance of the models for routing, ***A Priori*** features are not considered separately.

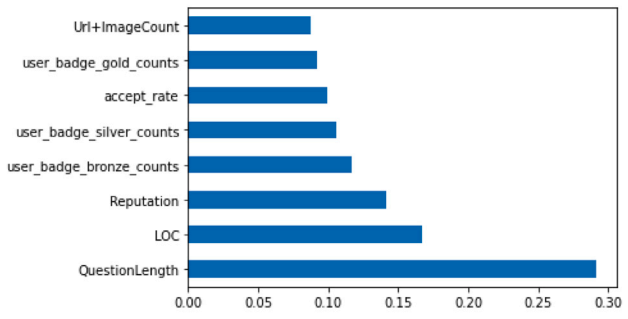
Tables 8 and 9 represent the performance metrics of each textual model for ***Pre-Hoc*** and ***Post-Hoc*** features side by side in the context of different data sets. We dropped the Tf-Idf model since it is unsuitable for question filtering in a real-world setting.

According to the tables, in both models, ***Post-Hoc*** features surpass ***Pre-Hoc*** features by not more than 0.04 for any performance indicator, with slightly better coverage. Even with random data set which incorporates a wide range of topics, ***Pre-Hoc*** features can classify questions with an accuracy of 0.58 and 0.598 using TM and Doc2Vec models, respectively. In contrast, ***Post-Hoc*** features can classify with an almost identical accuracy of 0.597 and 0.629 using TM and Doc2Vec models, respectively. This is because the extra information that ***Post-Hoc*** contains is less cohesive than other features. As a result, the effect of including those features is somewhat insignificant.

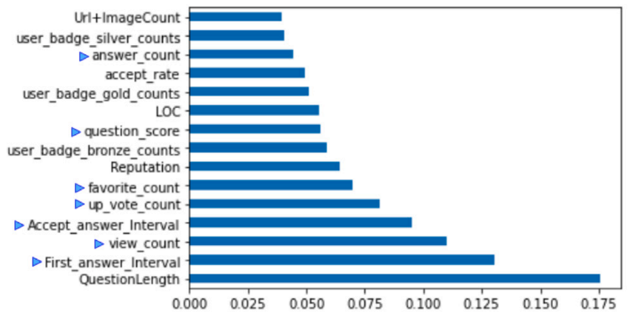
Figs. 4(a) & 4(b) demonstrate the importance of each feature in Pre-hoc and Post-hoc feature sets except for question title, question content, tags, question size as they are used in the semantic analysis model (TF-IDF, TM, Doc2Vec) which in combined actually has the highest importance. In the figures, importance refers to how much a feature contributes to classifying questions based on their difficulty. The horizontal bars reflect the percentage values of the importance of features.

Among the ***Pre-Hoc*** features, the most important features are question length, LOC, and reputation. Whereas for ***Post-Hoc*** features, question length, first_answer_interval, view_count, accept_answer_interval, up_vote_count are the most significant.

Answer to RQ2. ***Pre-Hoc*** features perform nearly identical to ***Post-Hoc*** features in both the TM-based and Doc2Vec-based models, with a maximum difference of 0.04 for any performance metric. So, it can be inferred that ***Pre-Hoc*** characteristics are sufficient for filtering questions based on difficulty.



(a) ***Pre-Hoc*** Features



(b) ***Post-Hoc*** Features

Fig. 4. Importance of ***Pre-Hoc*** and ***Post-Hoc*** features.

