

# Duplicate Question Retrieval and Confirmation Time Prediction in Software Communities

Rima Hazra  
IIT Kharagpur  
India

to\_rima@iitkgp.ac.in

Debanjan Saha  
IIT Kharagpur  
India

debanjansaha@iitkgp.ac.in

Amrui Sahoo  
IIT Kharagpur  
India

amrui2k@iitkgp.ac.in

Somnath Banerjee  
IIT Kharagpur  
India

som.iitkgpcse@kgpian.iitkgp.ac.in

Animesh Mukherjee  
IIT Kharagpur  
India

animeshm@cse.iitkgp.ac.in

**Abstract**—Community Question Answering (CQA) in different domains is growing at a large scale because of the availability of several platforms and huge shareable information among users. With the rapid growth of such online platforms, a massive amount of archived data makes it difficult for moderators to retrieve possible duplicates for a new question and identify and confirm existing question pairs as duplicates at the right time. This problem is even more critical in CQAs corresponding to large software systems like askubuntu where moderators need to be experts to comprehend something as a duplicate. Note that the prime challenge in such CQA platforms is that the moderators are themselves experts and are therefore usually extremely busy with their time being extraordinarily expensive. To facilitate the task of the moderators, in this work, we have tackled two significant issues for the askubuntu CQA platform: (1) retrieval of duplicate questions given a new question and (2) duplicate question confirmation time prediction. In the first task, we focus on retrieving duplicate questions from a question pool for a particular newly posted question. In the second task, we solve a regression problem to rank a pair of questions that could potentially take a long time to get confirmed as duplicates. For duplicate question retrieval, we propose a Siamese neural network based approach by exploiting both text and network-based features, which outperforms several state-of-the-art baseline techniques. Our method outperforms DupPredictor [33] and DUPE [1] by 5% and 7% respectively. For duplicate confirmation time prediction, we have used both the standard machine learning models and neural network along with the text and graph-based features. We obtain Spearman's rank correlation of 0.20 and 0.213 (statistically significant) for text and graph based features respectively. We shall place all our codes and data in the public domain upon acceptance.

**Index Terms**—community question answering, duplicate retrieval, duplicate confirmation time, tag co-occurrence network, negative sampling

## I. INTRODUCTION

Community question answering (CQA) platforms are rapidly becoming popular because of their extensive collection

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASONAM '23, November 6-9, 2023, Kusadasi, Turkey

© 2023 Association for Computing Machinery.

ACM ISBN 979-8-4007-0409-3/23/11...\$15.00

<http://dx.doi.org/10.1145/3625007.3627310>

of questions and answers. Due to the burgeoning growth of such CQA portals, questions posted by users can be repetitive. In many cases, new users tend to post duplicate questions since they are not fully aware of the navigation tools available on the platform. Moderators/experienced users need to identify and mark duplicate questions in such cases. This becomes extremely challenging and time-consuming given the scale of data they need to sieve through. While posting a new question, if a user is prompted with similar (or precisely the same) queries reported previously, it can reduce the platform's redundancy. CQAs pertaining to large software systems like askubuntu pose a larger challenge since the moderators need to be mostly experts to identify if a question is a duplicate. The availability of such experts is limited and usually quite expensive. Further confirming a pair of questions as actual duplicate is a manual (mostly moderator or experienced users) task. The manual nature of this task leads to the consumption of a long time (with respect to the speed of knowledge exchange in the community) for a pair of questions to get confirmed as duplicate since it was the first identified. For instance, as per the askubuntu policy, at least five votes are needed to confirm that a pair of questions are duplicates. Typically, these votes get accrued over a long period of time and increase the time to closure. In this work, we attempt to retrieve possible duplicate questions for a newly posted question. Further, we attempt to direct the moderator's attention toward marked duplicate pairs that could have got identified (confirmed) in longer than usual time. We will use queries and questions interchangeably in the following sections.

**Duplicate question retrieval:** In this task, for each new query, we shall attempt to recommend the top  $k$  possible duplicates to the users so that they have the option to choose a similar query from the previously posted questions. When a new user posts a repetitive question, a moderator should be able to quickly find the possible duplicates from the earlier queries. An example of redundant question is noted in Table I.

**Duplicate confirmation time prediction:** It is observed that after a pair of queries are initially marked as a possible duplicate, it takes a long time for them to acquire enough votes to be eventually confirmed as duplicates. In askubuntu, as per our analysis, there are around  $\sim 40\%$  question pairs that take more than five days to get confirmed as duplicates.

Also, out of all these pairs, 50-55% have a high view count of 1000 – 10,000 thus showing that they engage a lot of users. Our task is to identify those pairs which took a long time to be confirmed as duplicates. We intend to get a rank list of the pairs according to their *time taken* in decreasing order. Such pairs will be explicitly suggested to the moderators for more attention. An example of duplicate question pairs and their duplicate confirmation timestamp is noted in Table I. To

Question A	Question B	DCT
<b>Title:</b> How to remove WUBI installed Ubuntu without affecting Windows files?	<b>Title:</b> How do I uninstall Ubuntu Wubi?	2013-02-18 03:03:21
<b>Body:</b> ...Now I want to remove Ubuntu from my laptop without affecting my Windows files ...	<b>Body:</b> I want to uninstall Ubuntu because I just don't like it... I have windows 7 ...	
<b>Posting time:</b> 2012-11-07 13:35:45	<b>Posting time:</b> 2012-05-30 18:58:11	

TABLE I

Duplicate confirmation timestamp. Example of a pair of duplicate questions with the title, body and posting timestamp. The duplicate link formation is also given.

the best of our knowledge, the first problem, i.e., duplicate question retrieval, has been treated as a classification task [3], [21], [28] or as a recommendation task using a classification/regression objective function [1], [32]. However, such a scheme cannot be easily deployed in a real-time scenario given a large corpus. In this work, we treat this problem as a recommendation task and propose a method based on Siamese neural network [6] to solve the problem. In addition, we use the node embedding obtained from the tag co-occurrence network as the representation of a tag in order to enrich our model. Further, we compare the state-of-the-art methods [1], [25], [28], [33] with our approach. The second problem, i.e., duplicate confirmation time prediction, has not been attempted in literature for any platform to the best of our knowledge. However, this problem is important when the system already has a lot of unconfirmed duplicate pairs.

#### Our contribution and results:

**Duplicate question retrieval:** We propose a simple method for retrieving actual duplicates from the candidates of a given question. We compare our approach with various state-of-the-art baselines. First, we use question title and body text representation as features. Further, including tag representations from the tag co-occurrence network increases the overall performance. Using text features, we obtain an MRR of 9.45%, considering a list of 485 duplicates with an average candidate set size of 5941. In addition, the recall rate  $RR@10$  is 15.88%. The inclusion of network features brings additional benefits, which leads the MRR and  $RR@10$  to rise to 11.10% and 18.35% (considering a list of 485 duplicates with an average candidate set of size 5941), respectively. Our model's uniqueness lies in tackling this problem by *not* using a conventional classification objective function [1], [32] and how we sample the negative examples and select the candidate set. Further, including features from the tag co-occurrence network along with textual features helps to considerably outperform the baseline approaches.

