Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network

Lin Shi[1,§], Mingzhe Xing[1,2,§], Mingyang Li[1,2], Yawen Wang[1,2], Shoubin Li[1,2], Qing Wang[1,2,3*]
[1]Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China
{shilin,shoubin,wq}@iscas.ac.cn, mingzhe@itechs.iscas.ac.cn, {mingyang,yawen}@nfs.iscas.ac.cn

ABSTRACT

Online chatting is gaining popularity and plays an increasingly significant role in software development. When discussing functionalities, developers might reveal their desired features to other developers. Automated mining techniques towards retrieving feature requests from massive chat messages can benefit the requirements gathering process. But it is quite challenging to perform such techniques because detecting feature requests from dialogues requires a thorough understanding of the contextual information, and it is also extremely expensive on annotating feature-request dialogues for learning. To bridge that gap, we recast the traditional text classification task of mapping single dialog to its class into the task of determining whether two dialogues are similar or not by incorporating few-shot learning. We propose a novel approach, named FRMiner, which can detect feature-request dialogues from chat messages via deep Siamese network. We design a BiLSTM-based dialog model that can learn the contextual information of a dialog in both forward and reverse directions. Evaluation on the real-world projects shows that our approach achieves average precision, recall and F1-score of 88.52%, 88.50% and 88.51%, which confirms that our approach could effectively detect hidden feature requests from chat messages, thus can facilitate gathering comprehensive requirements from the crowd in an automated way.

KEYWORDS

Feature Requests, Requirements Engineering, Deep Learning, Siamese Network

ACM Reference Format:
Lin Shi[1,§], Mingzhe Xing[1,2,§], Mingyang Li[1,2], Yawen Wang[1,2], Shoubin Li[1,2], Qing Wang[1,2,3]. 2020. Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network . In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380356

*Corresponding author.
§ Both authors contributed equally to this research.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380356

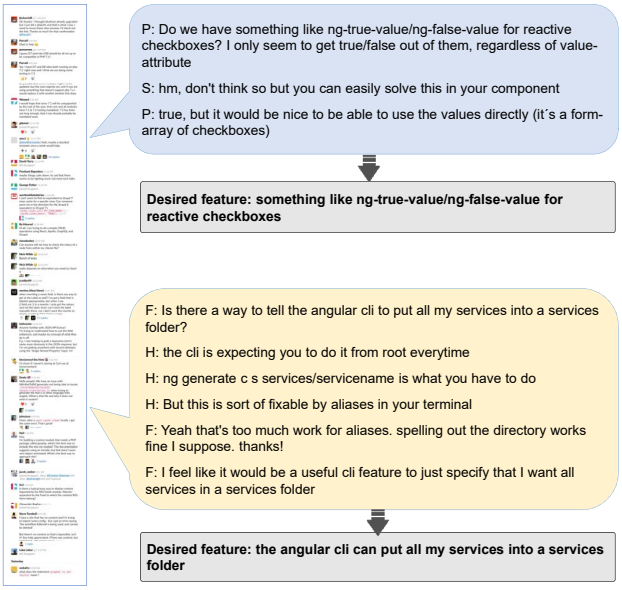

Figure 1: Example chat message from AngularJS project, where requests to desired features are buried in the massive chat history.

1 INTRODUCTION

Recent studies reported that the usage of online chatting is gaining popularity and plays an increasingly significant role in software development, having replaced emails in some cases [35, 56, 57]. Developers are turning to public workplace chat platforms, such as Slack, IRC, HipChat, Gitter, and Freenode to share opinions and interesting insights, discuss how to resolve defects as well as what features to implement in future [9].

Although developers reveal their desired features when communicating with other developers, the open and crowding nature of online chatting makes these feature-request dialogues get quickly flooded by newly incoming messages. Typically, the feature requests discussed in online chatting are likely to be buried and ignored if they are not documented. Taken the chat messages from AngularJS project as an example (Figure 1), developer P and F posted their questions in online chatting. In the beginning, their intentions are asking for help from other developers to seek feasible solutions to problems. After chatting with other developers, they realized that the existing system could not behave the way they want. Then

their intentions shift from asking for solutions to requesting features. P requests for "something like ng-true-value/ng-false-value for reactive checkbox", and F requests that "the angular cli can put all my services into services folder". In this work, we consider dialogues that contain requests for new functionalities/enhancements as feature-request dialogues. In practice, the release team monitors a variety of communication channels to have multiple sources of information that could be relevant to the next release [51, 61]. If the release team could acknowledge those hidden feature requests from chat messages, their next release planning may have the opportunity to maximize the stakeholder satisfaction by considering more feature requests [50].

Automated mining techniques towards retrieving valuable information from massive chat messages are badly needed for gathering comprehensive feature requests from a large number of users, which contribute to requirements elicitation and release planning, and in turn, promote the success of software development [6, 21, 25]. Although the chat messages could be a large volume and embedded with feature requests over time, it is quite challenging to mine massive chat messages due to the following barriers.

Dialog-wise analysis. Analyzing dialogues from chat messages differs from regular text mining tasks in that it needs to consider contextual information among the dialog-wise scope when understanding one single sentence. Therefore, existing studies on sentence-wise feature request detection [13, 25, 55] cannot be directly utilized for this task. For example, the sentence "We need to add vertical Navbar option" is classified as a feature request by sentence-wise techniques. But when posted in online chat, following-up conversation pointed out the existing functionality can fulfill that request in an alternative way. Moreover, the sentence-wise detection results will be inaccurate as a large number of off-topic sentences will be identified as feature requests in the chat messages, e.g., "I really need to get my programming skills back.", "I would like to get some coffee and cookies."