4.3. *Correlation between features and question difficulty level*

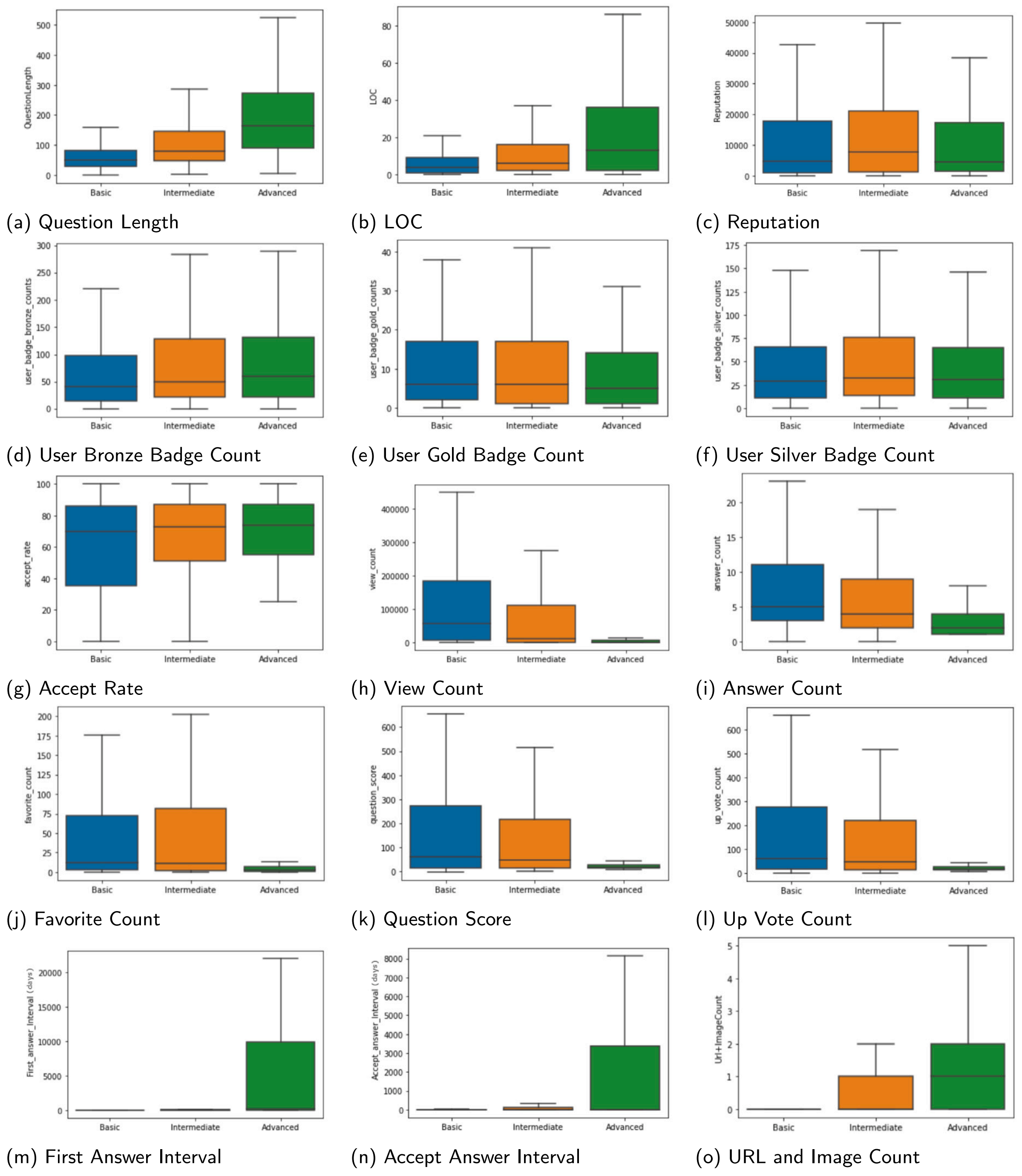
Table 10 and Fig. 5 depict how the value of features change as complexity increases. It provides some interesting insights into the relationship between features and question difficulty levels which can be useful for selecting features to filter questions in future studies.

Previously researchers (Hassan et al., 2018a) suggested that the complexity of the question is proportional to the length of the question. Our study took a step further in this direction and elaborated that the code size measured in LOC Table 10[1, 2] & Fig. 5[a, b] is also proportionate to the difficulty of the questions. To identify the reason behind this relationship, we uncover two plausible explanations. The first is that deciphering lengthier codes is more difficult. The second issue is that people tend to include as much content (textual and code) as possible to communicate a complex subject properly. As a result, while the number of code lines increases, so does the difficulty.