**Duplicate confirmation time prediction:** We model this problem as a regression task where the input is the text representation of a question (aka text), and the output is a probability

ranking of questions (based on the time required to close a question as duplicates from the time they have been first identified as being duplicates). Including tag representations obtained from the tag co-occurrence network as additional features (aka text+network) further improves the performance. MLP-based models achieve the best Spearman's rank correlation of 0.208 (text) and 0.213 (text+network), respectively, considering the complete rank list of 3756 duplicates. Adding network features always shows improvement, and these results are statistically significant. While we perform our experiments on the askubuntu platform, we would like to highlight that our methods are generic and can be extended to any other platform.

## II. RELATED WORK

**Duplicate question retrieval:** Duplicate detection is one of the major problems in various large systems since the growth of Internet usage. Duplicate detection has been an important problem in databases [9], [30], webs [29], bug tracking systems [2], [12], [26], [27], and community question answering systems [1], [23], [31], [33]. Zhang *et al* [33] proposed a novel method called DupPredictor to identify possible duplicates of a new question by considering various factors. The authors in [31] proposed a classification method for duplicate question detection on StackOverflow<sup>1</sup>. For a pair of questions, they obtain the features from word2vec, topic modelling and phrase pairs that co-occur in duplicate questions. In [3] the authors used standard machine learning models such as support vector machine and convolutional neural network to identify semantically similar questions in an online forum. The authors in [16] used Siamese-LSTM [20] along with dense layer and classifier to detect semantically equivalent question pairs in Quora. In [14] the authors used Siamese GRU network to detect the semantically equivalent question pairs in Quora. Finally, the authors in [19] proposed a Siamese network based method for detecting duplicate questions in StackExchange data. Further, they employed domain adaptation with transfer learning to improve performance<sup>2</sup>.

**Identifying duplicate question time:** Confirming a pair of a question as a duplicate within the tangible time frame is a challenging task. Less or no earlier work is present where this problem has been addressed. In [1], while characterizing the same questions in the StackOverflow platform, the authors have analyzed the time taken to close a question as duplicate. Our work is unique in different ways. First, we have considered the latest dump of a popular CQA platform – askubuntu – vital for the software development community. While our model is simple, the main novelty lies in how we perform negative sampling and candidate set selection for duplicate retrieval. Further, we conceive of a novel tag co-occurrence network that brings additional performance boosts for both tasks.

<sup>1</sup>The results could not be reproduced due to lack of requisite information about experimental setup and feature calculation.

<sup>2</sup>In [15], although the authors used Siamese neural networks, the duplicate pairs for testing are predefined and thus cannot be used as an additional baseline.

### III. DATASET

In this paper, we use the community question-answering platform askubuntu data dump released at the beginning of 2021. The data dump consists of  $\sim 366K$  questions and textual information such as question title, question body, and corresponding answers. Question metadata includes question reporting time, question tags, answer posting time, question reporting user, users who posted the answers, and duplicate link formation timestamp. The primary contents of the dataset are noted in Table II. For our experiment, we have chosen askubuntu because it is based on a single ecosystem (ubuntu ecosystem) and contains large volumes of duplicates. Further, moderators on these platforms are experts who are usually very busy with their time being extraordinarily expensive. In previous papers, certain question groups (Java, C++, Python, Ruby, HTML, and objective-C) [1] or older repo (contains 1641 duplicates only) [33] of StackOverflow and StackExchange has been used for duplicate detection. The authors of paper [4] used the askubuntu data for detecting schematically equivalent questions.

Information	Count
Total number of questions	366,324
Average number of words in body	139.62
Average number of words in title	8.51
Average number of tags	2.77
Total number of tags	3118
#duplicate question pairs	68286

TABLE II  
DATASET STATISTICS.

In this paper, we use the community question-answering platform askubuntu data dump released at the beginning of 2021. The data dump consists of  $\sim 366K$  questions and textual information such as question title, question body, and corresponding answers. Question metadata includes question reporting time, question tags, answer posting time, question reporting user, users who posted the answers, and duplicate link formation timestamp. The primary contents of the dataset are noted in Table II. Note that we have the tags associated with the questions. Like any other platform, these tags attempt to topically organize the questions to facilitate a better search. An inspection of these tags across the duplicate questions show that they are primarily on system configuration requirements for Ubuntu installation, driver installation, new package installation, and suitable Ubuntu distribution according to the hardware configuration and basic Linux commands.

In order to extract meaningful information from these tags and their relationships we construct a tag co-occurrence network where the tags are the nodes and two nodes are connected if they co-occur in a question. We compute the Jaccard overlap of question sets to which two tags  $t_1, t_2$  are common which defines the weight of the edge. Edges with a weight larger than 0.005 are only retained<sup>3</sup>.

Another essential parameter in our data is the duplicate confirmation time. We assume the pairs are marked as soon as the

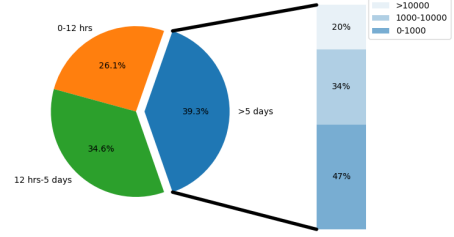


Fig. 1. The pie chart represents the fraction of marked duplicate pairs that have been confirmed as duplicates after a particular time. The corresponding bar shows the view count of the latest question in the pair which took more than 5 days to get confirmed.

recent most question of the pair has been posted. We observe that there are  $\sim 40\%$  question pairs that require more than 5 days to become confirmed as duplicates (see Fig. 1). Around 20% of these pairs are viewed by more than 10000 users and 30-35% of these pairs are viewed by 1000-10000 users showing high levels of user engagement thus necessitating the prediction of duplicate question pair confirmation time.

For our experiment, we have chosen askubuntu because it is based on a single ecosystem (ubuntu ecosystem) and contains large volumes of duplicates. Further, moderators on these platforms are experts who are usually very busy with their time being extraordinarily expensive.