Extremely expensive annotation. The chat messages are typically large in size. Finding the feature requests dialogues among the massive chat messages is like looking for a needle in a haystack. It is extremely expensive to annotate feature requests from chat messages due to the high volume corpus and a low proportion of ground-truth data. Only a few labeled chat messages are categorized into feature request types. How to make the maximum use of the few labeled data to accurately classify the unlabeled chat messages becomes a critical problem.

Entangled and noisy data. Chat messages are typically high-volume and contain informal conversations covering a wide range of topics. Two or more developers synchronously interact with each other where their utterances are largely entangled in the chat messages. Moreover, there exist noisy utterances such as duplicate and off-topic messages in chat messages that do not provide any valuable information. The entangled and noisy data poses a difficulty to analyze and interpret the communicative dialogues.

In this work, we take the first step towards dialog-wise technique that aims to automatically detect hidden feature requests posted in chat messages. we propose a novel approach, named FRMiner, which can detect feature-request dialogues from chat messages via deep Siamese network. To better understand the contextual information in the dialog-wise scope, we first build a context-aware dialog model based on a bidirectional LSTM (BiLSTM) structure that can deeply learn the contextual information of a dialog in both forward and reverse directions. Inspired by the few-shot learning techniques that aim to build performance prediction models by utilizing insufficient labeled resources, we recast the traditional text classification task of mapping single dialog to its class into the task of determining whether two dialogues belong to the same or different class. Hence, we combine context-aware dialog models with the Siamese network to learn the similarity between a pair of dialogues rather than the patterns of a specific class. The prediction result of a feature-request dialog can be inferred based on the similarity prediction and the observed class of its partner dialog in the pair. To evaluate the proposed approach, we annotate 1,035 dialogues taken from three popular open-source projects. The experimental results show that our approach significantly outperforms two sentence-wise classifiers and four traditional text classification approaches with average precision, recall and F1-score of 88.52%, 88.50%, and 88.51%. The results confirm that our approach could effectively detect hidden feature requests from chat messages, thus can facilitate gathering comprehensive requirements from a large number of users in an automated way.

The major contributions of this paper are as follows.

- We are the first to promote detecting hidden feature requests from massive chat messages that can benefit comprehensive requirements gathering.
- We introduce a solution that can effectively predict feature-request dialogues based on limited labeled data by incorporating Siamese Network, which significantly relieves the burden of annotating supervised data.
- We evaluate our approach on three active open-source projects, and an empirical comparison shows that the proposed approach outperforms existing studies and four text classification approaches.
- Publicly accessible dataset and source code[1] to facilitate the replication of our study and its application in other contexts.

2 BACKGROUND

This section describes three key technologies related to this research: TextCNN, BiLSTM, and few-shot learning techniques. We include them here because our work is based on these technologies.

2.1 TextCNN

Dialogues in chat messages are the form of textual sentences recorded in the chronological order that were discussed by a community of developers during online communications. Modeling sentence representation is the foundation of high-level dialog analysis. In this paper, we represent sentences by using TextCNN [30], which has an advantage over learning on insufficient labeled data as it employs concise network structure and a small number of parameters.

TextCNN is a classical method for sentence modeling which uses a shallow Convolution Neural Network (CNN) [32] to model sentence representation. CNN is one kind of deep learning models that has been widely used in computer vision. It uses several convolution kernels to capture local information as the receptive field,

[1] https://github.com/FRMiner/FRMiner

then the global representation is produced with these local information. Analogously, in Natural Language Processing (NLP), CNN can aggregate n-gram information and model sentence representation. TextCNN takes the pre-trained or random generated word embedding as an input. The dimension of its output depends on the number and the size of convolution kernels. A n-length sentence can be represented as a matrix with a shape of $n \times d$, where d is the dimension of word embedding. Each kernel $w \in \mathbb{R}^{kd}$, where k is the size of convolution kernel, is applied to a window of k words to be mapped into a new one-dimension vector. Let $X_{i:i+k}$ represents the concatenation of k-gram words in the original sentence, and then a convolution operation will be performed on it. The output of the convolution layer can be computed as $o_i = f(w \cdot X_{i:i+k} + b)$ where b is a bias term and f is an activation function. Given the length l of a sentence and the convolution kernel size k, we can get the representation of the sentence, whose size is $l - k + 1$. The convolution layer is followed by a max polling layer, which can capture the key information with the highest value.

To obtain more sufficient semantic information ensembled by different scales of local information, multiple convolution kernels with different sizes are applied to the sentence. Hence, for a sentence, given $n \times m$ convolution kernels, where n is the number of different sizes of kernels, and m is the number of kernels of each size, we can get the sentence representation with size of $n \times m$, which encodes the different scales of local information into a global representation of the sentence.

2.2 Bidirectional LSTM

Analyzing dialogues from chat messages is a high-level text mining task as it needs to consider contextual information among the dialog-wise scope when understanding one single sentence. In this paper, we utilize the Bidirectional Long Short Term Memory network (BiLSTM), regarding the sentences of dialogues as sequential items, to capture the contextual information, where the representations of sentences are embedded by TextCNN. BiLSTM was proposed by Graves et al. [20] to learn bidirectional information for the sequence learning task. BiLSTM stacks two standard Long Short Term Memory network (LSTM) [23] layers with opposite directions to learn the one-way representation respectively. Then it combines the forward and backward representations as the bidirectional embedding. Long Short Term Memory network (LSTM) is an optimized Recurrent Neural Network (RNN) structure based on gate mechanism that was proposed by Hochreiter et al. [23]. LSTM utilizes gate mechanism to filter key information and pass them down to the long sequence. An LSTM cell composes of input gate, forget gate, cell state, and output gate. The outputs of LSTM cell gates can be specified as follows:

$$\begin{bmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \tilde{\mathbf{c}}_t \\ \mathbf{o}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ tanh \\ \sigma \end{bmatrix} \left(\mathbf{W} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b} \right)$$

$$\mathbf{c}_t = \tilde{\mathbf{c}}_t \odot \mathbf{i}_t + \mathbf{c}_{t-1} \odot \mathbf{f}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

where $\mathbf{x}_t$ is the i^{th} token in the input sequence; $\mathbf{W}$ is the weight matrix of LSTM cells; $\mathbf{b}$ is the bias term; σ denotes logistic sigmoid activation function, and **tanh** denotes hyperbolic tangent activation function; $\odot$ denotes element-wise multiplication.