With the rising complexity of a question, both the View_Count and the Answer_Count (Table 10[8, 9] & Fig. 5[h][i]) rapidly fall as users choose to go through the questions that they can understand.

As for the features, Favorite_Count, Question_Score, and Up_Vote_ Count (Table 10[10, 11, 12]), we can see that on average, the intermediate level questions gain the highest score while advanced level ones receive the lowest. However, the density distribution of Question Score and Up_Vote_Count (Fig. 5[k][l]) of the questions gradually falls towards a lower value as the difficulty arises. So it is safe to infer that users prefer a certain amount of brainstorming to solve a problem, but they do not want to spend too much time and mental energy comprehending a single question.

Other strong measures of the complexity of the questions are the User_Reputation and the User_Badges. However, one essential point to note is that people with a higher reputation ask intermediate-level questions, while those with a lower reputation ask the most challenging questions (Table 10[3] & Fig. 5[c]). Knowing that upvotes, accepted answers, and bounty all contribute to user reputation, it is easy to see why people who ask intermediate-level questions have a greater reputation than those who ask tough questions.



(a) Question Length (b) LOC (c) Reputation
(d) User Bronze Badge Count (e) User Gold Badge Count (f) User Silver Badge Count
(g) Accept Rate (h) View Count (i) Answer Count
(j) Favorite Count (k) Question Score (l) Up Vote Count
(m) First Answer Interval (n) Accept Answer Interval (o) URL and Image Count

Fig. 5. Boxplot analysis of question difficulty and density distribution of different features.

Before moving into user badges, let us figure out what kind of badge is provided for what purpose. For basic site usage, bronze badges are awarded. Silver badges are awarded to experienced users who use the site frequently. The most dedicated users earn gold badges. On average, questioners of intermediate level inquiries have the most user badges, and questioners of advanced level queries have the least (Table 10[4, 5, 6]). This is because active users frequently ask efficiency-related or why-type questions, whereas infrequent users occasionally ask about critical/rare situations. On the other hand, the badges' density distribution depicted in Fig. 5[d][e][f] differs from the averages of the Table 10 data while tacitly conveying the same. According to the distribution, users who ask the more challenging questions have a higher number of bronze badges (Fig. 5[d]) than those who ask the easy ones. In contrast, the questioners of intermediate level questions have more gold and silver badges (Fig. 5[e][f]), whereas questioners of advanced level questions have fewer. This implies that frequent and motivated users are more interested in intermediate level questions, while irregular users are more interested in difficult questions.

The Accept_Rate (Table 10[7] & Fig. 5[g]), on the other hand, rises with difficulty level, implying that users improve their ability to ask questions based on their domain expertise. They ask less irrelevant, redundant, or unclear inquiries the more knowledgeable they are.

However, as the difficulty of the question increases, so does the First_Answer_Interval and Accepted_Answer_Interval (Table 10[13, 14] & Fig. 5[m][n]), and the maximum of these intervals lengthen significantly for advanced questions.

Table 10
Changes of different features according to question difficulty.

Index	Features	Basic (736)	Intermediate (410)	Advanced (102)
		Avg	Avg	Avg
1	Question size	66	115	212
2	LOC	7	13	23
3	User Reputation	22 193	25 352	16 397
4	User_Bronze_Badge	96	112	90
5	User_Gold_Badge	17	18	11
6	User_Silver_Badge	62	75	56
7	Accept Rate	58	63	64
8	View Count	189 697	99 439	21 408
9	Answer Count	9	8	4
10	Favorite_Count	81	124	24
11	Question Score	281	289	77
12	Up_Vote_Count	283	290	78
13	First_Answer_Interval	4956	6959	21 278
14	Accepted_Answer_Interval	27 902	40 154	73 601
15	Url+Image_Count	0.3	1	2

Finally, although the relationship between URL+image counts and difficulty is not particularly strong, simple questions typically contain fewer subsidiary resources than difficult ones (Table 10[15] & Fig. 5[o]), indicating that extra supplementary resources must be provided in order to properly communicate with tough questions.