### IV. NOTATION AND PRELIMINARIES

We have a set of  $Q$  questions in a CQA ecosystem indexed as  $q \in [Q] = [1 \dots Q]$  where  $q$  represents a single question. Each question  $q$  is associated with the reporting timestamp  $ts(q)$ . There is a set of tags  $\mathcal{T}$  indexed by  $t \in [\mathcal{T}] = [1 \dots \mathcal{T}]$ . Given a question  $q$ , there is a set of associated tags  $T_q \subset \mathcal{T}$ . Each question  $q$  has three important features – the title of the question denoted by  $QT$ , the body of the question denoted by  $QB$ , and the tags of the question denoted by  $T_q$ . We have a set of duplicate pairs of questions  $(q_1, q_2)$  where  $q_1, q_2 \in Q$ . We assume that the latest question within the duplicate pair is an anchor question  $q_1$  and the older question as its master (usually positive pair) denoted by  $q_2$ . In specific, for a duplicate pair  $(q_1, q_2)$ ,  $q_1$  is an anchor if  $ts(q_1) > ts(q_2)$ ; else  $q_2$  is the anchor. We denote the time when the question was confirmed by  $ts(q_1, q_2)$  where  $(q_1, q_2)$  is a duplicate pair. This is also called the time of duplicate link formation. The tag co-occurrence network is denoted as  $G$  where nodes are the tags and edges have weights as already defined earlier.

### V. DUPLICATE QUESTION RETRIEVAL

Suppose we have a set of questions  $Q$  and graph  $G$  (tag co-occurrence network). Given a pair of questions  $(q_1, q_2)$ , for  $q_1$  (anchor question), our task is to find out its duplicate question  $q_2$ . In the subsequent sections, we will denote the anchor question as  $q_a$ , its actual duplicate question as  $q^+$ , and

<sup>3</sup>We set this threshold based on manual inspection of the data.

other questions which are not duplicates will be denoted by  $q^-$ . Given anchor question  $q_a$ , we intend to rank the possible duplicate questions according to decreasing order of duplicity scores. The position of the gold duplicate  $q^+$  can be found in this rank list and we evaluate the system’s performance using Mean Reciprocal Rank (MRR) and recall rate at  $k$  (RR@ $k$ ). In subsequent paragraphs, we describe the strategy for sampling  $q^-$  and building the candidate set.

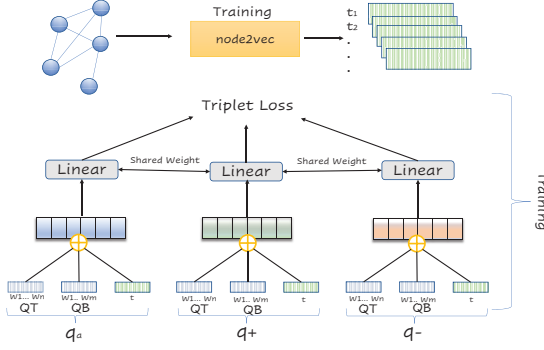


Fig. 2. Training phase.

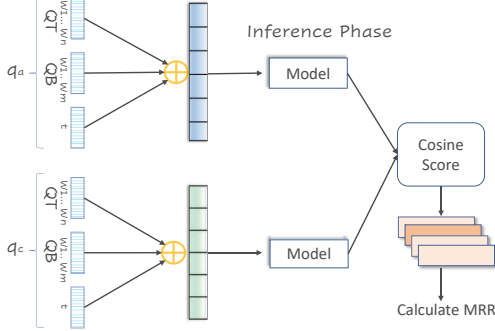


Fig. 3. Inference phase.

**Model architecture:** Our model consists of a transformer encoder with mean pooling for embedding generation followed by concatenation and linear transformation with activation. Going forward in this paper, we denote our method using **TE**: transformer encoder. We have used a Siamese neural network [6] for solving this problem. The pipeline of the proposed model is presented in Figure 2.

**Text embedding:** We use the representation of the title and body of the questions as features. As part of preprocessing these two pieces of text, we removed the URLs, stopwords, etc. We have used transformer encoders with mean pooling to generate the embeddings. For a given question  $q$ , the embeddings for the title and body are denoted as  $e_q^{QT}$  and  $e_q^{QB}$  respectively.

**Graph embedding:** We have used the tag co-occurrence network to compute tag features. These features are blended with the text to obtain the final embedding. We train the tag co-occurrence network using the node2vec [10] algorithm. Here, we cannot train any graph neural network to obtain the embeddings of the node because we do not have any specific

target variable<sup>4</sup>. So, we did not go forward with this setup. We have ordered the tags based on their occurrence in training data. For every question, we have an ordered list of tags; from this list, we take only the embedding of the top tag ( $e_q^t$ ) to be blended with the text.

**Concatenation and linear transformation:** In this step given a question  $q$ , we concatenate the feature vectors  $e_q^{QT}$  and  $e_q^{QB}$  for text-based model. For text+network based model, we concatenate the feature vectors  $e_q^{QT}$ ,  $e_q^{QB}$  and  $e_q^t$ . Concatenation is denoted by  $\oplus$ .

$$E_q^{TB} = [e_q^{QT} \oplus e_q^{QB}], E_q'^{TB} = \sigma(W_{TB} \cdot E_q^{TB} + b_{TB}) \quad (1)$$

**Objective function:** Given an anchor question  $q_a$ , positive question  $q^+$  and negative question  $q^-$ , the triplet margin loss<sup>5</sup> tune the model ( $\theta$ ) in such a way that the distance between  $\theta(q_a)$  and  $\theta(q^+)$  will decrease but the distance between  $\theta(q_a)$  and  $\theta(q^-)$  will increase. We assume that we have got  $E_{q_a}^{TB}$ ,  $E_{q^+}^{TB}$ ,  $E_{q^-}^{TB}$  representation from the model  $\theta$  for anchor question  $q_a$ , positive question  $q^+$  and negative question  $q^-$  respectively. The loss function is as follows -  $L(\theta) = \max[d(E_{q_a}^{TB}, E_{q^+}^{TB}) - d(E_{q_a}^{TB}, E_{q^-}^{TB}), 0]$ . Here the  $d(E_{q_a}^{TB}, E_{q^+}^{TB}) = \|E_{q_a}^{TB} - E_{q^+}^{TB}\|_p$ .  $p$  is the norm degree of the pairwise distance.