Hence, the final representation of BiLSTM can be formed as $h = [\overrightarrow{h} \oplus \overleftarrow{h}]$, where $\overrightarrow{h}$ and $\overleftarrow{h}$ denote the outputs of two LSTM layers respectively, and $\oplus$ is concatenate operation.

2.3 Few-shot learning

Deep learning has gained significant success both in the field of computer vision and NLP. But it relies on adequate volume of training data heavily, and has difficulty in performing well when the labeled resource is insufficient. Automated mining in chat messages also faces the insufficient labeled resource problem. In online chatting, a large community of developers create plenty of discussions in a short time. It is extremely time-consuming to annotate a large number of dialog data for learning effective models since they are not only long but also require domain knowledge to thoroughly understand. Few-shot learning approaches are proposed to overcome these constraints [64]. The few-shot learning approaches can be classified into the following three categories [10]. Model-based approaches aim at learning projectors from few labeled data to taxonomies through model designing. Optimizer-based approaches adjust the traditional gradient descent optimizer method to fit the data. Metric-based approaches classify samples through learning similarity metric functions.

In this paper, we leverage a metric-based few-shot learning technique, named Siamese network [8], which is widely used to measure the semantic similarity among texts or images [45]. Traditional classification models are trying to learn a mapping from a single instance to its class, but they cannot always work well when there is low labeled resource data available. Different from the traditional approaches, Siamese network takes pairs of instances as inputs, aiming at learning key characteristics that determine whether the two instances belong to the same or different class. It consists of two identical sub-components that not only share model structure but also parameters to encode the pairs of instances respectively. Intuitively, it is easier for us to determine whether two dialogues are similar rather than given the exact class for each dialog. Since Siamese network takes pairs as inputs, the dataset is converted from element-wise to pair-wise, and can be augmented with permutations.

3 APPROACH

Figure 2 demonstrates the overall framework of our approach. We construct the training dataset by disentangling dialogues in the chat messages. Then we build a hierarchical context-aware dialog model for each dialogue. The context-ware dialog model encodes dialogues by BiLSTM structure which uses TextCNN-based sentence embedding as inputs. After that, we build a Siamese network with two identical context-aware dialog models. Finally, we infer the predictive class based on the probabilities produced by Siamese network and the actual labels of golden dialogues in the paired instance.

3.1 Dialogues Disentanglement

Chatting channels is one type of synchronous textual communication among a community of developers. Messages in chats form stream information, with conversations often entangling such as a single conversation thread is interleaved with other conversations.