Answer to RQ3. While question size, LOC, accept rate, URL+image count, as well as first and accepted answer interval, are all proportionally related to question difficulty, view and answer count are inversely proportional.
However, user reputation and badges, question score, favorite, and upvote count improve up to the intermediate level difficulty but drop for advanced level questions.

5. An illustrative example

Our work is motivated by the need to address the challenges in tagging system, faced on online judges for programming challenges, such as LeetCode,[9] Hackerearth,[10] and others. These platforms utilize tags (e.g., Tree, Graph) and difficulty levels (e.g., Easy, Medium, Hard) to categorize problems. We have drawn inspiration from the concept of difficulty levels on these platforms to enhance the problem-solving experience on SO. By incorporating difficulty levels, users can more easily find questions that align with their expertise, allowing them to choose worthwhile questions to answer. Furthermore, our objective is to apply machine learning techniques to classify the difficulty level of questions.

This would enable users to not only search for questions based on specific tags, but also consider the anticipated difficulty level. For instance, a user asked a question with the relevant information.

Question: “How do I convert a string to an integer number in Java?”
SO Tags: Java, String

Now, if the answerer searched for questions with the tag Java or string, SO gives a list of Java or string related questions based on user-selected filters (e.g., unanswered, newest, etc.).

The goal of our proposed method is to classify the difficulty of the questions with the highest possible accuracy at any time after the question is posted. To do so, we incorporated a supervised approach where a set of questions is labeled with a difficulty level by following a set of rules 2. Several classifiers are trained and tested to check the performance of the approach. So, in a real usage scenario, the moment a question is posted by any anonymous user, our model will start collecting the attributes from the question's textual elements, namely the question body, question title, tags, code snippet extracted from the question, number of links attached, and so on. These attributes are then fed into our proposed model's intricate classification system, which effectively assesses the difficulty level. If the user's information is available, it becomes empowered to extract pre-hoc attributes, encompassing both textual cues and user-specific features. By considering these attributes, our model is adept at determining a question's inherent difficulty level. Even after a considerable period has elapsed since a question's initial posting, our model accumulates a comprehensive set of post-hoc attributes. These attributes encapsulate the previous pre-hoc features with the collective feedback of the community—measured through votes, views, and answers. With these comprehensive post-hoc features, our model is continuously refined, ensuring that its evaluation remains relevant and receptive to the evolving dynamics of the user experience. This adaptive model is capable of categorizing questions based on their difficulty levels. This model forms the bedrock of an advanced recommendation engine, which would match questions with users depending on their proficiency level and interests.

While suggesting questions solely based on user-defined tag(s), suggested questions are limited to exact matching; our proposed method would assign a difficulty level to each of the questions based on latent textual information. This additional layer of difficulty measurement complements existing filtering options on SO, providing answerers with a more comprehensive set of questions to explore.

With difficulty levels, the proposed approach encourages novice users to engage with and attempt to answer basic questions that are relatively easier. At the same time, expert users are prompted to tackle more challenging questions. It enhances engagement and fosters a dynamic culture among all users. Thus, our approach does not replace existing methods of searching and answering questions on SO, but rather introduces a novel perspective inspired by the practices commonly employed by online judges.

6. Implications

Difficulty-wise SO question classification, has significant importance in facilitating coordination between knowledge seekers and appropriate responders. We have discussed implications for practitioners and researchers.

6.1. Implications for practitioners

Users can ask questions related to programming and other technical domains on Stack Overflow. However, not all questions are equal in terms of difficulty or the knowledge required to provide a meaningful answer.

By employing a question difficulty measurement system, both questioners and answerers can be benefited. For questioners, it helps to ensure that their questions are appropriately categorized based on the level of knowledge required to answer them. This reduces the chance of not receiving any answers due to users' limitations or lack of motivation.

For instance, when a user asks any question that requires moderate or high knowledge in relevant domains, it should be distinguished from trivial questions. Treating all questions equally can result in the issue of unanswered questions, as certain questions may necessitate specialized expertise.

Moreover, experienced users who have already gained a good reputation on the site may be less inclined to engage in answering trivial

[9] https://leetcode.com
[10] https://www.hackerearth.com

questions, as they may seek more challenging or complex problems to contribute according to their expertise. On the other hand, novice users with sufficient knowledge may find great enthusiasm in answering relatively simpler questions.