#### Algorithm 1 Negative sampling strategy

---

Buckets  $b_i \in \mathcal{B}$ , similarity matrix  $\mathcal{B}[b_1, b_2]$  between the representation of the buckets  $b_1$  and  $b_2$ ;

**for** each positive pair  $(q_a, q^+)$  **do**

  create an empty dictionary  $D_{J, q_a}$ ;

  get the bucket of  $b_a \in \mathcal{B}$  to which  $q_a$  belongs;

  obtain an ordered list (high to low) of buckets  $\mathcal{B}_k \setminus b_a$  where the ordering is based on the similarity  $\mathcal{B}[b_a, b_k] \wedge \mathcal{B}[b_a, b_k] > \alpha$ ;

**for** each bucket  $b_k \in \mathcal{B}_k$  **do**

**for** each  $q \in b_k$  **do**

      calculate tag overlap  $T_{overlap}(q_a, q)$ ;

**if**  $q$  is answered **then**

        append item  $\{q : T_{overlap}(q_a, q)\}$  to the dictionary  $D_{J, q_a}$ ;

**end if**

**end for**

**end for**

  choose the  $q$  with maximum  $T_{overlap}(q_a, q)$  from  $D_{J, q_a}$ ;

$q \leftarrow q^-$ ;

**end for**

---

**Negative sampling strategy:** In the training, we prepare triplets consisting of  $(q_a, q^+, q^-)$ . This section discusses how we sample the  $q^-$  for every duplicate pair present in the training data. We obtain buckets based on the duplicate clusters. Suppose there are four duplicate pairs  $(q_1, q_2)$ ,  $(q_1, q_3)$ ,  $(q_3, q_4)$  and  $(q_5, q_6)$ . As the pairs  $(q_1, q_2)$ ,  $(q_1, q_3)$ ,  $(q_3, q_4)$  have transitive relationship, they would form a bucket whereas  $(q_5, q_6)$  is not transitive with them so it would form another bucket. We form the buckets out of all the duplicate pairs in the training set. We next obtain a representation of a bucket as average embeddings of all the questions in that bucket. Based on this representation, we compute the bucket-bucket similarity matrix  $\mathcal{B}$ . We then sample buckets most similar to the bucket containing  $q_a$ . Out of the questions in these similar buckets, we choose the one with the highest tag overlap with  $q_a$  as  $q^-$ . The heuristic attempts to identify one of the hardest negative samples so that the decision boundary is robust. The detailed steps are given in Algorithm 1.

<sup>4</sup>The unsupervised approach suitable for graph neural network also did not perform well.

<sup>5</sup>[https://en.wikipedia.org/wiki/Triplet\\_loss](https://en.wikipedia.org/wiki/Triplet_loss)

**Inference:** Given an anchor question, we obtain the similarity scores for the possible duplicate questions with the anchor and rank them during the inference. Before getting the scores, we must prepare a set of possible duplicate questions for a given anchor question. Given an anchor question  $q_a$ , we call this set a candidate set ( $Q_a^c$ ). The detailed inference phase is presented in Figure 3. This Figure shows the process of getting similarity scores between  $q_a$  and  $q_c \in Q_a^c$  from the model. The following section discusses the strategies used for generating the candidate set  $Q_a^c$  for all the anchor questions.

**Candidate set generation:** We construct a candidate set  $Q_a^c$  for each anchor question  $q_a$  using the following selection heuristic - (i) the extent of tag similarity between anchor and candidate question, (ii) if the candidate question has an answer (either accepted or unaccepted), and (iii) the question title ( $QT$ ) similarity between the anchor and the candidate question. Our intuition is that most duplicate pairs have tags in common and a similarity in their question title.

Let us denote the tag list of the anchor question as  $T_{q_a}$ . Further for each anchor question  $q_a$ , we create an empty candidate set  $Q_a^c$ . We collect the previously posted questions (strictly earlier to the anchor question), which have at least one of the tags common with  $T_{q_a}$ . Let us call this question set as  $Q^c$  and the tag list of each  $q^c \in Q^c$  as  $T_{q^c}$ . Next, we calculate the Jaccard overlap ( $J(T_{q_a}, T_{q^c})$ ) between  $T_{q_a}$  and  $T_{q^c}$  for each  $q^c$ . We retain only those questions in  $Q^c$  which have  $J(T_{q_a}, T_{q^c}) > 0.15$ . We further filter  $Q^c$  to have only those questions that have been already answered. Finally, we proceed with the last filter retaining only those questions in  $Q^c$  whose embeddings have a cosine similarity of 0.27 or more with the embedding of the anchor question. We populate  $Q_a^c$  with the final set of questions present in  $Q^c$ .

## VI. DUPLICATE CONFIRMATION TIME PREDICTION

Suppose we have a question and its textual data such as  $QT$ ,  $QB$ . Our goal is to predict the time gap  $ts_{Gap}$  between the time when the recent most question in the pair is reported as a duplicate and the duplicate link formation time (i.e., when the question is closed as a duplicate). Thus, given a question pair  $(q_1, q_2)$ , we have computed the time gap  $ts_{Gap}(q_1, q_2)$  as  $(ts(q_1, q_2) - ts(q_1))$  where  $ts(q_1) > ts(q_2)$ . We want to predict the time gap  $ts_{Gap}$  for all the possible duplicate pairs. We sort the  $|ts_{Gap}|$  in descending order and focus on the pairs that took a long time to close as duplicates. The idea is to present the top-ranked pairs with a long time gap to the moderators so that it could be addressed quickly.

Note that our assumption here is that the recently posted question has already been marked as a duplicate of some earlier question by some user (regular user/moderator) but has yet to receive the necessary attention<sup>6</sup> from the moderators or anyone having more than 3K reputation. Unless the recently posted question gets a certain number of votes (usually takes a minimum of 5 votes) from the moderators/experienced users, the question is not considered a duplicate of the earlier

question (i.e., the question link formation cannot take place). Our idea is to early predict those pairs which have remained “open” for a long (i.e.,  $ts_{Gap}$  is large) and facilitate their closing by bringing them to the notice of the moderators. We have the gold  $ts(q_1, q_2)$  during the evaluation, and we compare the gold ranks with the predicted ranks using rank correlation methods.

**Text and graph embedding:** These are generated exactly the same as in case of duplicate question retrieval.

**Model architecture:** We use two models for this task – (i) Decision Tree (DT), (ii) XGBoost (XGB) and (iii) Multi-layer perceptron (MLP).

We use the standard DT regressor [5] and XGBoost regressor [7] with inputs as (i) text and (ii) text+ network features. The output is a regression score with duplicate pairs requiring the largest time to close.

In case of MLP, we use L1 loss<sup>7</sup> between the predicted time gap ( $ts'_{Gap}$ ) and the gold time gap ( $ts_{Gap}$ ). The model architecture is summarized by equations 2, 3 and 4. Given a pair of questions  $(q_1, q_2)$ , we use the same model for both features.

$$\begin{aligned} E_{q_1}^{TB} &= ReLU(W_1^{TB} \cdot [e_{q_1}^{QT} \oplus e_{q_1}^{QB}] + b_1^{TB}) \\ E_{q_2}^{TB} &= ReLU(W_2^{TB} \cdot [e_{q_2}^{QT} \oplus e_{q_2}^{QB}] + b_2^{TB}) \end{aligned} \quad (2)$$

$$\begin{aligned} E_{q_1}'^{TB} &= ReLU(W_1'^{TB} \cdot E_{q_1}^{TB} + b_1'^{TB}) \\ E_{q_2}'^{TB} &= ReLU(W_2'^{TB} \cdot E_{q_2}^{TB} + b_2'^{TB}) \end{aligned} \quad (3)$$

$$ts'_{Gap} = TanhShrink(W_{12}'^{TB} \cdot [E_{q_1}'^{TB} \oplus E_{q_2}'^{TB}] + b_{12}'^{TB}) \quad (4)$$

In the case of text+network features, we change the input embedding, i.e., instead of  $[e_{q_1}^{QT}, \oplus e_{q_1}^{QB}]$ , we pass  $[e_{q_1}^{QT} \oplus e_{q_1}^{QB} \oplus e_{q_1}^t]$  as an input.  $W_1^{TB}, W_2^{TB}, W_1'^{TB}, W_2'^{TB}, W_{12}'^{TB}$  are the trainable weights.  $b_1^{TB}, b_2^{TB}, b_1'^{TB}, b_2'^{TB}$  and  $b_{12}'^{TB}$  are the trainable biases. In the last layer, we use TanhShrink<sup>8</sup> because of the span of the data where the lowest target is negative and the highest target value is positive.