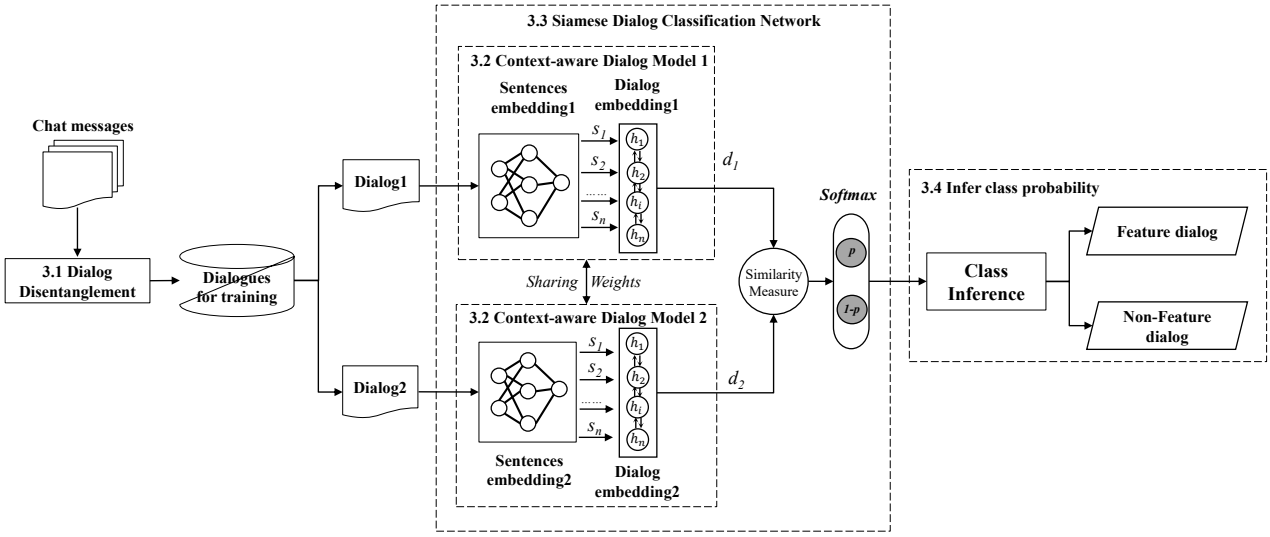

Figure 2: The Overview of FRMiner

Dividing chat messages into a set of distinct conversations is an essential prerequisite for any kind of high-level dialogue analysis. We leverage the state-of-the-art technique for conversation disentanglement proposed by Kummerfeld et al. [33]. Their model is trained from 77,563 manually annotated messages of disentangled dialogues from online chatting. It is a feedforward neural network with 2 layers, 512 dimensional hidden vectors, and softsign nonlinearities. The input of the model is a 77 dimensional vector, where each element is a numerical feature extracted from the original conversation texts, that include, time interval from previous chat messages posted by the current user, is there a target user in the chat content, do two chat texts contain the same words and so on. Figure 3 is a demonstration of the dialogues before and after disentanglement. The model can achieve relatively good performance with 74.9% precision and 79.7% recall.

```
[03:05] <delire> hehe yes. does Kubuntu have
'KPackage'?
=== delire found that to be an excellent
interface to the apt suite in another
distribution.
=== E-bola [...@...] has joined #ubuntu
[03:06] <BurgerMann> does anyone know a
consoleprog that scales jpegs fast and
efficient?.. this digital camera age kills me
when I have to scale photos :s
[03:06] <Seveas> delire, yes
[03:06] <Seveas> BurgerMann, convert
[03:06] <Seveas> part of imagemagick
=== E-bola [...@...] has left #ubuntu []
[03:06] <delire> BurgerMann: ImageMagick
[03:06] <Seveas> BurgerMann, i used that to
convert 100's of photos in one command
[03:06] <BurgerMann> Oh... I'll have a look..
thx =)
```

Figure 3: Example of dialogues before and after disentanglement [33]. These curves with different colors represent the links of different dialogues after disentanglement.

3.2 Build Context-aware Dialog Model

We design a hierarchical context-ware dialog model that can capture the contextual information as well as the semantic meaning of each sentence in a dialog. As shown in Figure 4, the context-aware dialog model consists of four layers: input layer, sentence embedding layer, dialog embedding layer, and output layer.

Input layer. We first tokenize the sentences into tokens as the basic terms. To obtain a better performance, we utilize the 50 dimension Glove word embeddings [49] that are pre-trained on 6 billion words of Wikipedia and Gigword corpus as the initial vectors of the corresponding words. Moreover, inspired by previous works [58] [55], we notice that part-of-speech (POS) patterns or templates obviously exist in feature-request texts. Intuitively, the POS tag can benefit semantic understanding by introducing explicit lexical information. Therefore, we add POS tag information into word representation to enhance its feature. Specifically, each type of POS tag will be initialized as a random vector with uniform distribution and optimized during training. Hence, each word can be represented as $w_i = [we_i \oplus pos_i]$, where we_i denotes the corresponding word embedding and pos_i denotes the embedding of the POS tag of the word.

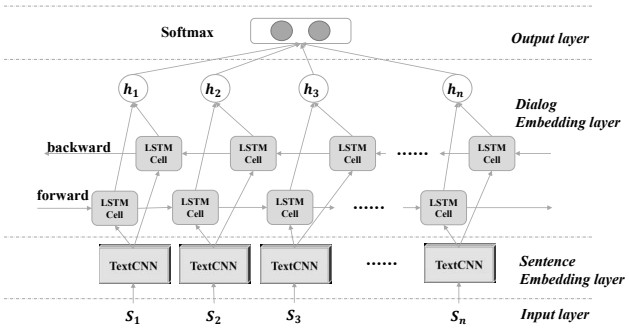

Figure 4: Hierarchical Context-aware Dialog Model

Sentence embedding layer. After transforming the original sentence into a matrix stacked with word embedding and POS tag embedding, we feed the embedding matrix into TextCNN to obtain the sentence representation. Details of representing sentences by TextCNN have been introduced in Section 2.1.

Dialog embedding layer. In the dialog embedding layer, we use a sequence of sentence embeddings to represent a dialog. Each embedded sentence acts as a token when inputting the BiLSTM encoder according to their sequences in the dialog. After encoding BiLSTM, the bidirectional contextual information of the dialog will be learned.

Output layer. In the output layer, we combine the two direction representations $\overrightarrow{h}$ and $\overleftarrow{h}$ encoded by BiLSTM as the output vector of the dialog, which can be presented as $h = [\overrightarrow{h} \oplus \overleftarrow{h}]$.

3.3 Construct Siamese Dialog Classification Network

In order to alleviate the insufficient labeled data problem, we construct the Siamese dialog classification network that can augment the dataset by recasting the traditional text classification task of mapping single dialog to its class into the task of determining whether two dialogues belong to the same or different classes. For the purpose of clarity, we will use **feature dialog** and **non-feature dialog** in the rest of this paper to refer to the dialogues that are requesting features and dialogues that are not requesting features.

The dashed box of '3.3' in Figure 2 presents the detailed architecture. The Siamese dialog classification network contains two context-aware dialog models that share structure and parameters to encode a pair of dialogues to d_1 and d_2 respectively. We use the combination forms of d_1 and d_2, $[d_1 \oplus d_2]$, as representations of the relation between the two dialogues. Then the representation of the relation between a pair of dialogues is projected from the dialog embedding to a similarity metric. Due to explicit equations for similarity measures, such as Cosine Similarity [24] and Euclidean Distance [24], usually used to measure the closeness between vectors in linear space, they are not suitable for complex dialog in semantic space. Therefore, we employ similarity function learned during training neural network. As the diff or same label of two dialogues can be obtained, we can train a similarity layer in the neural network. It is like a black-box component. The inputs are embedded representations of two dialogues with 'same' or 'diff' labels, the outputs are their similarity.