By incorporating our approach that recognizes the varying levels of difficulty in questions and promotes the participation of users with relevant knowledge and motivation, Stack Overflow can improve the overall quality and effectiveness of knowledge sharing on the platform.

6.2. Implications for researchers

With its growing popularity among programmers, SO has also become a fascinating research topic. The research area of this topic ranged from analyzing current programming philosophy to providing various functional and algorithmic enhancements.

Although our study mainly focuses on providing new functionality, it requires analyzing different SO features. So we prepared three different feature sets named ***A Priori***, ***Pre-Hoc***, ***Post-Hoc*** which cover different purposes. We found that the feature sets are quite efficient for representing the nature of the question. While contextual features like question body, title, and tag provide the most important insights, the auxiliary features about the question and questioners help to predict the type of the questions.

Nevertheless, it is time-consuming for researchers to manually curate difficulty-wise categorized SO posts. The labeled dataset of 1245 posts, along with its associated feature set, is a good source for analyzing the posts of SO and the characteristics of its users.

We proposed classifiers that perform well in categorizing SO questions according to their difficulty level. This difficulty-wise classification will further help to construct a difficulty-wise question recommendation system or facilitate a personalized profile setup. Moreover, our study did not use any language or topic-specific features, so it can be useful for any question-answering site as well.

7. Threats to validity

While conducting our study, some threats and challenging aspects hinder the validity of our work. These threats need to be mentioned to establish the effectiveness of our proposed approach. Threats are mentioned by being categorized into Internal, External, and Construct Validity. All of these are explained in the following subsections.

7.1. Internal validity

Threats to internal validity include the models' tendency towards topic-based question difficulty classification that may overlook the actual context of the inquiry. We mitigated this threat by employing a dataset in which questions were chosen randomly regardless of the topic. Our experimental dataset has 908 distinct tags, ensuring that the training and testing set is topic diverse.

Another concerning factor is the incorporation of features that are highly related to the language since we have considered only Java language. But we did not use any language-specific feature to keep it extendable for other languages. Moreover, we selected Java because it is very popular and mature and covers a lot of programming philosophies, and the Java community exhibits the characteristics of any generic programming community.

7.2. External validity

Threats to external validity concern the generalizability of our models for any scenario. While several features are available for simply assessing question difficulty, there can only be a limited number of features accessible to enable difficulty-based question filtering, which may reduce the models' efficiency. To address this problem, we prepared different feature sets like ***A Priori*** for completely new questions (i.e., the cold start) and ***Pre-Hoc*** features for usual situations where questioner's features are available. Using these, we trained our models accordingly for assessment. Finally, we found that, even with the cold start issue, all models have an accuracy of at least 0.60, which is fairly satisfactory.

7.3. Construct validity

The main challenge regarding construct validity is labeling the dataset. With anonymous majority voting, we attempted to mitigate this threat. At least four researchers labeled each question separately. In case of conflicts, we discussed and resolved them through expert judgment. The Cohen's Kappa value for inter-rater agreement also indicates the validity of our labeling process.

To ensure the validity of the rules set, we focused on Hassan et al. (2018a)'s rules and expanded those into granular levels along with some new rules. These comprehensive rules helped to reduce any changes of disagreement and ensured the validity of our approach.

8. Future works & conclusion

Question difficulty estimation is important to improve the routing and tagging policy of Stack Overflow. For this reason, we proposed different semantic models to determine the questions' difficulty. We utilized various contextual features and classifiers to approximate the difficulty of such questions. For experimental purposes, we curated 1245 difficulty-wise labeled questions manually, which capture generalized and topic-wise data variation. It provides a good taxonomy of rule sets for difficulty-wise question labeling as a byproduct. We implemented the semantic models with different classifiers and evaluated the performance of those models along with the features sets (***A Priori***, ***Pri-Hoc*** and ***Post-Hoc***) using multiple metrics like F1-score, AUROC. It has been found that the Doc2Vec model and XG-Boost classifier achieved the best performance than other models. From the feature usefulness analysis, we found that ***Pre-hoc*** features are sufficient to classify a question with limited information. Moreover, different features have a proportional and inverse relationship concerning the question difficulty.