## VII. EXPERIMENTS AND RESULTS

### A. Duplicate question retrieval

**Upper bound:** To calculate the upper bound<sup>9</sup>, given an anchor question, we identified whether the actual duplicate is present in the candidate set or not. If it is present in the candidate set, we consider the rank 1; otherwise, 0. For the test data, we obtained an upper bound of 62.8%. Thus this is the best possible recall that can be achieved.

**Text features:** We use InferSent [8], BM25 [25], Glove + BiLSTM [13], [22], word2vec + BiLSTM [28], word2vec [18] (word2vec algorithm directly trained on our CQA corpus) and doc2vec [17] (doc2vec algorithm directly trained on our CQA corpus) to generate text embeddings. All the hyperparameters used in these baselines are obtained through grid search and are noted in Table V.

<sup>7</sup><https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html>

<sup>8</sup><https://pytorch.org/docs/stable/generated/torch.nn.Tanhshrink.html>

<sup>9</sup>This term is adapted from [11]

<sup>6</sup><https://askubuntu.com/help/duplicates>

*Network features:* While both the word2vec and doc2vec models discussed above are text only, here we add the network features obtained from the node2vec embeddings. The title, body, and tag embeddings are fed to the MLP layer (Figure 2).

**Experimental setup for our method (TE):** We divide all the questions into three parts – training, validation, and test. In training, we consider the duplicate pairs closed between 2010 to 2018, whereas, for the validation set, we use the last three months’ data from 2019. For testing, we use the last three months’ data from 2020. Since we follow a retrieval-like evaluation, we need to compare anchor questions with every candidate question in the candidate set. Thus the total number of comparisons being relatively high, we have chosen only three months of data for validation and testing. We have a total number of  $\sim 32K$  positive pairs in the training set. In the inference phase, based on the candidate set generation heuristic, the average number of questions in a candidate set is 5941<sup>10</sup> for test data. In our test data, we have 485 anchors, thus making the total number of comparisons equal to almost 2.8 million ( $485 \times 5941$ ).

*Specifications of the text embedding generation:* We have used *multi-qa-MiniLM-L6-cos-v1*<sup>11</sup> pretrained model to generate the embeddings of the *QT* and *QB*. The default embedding dimension is 384. This model uses the pretrained setup with 6 layer version of Microsoft/MiniLM-L12-H384-uncased by keeping only every second layer<sup>12</sup>.

*Specifications of the network embedding generation:* We investigate different values of the parameters for training node2vec through grid search and populate 64-dimensional embedding. We got  $p$  as 1.3,  $q$  as 0.8, the number of the walk as 5, the walk length as 80, *min\_count* as 3, *batch\_word* as 5, and parameter *window* to 10.

*Hyperparameters:* For the hyperparameter tuning of the text-only models, we have found the learning rate as  $1e-3$  and  $\epsilon$  as  $1e-8$ . The output size of the representation is 512 and, the number of epochs is 40.

*Evaluation metrics:* We use the mean reciprocal rank (MRR) and recall rate ( $RR@k$ ) to evaluate all the models. We have used different values of  $k$  ranging from 10 – 500. Since the candidate set size is  $\sim 5K$ ,  $RR@500$  is expected to present good suggestions to the moderators for duplicate question closure, reducing the otherwise tremendous manual load. Note that the evaluation results presented here are only for those anchor questions that have duplicates.

**Baselines:** We use nine different baseline methods. **InferSent** [8]: Infsent is a sentence encoder where the representation of each sentence has been computed. It is a BiLSTM network with max pooling. We compute embeddings for each question and subsequently obtain the cosine similarity between

the anchor question and its candidates.

**BM25 Search** [25]: We use the standard unsupervised method for BM25 search. Here, we provide the title and body of the questions to train the BM25 model. Further, for each query, we obtain scores of its candidates and rank the candidates based on the scores (higher score corresponding to better rank).

**Glove + BiLSTM** [13], [22]: For each question, we extract the 300d representation of words and further use BiLSTM and one linear layer to obtain the final representation. Given a triplet of questions during the training, we compute the triplet loss between anchor, positive and negative questions. During the evaluation, we use cosine similarity.

**word2vec + BiLSTM** [28]: In this case, we use the pretrained ‘Google News-vectors-negative300’ model to obtain the embedding of the words present in the texts of each question. Due to the Siamese-like architecture, we do not use the sigmoid activation mentioned in their architecture. The rest of the architecture is verbatim similar. During training, we feed the embedding of a sequence of words to BiLSTM and one linear layer to get the final fixed-length representation for a question. Here, we use triplet loss. During the evaluation, we use cosine similarity to rank the candidates.

**word2vec** [18]: Everything else remaining the same as the **TE** model, we replace the transformer encoders with word2vec to generate the question title and body embeddings. The word2vec embeddings are obtained by training the word2vec algorithm from scratch using the entire CQA corpus. All parameters were identified through a grid search.

**doc2vec** [17]: We replace the transformer encoder with the trained doc2vec to generate question titles and body embeddings. We train the doc2vec model using the whole CQA corpus. Further, we obtain the question title and body embeddings from the trained doc2vec models.

**DupPredictor** [33]: We implement DupPredictor algorithm and test it on our dataset. We create four components – title similarity component, question body similarity component, topic similarity component, and tag similarity component. For topic modeling, we train the LDA model on the whole corpus (concatenating the title and the body of a question). The number of topics is 100.

**Dupe** [1]: We implement the Dupe method for our dataset. In their paper, they concluded that title, body, tag, title-body, body-title, title-tag, code similarity features are contributing to the best performance. So, compute these features on our dataset. We use the logistic regression as mentioned in the paper.

**SBERT STSb models** [24]: We use two pretrained models for obtaining the title and the body embedding of the question. The pretrained models are distilbert-base-nli-stsb-quora-ranking and distilbert-multilingual-nli-stsb-quora-ranking. Further, we feed the embedding to our MLP model.

To compare our model with the existing methods, we treat the models in the Siamese network setup. All the hyperparameters used in these baselines are obtained through grid search and are noted in Table V.