We train the Siamese Dialog Classification Network according to the following steps. (1) We divide the dataset into training and testing dataset randomly. *Train_d* is the original dataset that employs dialogues with label 'feature' or 'non-feature' as entries. (2) Typically, the size of labeled dialogues is insufficient for training effective learning-based models. To overcome that issue, we augment *Train_d* into *Train_p* by sampling a pair of dialogues from *Train_d* with label 'same' or 'diff' as one entry of *Train_p*. More specifically, for each dialog in training dataset, we randomly select a positive partner with 'feature' label, and a negative partner with 'non-feature' label from training dataset *Train_d*. For example, if the two dialogues are all *feature dialog* or *non-feature dialog*, we will assign the pair with label-'**same**', otherwise '**diff**'. Due to the one-positive-one-negative sampling policy, our training data can be balanced naturally. Besides, suppose we have *m feature dialogues* and *n non-feature dialogues* in *Train_d*, we can augment the origin data set to size of $\binom{m}{2} + \binom{n}{2} + m \times n$ in *Train_p*. Finally, since each pair of dialogues belongs to either 'same' class or 'diff' class, the output of similarity measure is a 2-length vector $[score_1, score_2]$ that represents the scores of the two classes, where $score_i \in \mathbb{R}$. We perform softmax on the 2-length vector, which can be specified as

$$Softmax(socre_i) = \frac{e^{score_i}}{\sum_{j=1}^{2} e^{score_j}}$$

Then the $[score_1, score_2]$ can be normalized to probabilities $[p, 1-p]$, where $p \in [0, 1]$.

3.4 Infer Class Probability

The output of Siamese dialog classification network is the probabilities that indicating whether two dialogues are *same* or *diff*. But what we need is the probabilities that indicating whether a dialog is a feature dialog or not, thus, we need to infer the label of a dialog based on the probabilities and the actual label of another dialog in the pair instance. For example, we sample a pair $< Dialog_1, Dialog_2 >$, where $Dialog_1$ is the golden dialog sampled from train dataset with observed 'non-feature dialog' label, and $Dialog_2$ is the unknown dialog to be predicted. We input the pair of them into the Siamese dialog classification network, then a prediction for deciding the two dialogues are *same* or *diff* is made. Suppose the prediction is *diff*, then we can infer that the class of $Dialog_2$ is a feature dialog. If the actual label of $Dialog_2$ is a feature dialog, it indicates that this prediction made by our model is true positive. Otherwise, we get a false positive prediction. To obtain a more reliable predict result, we employ the vote strategy during predicting phrase. For each unknown dialog, we construct k paired instances by sampling k different golden dialogues. After passing these pairs into FRMiner, we can get l instances which indicate the unknown one is a feature dialog and $k - l$ instances which indicate it is a non-feature dialog based on the predictions of FRMiner and the labels of golden dialogues. If l is greater than $\frac{k}{2}$, then we assign the predicted dialog with 'feature dialog' label, vice versa.

3.5 Tool Implementation

We implement our proposed approach, FRMiner, using Allennlp [16] which is an open-source NLP library built on PyTorch [15].

Implementation details. For these hyper-parameters, we use grid search [7] as the parameter selection method to obtain the best performance. The dimension of POS tag embedding is 50, the same as word embedding. Then, we can use $s = [w_1, w_2 \dots w_n]$ as the representation of sentences, where $w_i = [we_i \oplus pos_i]$. To obtain more sufficient semantic information ensembled by different scales of local information, multiple convolution kernels with different sizes are applied to the sentence. We set 4 different kernel sizes which are 2, 3, 4, 5 respectively and 25 feature maps for each kernel. The output dimension of BiLSTM is 300 (150 for each direction). We use a linear layer as the similarity layer to project the 300 dimension vector to two values that represent the probability scores of two classes. Since the task can be regarded as a classification problem, we use cross-entropy as the loss function.

Optimization. In addition, to avoid the over-fitting problem, we apply dropout [59] on the input embeddings with 0.1 drop rate, which means, 10% neuron cell will be randomly masked to reduce the parameters that need to be trained in each batch training. We also use the strategy of early stopping [52]. If the performance on the test dataset did not promote for 10 epochs, the training process will be stopped.

4 EXPERIMENTAL DESIGN

4.1 Research Questions

Our evaluation addresses the following three research questions.

RQ1: How effective is our approach for detecting hidden feature requests? To investigate the effectiveness of our approach, we conduct 3-fold cross-validation on detecting feature dialogues from chat messages of three open-source projects. We also compare the performances of two sentence-wise approaches that classify sentences of online discussions into feature requests and other types. We adapt the two approaches to our task by predicting dialogues that contain feature-request sentences as feature dialogues. Besides, we examine the performances of four widely used classification methods on the limited labeled resources to perceive the difficulties of automated feature request mining in chat messages.

RQ2: How does the Siamese Network facilitate feature request detection? To examine the performance enhancement introduced by the Siamese Network, we construct p-FRMiner, which is a plain FRMiner without incorporating Siamese Network technique. Detailed difference between FRMiner and p-FRMiner will be described in Section 4.3. We then compare the performance of p-FRMiner that are directly trained by dialog-instances, with the performance of FRMiner that are trained by pair-instances. Then we increasingly enlarge the size of the training pair-instances to examine the relationship between performance enhancement and data augment.

RQ3: Does our approach work well in cross-project validation? RQ3 examines the generalizability of our approach via cross-project validation on three open-source projects. We iteratively use two projects for training and the reserving one for testing. We also conduct cross-project validation on baseline approaches.

4.2 Data Preparation

Our experimental data is crawled by Scrapy [53] from three open-source projects: AngularJS [17], Bootstrap [60], and Chromium [18]. We select the three projects for the following reasons. First, they are under active developments. Second, large communities are formed around those projects. Third, developers from these projects actively use online chatting to share opinions, interesting insights, and discuss what features to implement in the future. For example, in the last three years, an average of 2,823 utterances are made per week in AnguluarJS community. Moreover, their historical chat messages are all documented and publicly accessible [1], which provide rich resources for mining valuable information. Our data is collected in the following steps:

Step 1: Preprocess. We first normalize non-ascii characters like emojis to standard ascii strings. Some low-frequency tokens cannot contribute to the result of classification such as URL, email address, code, HTML tags, and version numbers in chat messages. We replace them with specific tokens *<URL>, <EMAIL>, <HTML>, <CODE>, and <ID>* respectively. We utilize Spacy [2] to tokenize sentences into terms. To alleviate the influence of word morphology, we then perform lemmatization and lowercasing on terms with Spacy.