The proposed approach demonstrates that applying semantic analysis like Doc2Vec is useful for representing high-level features of questions. This provides the opportunity to understand a question's hidden knowledge representation without human intervention, like moderators' actions. Researchers can use other deep learning techniques to apprehend the knowledge base. Besides, to determine the questions' difficulty, our approach uses three feature sets. Researchers can consider similar features to improve the performance of the classifiers.

In the future, we plan to work on creating a recommendation system that would recognize the hidden relations between the question types, considering the information related to the question and the user's historical information of collaboration. This will enable us to recommend the questions to appropriate users with enough expertise to answer.

CRediT authorship contribution statement

Maliha Noushin Raida: Project administration, Conceptualization, Methodology, Data curation, Software, Visualization, Writing – original draft. **Zannatun Naim Sristy:** Conceptualization, Methodology, Data curation, Software, Investigation, Writing – original draft. **Nawshin Ulfat:** Conceptualization, Data curation, Writing – original draft. **Sheikh Moonwara Anjum Monisha:** Conceptualization, Data curation, Investigation. **Md. Jubair Ibna Mostafa:** Methodology, Writing – original draft, Supervision, Writing – review & editing. **Md. Nazmul Haque:** Methodology, Writing – original draft, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Ahmed, S., Bagherzadeh, M., 2018. What do concurrency developers ask about? A large-scale study using stack overflow. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '18, Association for Computing Machinery, New York, NY, USA, pp. 1–10.

Allamanis, M., Sutton, C., 2013. Why, when, and what: Analyzing stack overflow questions by topic, type, and code. In: 2013 10th Working Conference on Mining Software Repositories. MSR, pp. 53–56.

Amancio, L., Dorneles, C.F., Dalip, D.H., 2021. Recency and quality-based ranking question in CQAs: A stack overflow case study. Inf. Process. Manage. 58 (4), 102552.

Bacchelli, A., Ponzanelli, L., Lanza, M., 2012. Harnessing stack overflow for the ide. In: 2012 Third International Workshop on Recommendation Systems for Software Engineering. RSSE, IEEE, pp. 26–30.

Barua, A., Thomas, S.W., Hassan, A.E., 2014. What are developers talking about? An analysis of topics and trends in stack overflow. Empir. Softw. Eng. 19 (3), 619–654.

Bhat, V., Gokhale, A., Jadhav, R., Pudipeddi, J., Akoglu, L., 2014. Min(e)d your tags: Analysis of question response time in stackoverflow. In: Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. ASONAM '14, IEEE Press, pp. 328–335.

Bird, S., Klein, E., Loper, E., 2009. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media, Inc.

Blanco, G., Pérez-López, R., Fdez-Riverola, F., Lourenço, A.M.G., 2020. Understanding the social evolution of the java community in stack overflow: A 10-year study of developer interactions. Future Gener. Comput. Syst. 105, 446–454.

Blei, D.M., Ng, A.Y., Jordan, M.I., 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3 (Jan), 993–1022.

Bloehdorn, S., Hotho, A., 2006. Boosting for text classification with semantic features. In: Mobasher, B., Nasraoui, O., Liu, B., Masand, B. (Eds.), Advances in Web Mining and Web Usage Analysis. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 149–166.

Burel, G., He, Y., 2013. A question of complexity: Measuring the maturity of online enquiry communities. In: Proceedings of the 24th ACM Conference on Hypertext and Social Media. pp. 1–10.

de Castro, C.F., de Souza Oliveira, D., Eler, M.M., 2016. Identifying characteristics of java methods that may influence branch coverage: An exploratory study on open source projects. In: Cubillos, C., Astudillo, H. (Eds.), 35th International Conference of the Chilean Computer Science Society. SCCC 2016, ValparaíSo, Chile, October 10-14, 2016, IEEE, pp. 1–8.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining. pp. 785–794.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Meas. 20 (1), 37–46.

Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20 (3), 273–297.