**Results:** We make a few observations from the results pre-

<sup>10</sup>Without the candidate set generation strategy, the number of earlier questions to which the anchor question would have to be compared would be close to  $\sim 300K$ .

<sup>11</sup><https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-cos-v1>

<sup>12</sup><https://huggingface.co/nreimers/MiniLM-L6-H384-uncased>

$q_a$	Title of $q_a$	$q^+$	Title of $q^+$	PR (TE)	PR (TE+net)	N	#w(T)
359751	Prefix argument for starting chromium with hardware acceleration	128126	How to execute a command with "=" sign in a desktop shortcut?	55	89	28	8
364255	Running, or "injecting" software with specific date	250575	Change Ubuntu time and date for specific application	2	3	16	10
363710	How to change Ubuntu 20.04 Desktop file manager (not gnome)?	338041	How to remove GNOME Shell from Ubuntu 20.04 LTS to install other desktop environment from scratch?	62	147	26	13
364236	how I would make Ubuntu GUI in wsl subsystem in Window	262015	What's the easiest way to run GUI apps on Windows Subsystem for Linux as of 2018?	4	10	31	11
364973	Why do I have to use sudo if I am the only user?	245098	What's exactly the point of the sudo command, in terms of security?	8	13	36	14

$q_a$	Title of $q_a$	$q^+$	Title of $q^+$	PR (TE)	PR (TE+net)	N	#w(T)
363584	t440 ubuntu drivers	140121	How to download all required Ubuntu drivers	45	39	89	3
362424	Cannot Update packages	3491	How do I fix the GPG error "NO_PUBKEY"?	4868	1363	49	4
363917	Remove plasma from ubuntu	187651	How to remove KDE Plasma-Desktop?	4	3	68	4
359502	Ubuntu Live USB boot problem	45554	My computer boots to a black screen, what options do I have to fix it?	3621	59	68	5
365800	AMD drivers Ubuntu 20.04.1	210683	Ubuntu 14.04.5/16.04 and newer on AMD graphics	80	43	54	4

TABLE III

$q_a$ : ANCHOR QUESTION,  $q^+$ : ACTUAL DUPLICATE, PR(TE): PREDICTED RANK OF **TE**, PR(TE+NET): PREDICTED RANK OF **TE+NETWORK**, N: #NEIGHBORS, #w(T): #WORDS IN TITLE. TEST EXAMPLES WHERE (UP) THE **TE** MODEL PREDICTS BETTER RANKS OF ACTUAL DUPLICATE QUESTION THAN THE **TE+NETWORK** MODEL, (DOWN) **TE+NETWORK** MODEL PREDICTS BETTER RANKS OF ACTUAL DUPLICATE QUESTION THAN **TE** MODEL.

Methods	MRR	RR@10	RR@20	RR@30	RR@50	RR@100	RR@500
<b>Text only</b>							
<b>word2vec</b>	4.980	7.628	10.309	13.814	17.319	21.855	39.175
<b>doc2vec</b>	0.840	1.440	2.061	2.061	4.536	9.278	23.505
<b>Pretrained word2vec + BiLSTM</b>	2.299	3.505	4.123	5.154	6.391	8.247	18.556
<b>Glove + BiLSTM</b>	1.403	2.474	3.711	4.123	4.742	6.597	15.463
<b>BM25 Search</b>	6.060	10.300	13.190	14.840	17.730	24.740	37.930
<b>InferSent</b>	3.200	4.120	6.180	7.210	8.650	11.340	22.680
<b>DupPredictor</b>	4.560	10.100	12.780	15.250	17.310	21.850	35.870
<b>DUPE</b>	2.750	3.910	5.360	7.210	9.480	12.780	24.740
<b>TE</b>	<b>9.452</b>	<b>15.876</b>	<b>19.381</b>	<b>21.649</b>	<b>25.154</b>	<b>31.546</b>	<b>44.948</b>
<b>Text+network</b>							
<b>word2vec + network</b>	4.980	8.453	12.371	14.226	17.113	22.680	40.618
<b>doc2vec + network</b>	0.740	1.649	2.886	3.298	5.154	8.041	23.711
<b>SBERT STSb distillbert + network</b>	4.190	10.220	12.710	15.080	18.100	24.200	38.770
<b>SBERT STSb distillbert multilingual + network</b>	3.490	7.180	10.820	13.190	15.400	19.880	33.000
<b>TE+network</b>	<b>11.088*</b>	<b>18.350</b>	<b>23.917</b>	<b>27.010</b>	<b>32.164</b>	<b>36.082</b>	<b>46.597</b>

TABLE IV

DUPLICATE QUESTION RETRIEVAL. ALL THE RESULTS ARE SHOWN IN PERCENTAGES. TE: TRANSFORMER ENCODER, \*: THE RESULT OF THE TEXT+NETWORK MODEL IS SIGNIFICANTLY DIFFERENT ( $p < 0.03$  USING M-W U TEST) FROM THE TEXT-ONLY MODEL.

sented in Table IV. Our method based on a transformer encoder (**TE**) outperforms all the other approaches in text-based settings. We present the results for MRR and RR@{10, 20, 30, 50, 100, 500}. The table shows that our proposed technique performs better than all baselines, including state-of-the-art **DUPE** and **DupPredictor**. Also, we observe exciting performance improvement in several other popular baselines. For example, **BM25 search** achieves an MRR score of 6.06 whereas our method achieves an MRR score of 9.452 (an increase of almost 3.5%). Similarly, we also observe an increase of nearly 5%, 6%, 8%, and 7% of RR values at RR@10, RR@20, RR@50, and RR@50 respectively, for our method (**TE**) over **BM25 search**, which secures a second position

across the different baselines. At RR@30 **DupPredictor** performs better than **BM25 search**, but our method outperforms **DupPredictor** by almost 6%. We note an improvement for the case of word2vec at RR@500 when compared to all other baselines; however, our technique outperforms word2vec by over 5%. So, with all types of evaluation metrics, our proposed method (**TE**) consistently achieves better results than all other baselines.