Table 1: The statistic of labeled dialogues

	Massive chat messages			Sample		
	Duration	#dialog	#sentence	#dialog	#sentence	#FR
AngularJS	2016.5-2019.4	38266	406553	316	9220	36
Bootstrap	2014.7-2019.5	10358	58871	379	2371	76
Chromium	2015.5-2019.7	16804	118890	340	4465	27
Total		65428	584314	1035	16056	139

Step 2: Dialogues Sampling. After disentanglement, there are a large number of identified chat dialogues. To observe the characteristics of the entire dialogues population, we randomly sample 400 dialogues from the three projects. Then we excluded unreadable dialogues: 1) Dialogues that are written in non-English languages; 2) Dialogues that contain too much code or stack traces; 3) Low quality dialogues such as dialogues with many typos and grammatical errors. 4) Dialogues that involve channel robots.

Step 3: Ground-truth Labeling. The labeled dialogues are used as the ground-truth dataset for method definition and performance evaluation. To guarantee the correctness of the labeling results, we built an inspection team, which consisted two senior researchers with four Ph.D candidates. All of them are fluent English speakers, and have done either intensive research work with software development or have been actively contributing to open-source projects. We divided the team into two groups. Each group consisted a leader (senior researcher) and two members. The leaders trained members on how to label and provided consultation during the process. The labeling results from the members were reviewed by the leaders while results from the leaders were reviewed by other leaders. We only accepted and included dialogues to our dataset when the dialogues received full agreement among the groups. When an dialog received different labeling results, we hosted a discussion with all the six people to decide through voting.

In total, we collected 65,428 dialogues from three open-source projects, and spent 720 person-hours on annotating 1,035 (1.6%) dialogues. The detailed characteristics of labeled dialogues are described in Table 1. The last column '#FR' denotes the number of feature dialogues.

4.3 Experiment Settings

We conduct 3-fold cross-validation [31] on the dataset collected from three open-source projects. We randomly divide our dataset into 3 parts. We use 2 of those parts for training and reserve one part for testing. We repeat this procedure 3 times each time reserving a different part for testing. The experimental environment is a desktop computer equipped with an NVIDIA 1060 GPU, intel core i7 CPU, 16GB RAM, running on Ubuntu OS.

Experiment I (RQ1) To prove the effectiveness of our approach, we select two advanced sentence-wise approaches and four text classification approaches as baselines. Detail information about baselines will be introduced in section 4.4. For the two sentence-wise approaches, we use the codes and models provided in the publications. For the four text classification approaches, we use the codes provided by official released packages [19] [14]. We apply the random over-sampling [36] to tackle with imbalance dataset. We extract the Term Frequency and Inverse Document Frequent

(TFIDF) [27] as feature vectors for each dialog. We train and fine-tune hyper-parameters by grid search for the four text classification approaches to achieve their best performances.

Experiment II (RQ2) In this experiment, we compare FRMiner with p-FRMiner and investigate how does the data augment improves performance. **Note that p-FRMiner is different with FRMiner** in that p-FRMiner is a classification model for a single dialog which is based on the context-aware dialog model (introduced in Section 3.2) following with an additional classification layer. The architecture of FRMiner can be derived from p-FRMiner by the following steps: 1) remove the top top-classification layer of p-FRMiner; 2) concatenate the outputs of two p-FRMiner that sharing weights; 3) add a similarity layer. First, we use the same size of data to train FRMiner and p-FRMiner, and observe the performance enhancement. Then we augment the training dataset of FRMiner 5, 10, 20, and 30 times to investigate the performance changes. Since FRMiner can tackle with imbalanced dataset issue by applying the Siamese network while p-FRMiner can not, we balance the samples by applying random over-sampling [36] when training p-FRMiner. To ensure the correctness of our experiments, FRMiner and p-FRMiner are trained with the same hyper-parameters, including dimensions for each layer, depth of the network, and learning rate.

Experiment III (RQ3) To validate whether our approach is generalizable to unfitted projects, we trained the FRMiner on two projects and evaluate on third projects. We also evaluate other baselines on the cross-project dataset with the identical hyper-parameters of experiment II.

4.4 Baselines

To demonstrate the advantages of FRMiner, we compare FRMiner with two advanced sentence-wise approaches as our baselines.

CNN-based Classifier (CNC) [25]. It is the state-of-the-art learning technique to classify sentences in comments taken from online issue reports. They proposed a convolution neural network (CNN) based approach to classify sentences into seven categories of intentions: Information Giving, Information Seeking, Feature Request, Solution Proposal, Problem Discovery, Aspect Evaluation, and Meaningless. We utilize the Feature Request category to predict dialogues that contain feature-request sentences as feature-request dialogues.

Rule-based Classifier (FRA) [55]. It is the state-of-the-art rule-based technique to classify sentences in feature requests from online issue tracking systems. They proposed 81 fuzzy rules to classify sentences into 6 types. We consider the dialogues that contain the *Intent* type of sentences are predicted to be feature dialogues, and dialogues do not contain the *Intent* type of sentences are predicted to be non-feature dialogues.