Diyanati, A., Sheykhahmadloo, B.S., Fakhrahmad, S.M., Sadredini, M.H., Diyanati, M.H., 2020. A proposed approach to determining expertise level of StackOverflow programmers based on mining of user comments. J. Comput. Lang. 61, 101000.

Fleiss, J.L., 1971. Measuring nominal scale agreement among many raters. Psychol. Bull. 76 (5), 378.

Hassan, S.A., Das, D., Iqbal, A., Bosu, A., Shahriyar, R., Ahmed, T., 2018a. SOQDE: A supervised learning based question difficulty estimation model for stack overflow. In: 2018 25th Asia-Pacific Software Engineering Conference. APSEC, pp. 445–454.

Hassan, S.A., Das, D., Iqbal, A., Bosu, A., Shahriyar, R., Ahmed, T., 2018b. SOQDE: A supervised learning based question difficulty estimation model for stack overflow. In: 2018 25th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 445–454.

Ho, T.K., 1995. Random decision forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition, Vol. 1. IEEE, pp. 278–282.

Honnibal, M., Montani, I., 2017. Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. Unpublished software application. https://spacy.io.

Jet Brains, 2020. Java programming - The state of developer ecosystem in 2020 infographic. URL https://www.jetbrains.com/lp/devecosystem-2020/java/.

Jiang, Y., 2020. Research on application value of computer software development in Java programming language. In: Journal of Physics: Conference Series, Vol. 1648. IOP Publishing, 032152.

Le, Q., Mikolov, T., 2014. Distributed representations of sentences and documents. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32. ICML '14, JMLR.org, pp. II–1188–II–1196.

Li, B., King, I., 2010. Routing questions to appropriate answerers in community question answering services. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management. CIKM '10, Association for Computing Machinery, New York, NY, USA, pp. 1585–1588.

Lin, C.-L., Chen, Y.-L., Kao, H.-Y., 2014. Question difficulty evaluation by knowledge gap analysis in question answer communities. In: 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. ASONAM 2014, IEEE, pp. 336–339.

Mondal, S., Saifullah, C.K., Bhattacharjee, A., Rahman, M.M., Roy, C.K., 2021. Early detection and guidelines to improve unanswered questions on stack overflow. In: 14th Innovations in Software Engineering Conference (Formerly Known As India Software Engineering Conference). pp. 1–11.

Negara, E.S., Triadi, D., Andryani, R., 2019. Topic modelling Twitter data with latent Dirichlet allocation method. In: 2019 International Conference on Electrical Engineering and Computer Science. ICECOS, pp. 386–390.

Onan, A., Korukoğlu, S., Bulut, H., 2016. Ensemble of keyword extraction methods and classifiers in text classification. Expert Syst. Appl. 57, 232–247, URL https://www.sciencedirect.com/science/article/pii/S0957417416301464.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al., 2011. Scikit-learn: Machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.

Rehurek, R., Sojka, P., 2011. Gensim–Python Framework for Vector Space Modelling, Vol. 3, No. 2. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, p. 2.

Schapire, R.E., 2013. Explaining adaboost. In: Empirical Inference. Springer, pp. 37–52.

Stack Exchange, 2009. All sites - Stack exchange. URL https://stackexchange.com/sites?view=list#traffic.

Stack Overflow, 2020. Stack overflow developer survey 2020. URL https://insights.stackoverflow.com/survey/2020#technology.

Sun, J., Moosavi, S., Ramnath, R., Parthasarathy, S., 2018. QDEE: Question difficulty and expertise estimation in community question answering sites. CoRR, abs/1804.00109.

Thukral, D., Pandey, A., Gupta, R., Goyal, V., Chakraborty, T., 2019. DiffQue: Estimating relative difficulty of questions in community question answering services. ACM Trans. Intell. Syst. Technol. 10, 42:1–42:27.

Treude, C., Robillard, M.P., 2016. Augmenting API documentation with insights from stack overflow. In: 2016 IEEE/ACM 38th International Conference on Software Engineering. ICSE, IEEE, pp. 392–403.