We also observe a similar trend in the **text+network** model. Here we see an increase in MRR score when compared with **word2vec + network**. Similarly we achieve 18.35% in RR@10, 23.917% in RR@20, 15.08% in RR@30, 18.1% in RR@50 and 36.082% in RR@100 which outperforms



Method	Hyperparameters
InferSent	version-1, word embedding dimension = 300, LSTM dimension = 2048, pooling = 'max'.
BM25	$b = 0.75$ and $k1 = 1.5$
Glove + BiLSTM	learning rate = $1e-5$ , batch size = 64, #words = 256, BiLSTM hidden dimension & final output dimension = 64.
word2vec + BiLSTM	learning rate = $1e-4$ , batch size = 64, BiLSTM hidden dimension & final output dimension = 64.
word2vec	window size = 10, #epochs = 5, learning rate = 0.025, final representation = averaged over all words, MLP layer (Figure 2) parameters: #epochs = 40, $\epsilon = 1e-8$ , learning rate = $1e-3$ , output dimension = 512 (2048 for word2vec+network).
doc2vec	All parameters same as word2vec except for the output dimension in the MLP layer = 2048 (1024 for word2vec+network.)
DupPredictor	$\alpha = 0.8$ , $\beta = 0.8$ , $\gamma = 0.1$ and $\delta = 0.6$
Dupe	batch size = 32, learning rate = $1.00E-05$ , number of epochs = 40
SBERT STSb distill-bert base	#epochs = 30, $\epsilon = 1e-7$ , learning rate = $1e-4$
SBERT STSb distill-bert multilingual	#epochs = 20, $\epsilon = 1e-7$ , learning rate = $1e-4$

TABLE V

HYPERPARAMETERS FOR THE BASELINE METHODS CHOSEN BASED ON GRID SEARCH.

its nearest competitor **SBERT STSb distilbert + network** with increase of almost 8% in RR@10, 11% in RR@20, 12% in RR@30, 14% in RR@50 and 12% in RR@100 respectively. We observe our proposed method **TE+network** reaches 46.597% for RR@500, which outperforms **word2vec+network** with a margin of almost 6%.

### B. Duplicate confirmation time prediction

**Experimental setup:** We have considered all the duplicate pairs present in the dataset in this setup. Further, we divide the dataset into train, validation, and test sets. For training, we have considered pairs of questions where all the questions were posted before 2020. Further, we use 25% of this training set for validation. For testing, we have considered all the pairs where the questions were posted after 2020. We have considered the time gap in hours. In specific, we predict  $\log_{10}(ts_{Gap})$  using two models – (i) Decision Tree (DT) and (ii) XGBoost (XGB) (iii) Multilayer perceptron (MLP).

Here again, we have used *multi-qa-MiniLM-L6-cos-v1* pre-trained model to generate the embeddings of  $QT$  and  $QB$ . The embedding dimensions for each of them are 384. To get the node embeddings from the tag co-occurrence network, we have used the node2vec [10] algorithm. For training, the same parameters noted in section VII-A have been used.

**Settings for the DT model:** For the DT model, after the parameter tuning, the criterion is set to squared error, splitter is set to 'best', max-depth is set to 7, and min\_samples\_split is set to 2. For the text only model, we concatenate the title and the body embeddings of a question  $q_i$  to obtain a 768 dimensional embedding –  $[e_{q_i}^{QT} \oplus e_{q_i}^{QB}]$ . For a pair  $(q_1, q_2)$  we feed the DT with  $[e_{q_1}^{QT} \oplus e_{q_1}^{QB} \oplus e_{q_2}^{QT} \oplus e_{q_2}^{QB}]$  as the feature. For the text+network model we feed the DT with  $[e_{q_1}^{QT} \oplus e_{q_1}^{QB} \oplus e_{q_1}^t \oplus e_{q_2}^{QT} \oplus e_{q_2}^{QB} \oplus e_{q_2}^t]$  as the feature where  $e_{q_i}^t$  represents the 64 dimensional embedding of the top tag of  $q_i$  obtained from the tag co-occurrence network.

**Settings for XGB model:** We have used the same setup as the DT model for text and text+network-based models. After tuning the parameters, the *n\_estimator*, *max\_depth* are kept as 1000 and 7, respectively. The value of *eta*, *subsample* and *colsample\_bytree* are set to 0.1, 0.7 and 0.8 respectively.

**Settings for the MLP model:** As in the DT setting, here also

we use the same  $[e_{q_i}^{QT} \oplus e_{q_i}^{QB}]$  embedding to represent a question in the text-only model. For a given a pair  $(q_1, q_2)$ , we pass the corresponding 768 dimensional representations of  $q_1$  and  $q_1$  to the input layer. The intermediate hidden layer is set to 256 and 64. The final output layer size is set to 1. We found the batch size is 64 and the learning rate is  $2e-5$ . For the text+network model, each question is again an 832 (384+384+64) dimensional embedding of the form  $[e_{q_i}^{QT} \oplus e_{q_i}^{QB} \oplus e_{q_i}^t]$ . The corresponding 832 dimensional representations of  $q_1$  and  $q_2$  for the pair  $(q_1, q_2)$  are fed to the input layer of the MLP. The intermediate hidden layers are set to 512 and 64. The identified batch size is 64 and learning rate is  $2e-5$ .

$q_a$	$q^+$	PR (Dup-Predictor)	PR (DUPE)	PR (TE)	PR (TE+net)
363584	140121	24	308	45	39
362424	3491	3424	3324	4868	1363
363917	187651	579	2655	4	3
359502	45554	2227	4144	3621	59
365800	210683	17	2607	80	43

TABLE VI

$q_a$ : ANCHOR QUESTION,  $q^+$ : ACTUAL DUPLICATE, PR (DUPREDICTOR): PREDICTED RANK OF **DUPREDICTOR**, PR (DUPE): PREDICTED RANK OF **DUPE**, PR (TE): PREDICTED RANK OF **TE**, PR (TE+NET): PREDICTED RANK OF **TE+NETWORK**

Methods	RMSE	$\rho$
Text-DT	1.336	0.130
Text-XGB	1.278	0.189
Text-MLP	1.186	0.208
Text+Network-DT	1.312	0.130*
Text+Network-XGB	1.262	0.202*
Text+Network-MLP	1.180	0.213*

TABLE VII

DUPLICATE QUESTION CONFIRMATION TIME PREDICTION.  $\rho$ : SPEARMAN'S RANK CORRELATION, \*: RESULTS OF TEXT+NETWORK MODELS ARE SIGNIFICANTLY DIFFERENT FROM THE TEXT ONLY MODELS WITH  $p < 0.01$  USING M-W U TEST.

**Results:** The experimental results are presented in Table VII. The least RMSE is obtained for the text+network model using MLP. The Spearman's rank correlation ( $\rho$ ) between the gold and the predicted rank for all the 3756 test pairs is also best for the text+network model using MLP. Given such a massive list of pairs, we believe that our results are pretty impressive. Further, we observe that adding network features always brings statistically significant improvements.

## VIII. ERROR ANALYSIS

In this section, we test our models for various use cases to identify which variant of the model fails and when. Here, we demonstrate two use cases – (a) **TE** performs better than **TE+network**: In Table III (up), we show a few test examples where **TE** performs better than **TE+network**. We observe that **TE** performs better when the title of the anchor question ( $q_a$ ) is long and more detailed thus allowing the model to obtain a richer representation for the recommendation task. Even if the neighborhood of the most frequent tag of the anchor question



is sparse, this does not affect the performance since the title text is elaborate and thus already rich in information. (b) **TE+network** performs better than **TE**: In Table III (down), we show few test examples where **TE+network** performs better than **TE**. **TE+network** performs better when the number of words in the title of the anchor question ( $q_a$ ) is less but the size of the neighborhood of the most frequent tag of the anchor question is relatively high. This additional information from the network neighborhood compensates for the shorter length of the title text. This observation demonstrates how the network features could be effective in enhancing the overall performance of the model.