Machine-learning-based Classifiers. *Naive Bayesian (NB)* [38] is a simple generation model for text classification based on bag-of-words assumptions and Bayesian rules. It conducts the joint probability of a sentence through prior probability and conditional probability learned by the model and training data. Then, given a sentence, it can deduce the probabilities of all the taxonomies. *Random Forest (RF)* [34] is an ensemble machine learning method that is constructed with several trees, and each tree can contribute to the final classification result. *Gradient Boosting Decision Tree (GBDT)* [29] is another kind of ensemble method, and the difference with RF is that its trees are decided by the residual error brought by the previous trees. When training GBDT, we set the initial learning as 1.0 and the max depth of trees as 1. We trained 100 epochs for FT, and set the initial learning rate as 1.0, the window size of input n-gram as 2. *FastText (FT)* [28] is the state-of-the-art text classification approach with a shallow neural network that is similar to the architecture of word2vec [42]. When training RF, we set the max depth of trees as 2.

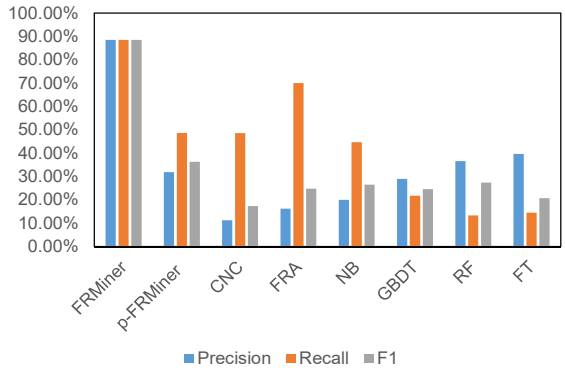

Figure 5: The average performances in 3-fold validation

4.5 Performance Measures

When evaluating the effectiveness of FRMiner towards detection feature requests from chat messages, we used the following metrics: (1) Precision, which refers to the ratio of the number of correct predictions of feature dialogues to the total number of predictions of feature dialogues; (2) Recall, which refers to the ratio of the number of correct predictions of feature dialogues to the total number of feature dialogues in the golden test set; and (3) F1-Score, which is the harmonic mean of precision and recall.

5 RESULTS AND ANALYSIS

5.1 Answering RQ1

Figure 5 presents the average performances achieved by different approaches in 3-fold cross-validation, and Table 2 presents the detailed performances for each project. The best results of precision, recall, and F1-score are highlighted in bold. We can see that FRMiner achieves the best results for all the three projects, with an average of 88.52%, 88.50%, and 88.51% in precision, recall, and F1-score. We also note that p-FRMiner performs better than all the baseline approaches, which indicates that memorizing contextual information in the BiLSTM dialog model can benefit the text classification task in chat messages. We further evaluate and analyze the improvement of FRMiner over p-FRMiner in section 5.2.

For the two sentence-wise approaches, the CNN based classifier can only achieve 17.33% F1-score on average, mainly because that the CNC model is trained by sufficient data from the domain of issue comments instead of feature dialogues. However, it still achieves 48.55% recall on average, which indicates that there might be common patterns between the two domains. Transfer learning techniques might help transfer the related common knowledge

Table 2: The performance achieved by different approaches for each project in intra-project validation

Approach \ Performance		AngularJS			Bootstrap			Chromium		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Our approach	FRMiner	**90.28%**	**89.73%**	**90.00%**	**86.28%**	**88.78%**	**87.52%**	**89.00%**	**87.00%**	**88.00%**
	p-FRMiner	31.71%	54.17%	40.00%	50.00%	47.80%	48.98%	14.00%	44.00%	20.00%
Existing studies	CNC	7.70%	44.44%	13.13%	16.38%	34.21%	22.13%	9.56%	67.00%	16.73%
	FRA	13.67%	80.33%	23.35%	23.00%	48.67%	31.00%	12.00%	81.00%	20.00%
Text classification	NB	20.00%	27.67%	22.33%	25.67%	62.00%	36.00%	14.33%	44.33%	21.00%
	GBDT	36.00%	22.33%	27.33%	41.67%	35.67%	38.33%	9.33%	7.33%	8.00%
	RF	52.67%	11.00%	16.33%	57.00%	29.00%	38.33%	0.00%	0.00%	NA
	FT	23.33%	5.33%	8.67%	57.67%	29.00%	38.33%	38.00%	9.10%	15.00%

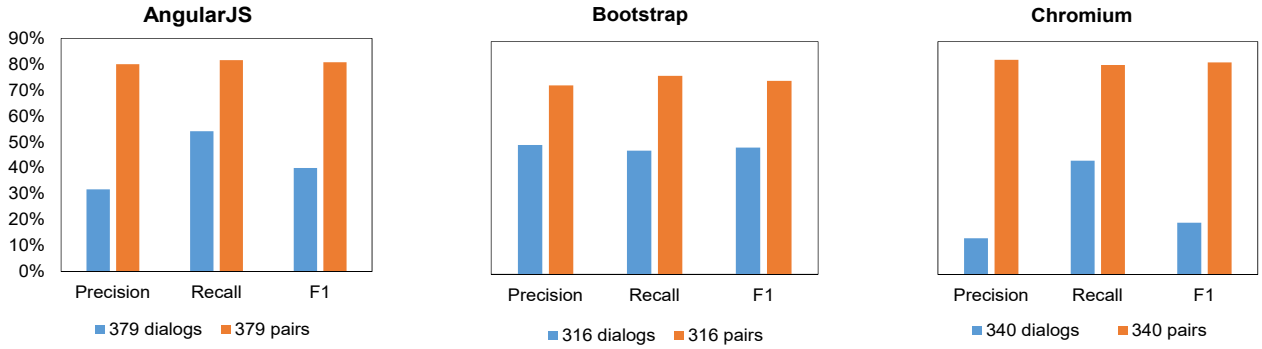

Figure 6: The comparison performances of p-FRMiner and FRMiner with the same volume of original training data.