Viriyadamrongkij, N., Senivongse, T., 2017a. Measuring difficulty levels of JavaScript questions in question-answer community based on concept hierarchy. In: 2017 14th International Joint Conference on Computer Science and Software Engineering. JCSSE, pp. 1–6.

Viriyadamrongkij, N., Senivongse, T., 2017b. Measuring difficulty levels of JavaScript questions in question-answer community based on concept hierarchy. In: 2017 14th International Joint Conference on Computer Science and Software Engineering. JCSSE, IEEE, pp. 1–6.

Wang, S., Chen, T.-H.P., Hassan, A., 2018. Understanding the factors for fast answers in technical q&a websites: an empirical study of four stack exchange websites. In: Proceedings of the 40th International Conference on Software Engineering.

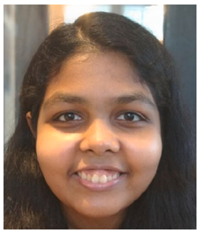
Wang, Q., Liu, J., Wang, B., Guo, L., 2013. Question difficulty estimation in community question answering services. In: EMNLP. pp. 85–90.

Wang, Q., Liu, J., Wang, B., Guo, L., 2014. A regularized competition model for question difficulty estimation in community question answering services. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. EMNLP, Association for Computational Linguistics, Doha, Qatar, pp. 1115–1126.

Wang, L., Wu, B., Yang, J., Peng, S., 2016. Personalized recommendation for new questions in community question answering. In: 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. ASONAM, IEEE, pp. 901–908.

Wang, L., Zhang, L., Jiang, J., 2019. IEA: An answerer recommendation approach on stack overflow. Sci. China Inf. Sci. 62 (11), 1–19.

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



Maliha Noushin Raida received a B.S. degree in software engineering from the Islamic University of Technology, Gazipur, Bangladesh, in 2022. She has previously worked as web development intern at JoomShaper. Currently, She is performing her duty as a lecturer in Computer Science and Engineering department of Islamic University of Technology, Gazipur, Bangladesh. She is also presently carrying on with her research in software repository mining and code quality analysis. Her field of interests include machine learning, data mining and software engineering.

Zannatun Naim Sristy received a B.S. degree in Software Engineering from the Islamic University of Technology, Gazipur, Bangladesh, in 2022. Currently, she is caring her career as a lecturer in Computer Science and Engineering department of Islamic University of Technology, while doing her research in software repository mining and code quality analysis. Her research interests include machine learning, data mining, data analysis and visualization, software engineering, software quality analysis.

Nawshin Ulfat received a B.S. degree in Software Engineering from the Islamic University of Technology, Gazipur, Bangladesh, in 2022. She has previous work experience in the software development field. She is currently working a as a lecturer in Computer Science and Engineering department at Green University, Bangladesh. Her research interests are software engineering, software quality analysis, machine learning, embedded engineering, data mining, analysis, and visualization.

Sheikh Moonwara Anjum Monisha received a B.S. degree in Software Engineering from the Islamic University of Technology, Gazipur, Bangladesh, in 2022. She has previously worked at Streams Tech Ltd. and Kernel Technologies as an intern. Present days, she is working as a lecturer in Computer Science and Engineering department at Bangladesh University of Business and Technology. She is also doing her research in software repository mining and code quality analysis. Her research interest are machine learning, software engineering, software quality analysis, data mining.

Md. Jubair Ibna Mostafa received the B.S. degree in software engineering from University of Dhaka, in 2017, and the M.S. degree in software engineering from the same university, in 2019. Currently, he is working as an Assistant Professor in Computer Science and Engineering department of Islamic University of Technology, Gazipur, Bangladesh. He is actively doing research on code smells, open source software repository, and collaborative platform like Stack Overflow mining from the software maintenance perspective.

Md. Nazmul Haque received the B.S. and M.S degree in software engineering from University of Dhaka, in 2017 and 2020. Currently, he is working as an Assistant Professor in Computer Science and Engineering department of Islamic University of Technology, Gazipur, Bangladesh. He is a former Software Engineer at the Samsung R & D Institute, Bangladesh. His research interest lies in Deep Learning, Software Engineering.