In Table VI, we present the predicted rankings for some of the most frequently asked questions. We observe inclusion of network features improves the rank of the actual duplicate in the rank list compared to the state-of-the-art and our text-based model; however, existing models perform better in some contexts. In the case of **DUPE** and **DupPredictor**, cosine similarity between titles, bodies, tags, and codes was generally used. Even then, cosine similarity scores as a feature do not help identify duplicates in a large ecosystem like Ubuntu since cosine similarity is more effective if the questions are either semantically similar or have a lot of word overlap.

## IX. CONCLUSION AND FUTURE WORK

In this paper, we have proposed methods to solve the two CQA-related problems –(i) duplicate question retrieval and (ii) duplicate question confirmation time. In both problem statements, our model outperforms other state-of-the-art baseline models. Further adding network features, we obtained statistically significant improvements. In the future, we would like to investigate the temporal characteristics of questions that are closed as a duplicate. In addition, we would like to study other comparable datasets and tackle similar problems.

## REFERENCES

- [1] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. Mining duplicate questions of stack overflow. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 402–412, 2016.
- [2] Anahita Alipour, Abram Hindle, and Eleni Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 183–192, 2013.
- [3] Dasha Bogdanova, Cícero dos Santos, Luciano Barbosa, and Bianca Zadrozny. Detecting semantically equivalent questions in online user forums. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 123–131, Beijing, China, July 2015. Association for Computational Linguistics.
- [4] Dasha Bogdanova, Cícero dos Santos, Luciano Barbosa, and Bianca Zadrozny. Detecting semantically equivalent questions in online user forums. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 123–131, Beijing, China, July 2015. Association for Computational Linguistics.
- [5] L. Breiman, Jerome H. Friedman, Richard A. Olshen, and C. J. Stone. Classification and regression trees. 1983.
- [6] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a “siamese” time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems, NIPS’93*, page 737–744, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.
- [7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. KDD ’16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.
- [8] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data, 2017.
- [9] Caichun Gong, Yulan Huang, Xueqi Cheng, and Shuo Bai. Detecting near-duplicates in large-scale short text databases. In Takashi Washio, Einoshin Suzuki, Kai Ming Ting, and Akihiro Inokuchi, editors, *Advances in Knowledge Discovery and Data Mining*, pages 877–883, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, 2016.
- [11] Rima Hazra, Hardik Aggarwal, Pawan Goyal, Animesh Mukherjee, and Soumen Chakrabarti. Joint autoregressive and graph models for software and developer social networks. In Djordj Hiemstra, Marie-Francine Moens, Josiane Mothe, Raffaele Perego, Martin Potthast, and Fabrizio Sebastiani, editors, *Advances in Information Retrieval - 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 - April 1, 2021, Proceedings, Part I*, volume 12656 of *Lecture Notes in Computer Science*, pages 224–237. Springer, 2021.
- [12] Rima Hazra, Arpit Dwivedi, and Animesh Mukherjee. Is this bug severe? a-text-cum-graph based model for bug severity prediction. In Massih-Reza Amini, Stéphane Canu, Asja Fischer, Tias Guns, Petra Kralj Novak, and Grigoris Tsoumakas, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 236–252, Cham, 2023. Springer Nature Switzerland.
- [13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
- [14] Yushi Homma, Stuart Sy, and Christopher Yeh. Detecting duplicate questions with deep learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2016.
- [15] Zainab Imtiaz, Muhammad Umer, Muhammad Ahmad, Saleem Ullah, Gyu Sang Choi, and Arif Mehmood. Duplicate questions pair detection using siamese malstm. *IEEE Access*, 8:21932–21942, 2020.
- [16] Reetu Kumari, Rohit Mishra, Shrikant Malviya, and Uma Shanker Tiwary. Detection of semantically equivalent question pairs. In Madhusudan Singh, Dae-Ki Kang, Jong-Ha Lee, Uma Shanker Tiwary, Dhananjay Singh, and Wan-Young Chung, editors, *Intelligent Human Computer Interaction*, pages 12–23, Cham, 2021. Springer International Publishing.
- [17] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.
- [18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.
- [19] Mohamed Shazan Mohamed Jabbar, Luke Kumar, Hamman Waqar Samuel, Mi-Young Kim, Sankalp Prabhakar, Randy Goebel, and Osmar Zaiane. Deepdup: Duplicate question detection in community question answering. In *Proceedings of the 2021 5th International Conference on Deep Learning Technologies, ICDLT ’21*, page 8–12, New York, NY, USA, 2021. Association for Computing Machinery.
- [20] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016.
- [21] Jiayan Pei, Yimin Wu, Zishan Qin, Yao Cong, and Jingtao Guan. Attention-based model for predicting question relatedness on stack overflow. *CoRR*, abs/2103.10763, 2021.
- [22] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.
- [23] Damar Adi Prabowo and Guntur Budi Herwanto. Duplicate question detection in question answer website using convolutional neural network. *2019 5th International Conference on Science and Technology (ICST)*, 1:1–6, 2019.
- [24] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.
- [25] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, apr 2009.

- [26] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. *29th International Conference on Software Engineering (ICSE'07)*, pages 499–510, 2007.
- [27] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262, 2011.
- [28] Liting Wang, Li Zhang, and Jing Jiang. Duplicate question detection with deep learning in stack overflow. *IEEE Access*, 8:25964–25975, 2020.
- [29] Rahulkrishna Yandrapally, Andrea Stocco, and Ali Mesbah. Near-duplicate detection in web app model inference. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20*, page 186–197, New York, NY, USA, 2020. Association for Computing Machinery.
- [30] Hui Yang and Jamie Callan. Near-duplicate detection by instance-level constrained clustering. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '06*, page 421–428, New York, NY, USA, 2006. Association for Computing Machinery.
- [31] Wei Emma Zhang, Quan Z. Sheng, Jey Han Lau, and Ermyas Abebe. Detecting duplicate posts in programming qa communities via latent semantics and association rules. In *Proceedings of the 26th International Conference on World Wide Web, WWW '17*, page 1221–1229, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.
- [32] Wei Emma Zhang, Quan Z. Sheng, Jey Han Lau, Ermyas Abebe, and Wenjie Ruan. Duplicate detection in programming question answering communities. *ACM Trans. Internet Technol.*, 18(3), apr 2018.
- [33] Yun Zhang, David Lo, Xin Xia, and Jian-Ling Sun. Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology*, 30(5):981–997, Sep 2015.