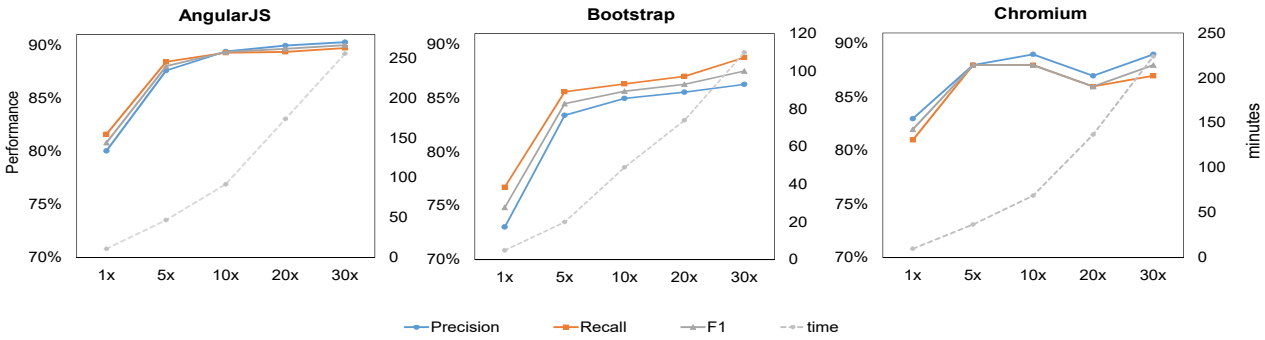

Figure 7: The performances of FRMiner when generating different numbers of pairs.

Table 3: The performance achieved by different approaches for each project in cross-project validation

Approach \ Performance		AngularJS			Bootstrap			Chromium		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Our approach	FRMiner	**85.23%**	**86.56%**	**85.89%**	**86.84%**	**85.89%**	**86.37%**	**85.87%**	**86.81%**	**86.34%**
	p-FRMiner	31.03%	50.00%	38.30%	27.56%	69.08%	39.40%	16.00%	50.00%	24.24%
Existing studies	CNC	7.70%	44.44%	13.13%	16.38%	34.21%	22.13%	9.56%	67.00%	16.73%
	FRA	13.67%	80.33%	23.35%	23.00%	48.67%	31.00%	12.00%	81.00%	20.00%
Text classification	NB	16.00%	75.00%	26.00%	27.00%	36.00%	31.00%	7.00%	26.00%	12.00%
	GBDT	18.00%	14.00%	16.00%	30.00%	11.00%	16.00%	20.00%	19.00%	19.00%
	RF	28.00%	14.00%	19.00%	37.00%	9.00%	15.00%	12.00%	26.00%	16.00%
	FT	32.00%	19.00%	24.00%	43.00%	13.00%	20.00%	19.00%	11.00%	14.00%

from issue comments to chat messages by parameter-transfer and fine-tuning [47]. Meanwhile, the rule-based classifier FRA achieves the highest recall among the six baseline approaches. The average recall is 70.00%, and it achieves 80.33% and 81.00% recall on AngularJS and Chromium project respectively. Although the precision is low, prediction results of FRA contain most of the actual feature

dialogues, which means that the feature-request sentences in chat messages also comply with FRA rules. As FRA utilizes rules instead of supervised learning, it is easier and time-saving to apply in mining massive chat messages. Meanwhile, we can take the high-recall advantage of FRA to tackle with the cold start problem where no annotated resources are provided in the beginning.

For text classification approaches, NB seems to be the best classifier among all the text classification baselines. Although RF achieves the highest 27.33% F1-score on average among the four approaches, it encounters an underfitting problem on the Chromium project. It can neither model the training data nor generalize to new data, mainly due to the Chromium data is not large enough for RF model to learn the relevant patterns. The reasons why FRMiner noticeably outperforms the four traditional text classification models are: (1) compared with traditional text-classifications, neural models have a larger capacity which can achieve better performances, especially for the complex dialog modeling task. (2) it is easier for the model to identify whether two dialogues belong to the same class than classifying every single dialog. (3) text classification algorithms are not trained sufficiently from the small dialog datasets, while FRMiner is a pair-wise approach that can augment the original dataset dramatically which ensures the training to be sufficient.

Summary: FRMiner significantly outperforms two sentence-wise baselines and four traditional text classification approaches. As the two sentence-wise baselines can be directly applied to chat messages and achieve relatively good recall, they have the natural advantages over other approaches under the cold-start situation.

5.2 Answering RQ2

Figure 6 demonstrates the performance of p-FRMiner training by single-dialogue instances and the performances of FRMiner training by the same sizes of pair-dialogue instances. The blue column denotes the size of the training dataset for p-FRMiner, which is the size of 2 folds of data. The orange column denotes the size of pair-instances generated by the Siamese network. We can see that when the sizes of training datasets are the same, FRMiner can achieve much higher performances than p-FRMiner. The FRMiner improves the Precision, Recall, F1-score over the p-FRMiner by 46.79%, 31.13%, 42.90% on average.

Figure 7 illustrates the relationship between performance enhancement and training pair-instances quantity, along with the time cost on the training phase. The initial volumes (1x) are the original sizes of training data shown in Figure 6, which are 379, 316, and 340 pairs. We can see that enlarging the size of training pair-instances can moderately increase the model performances. When enlarging the size of the training pair-instances from 1 to 30 times, the precision, recall, and F1-score increase 9.82%, 8.72%, and 9% on average. We observe that the performance sharply increases when enlarging the training dataset 5 times, and slowly increases from 5 times to 30 times on all the projects. The performances of the Chromium project even slightly decline when enlarging by 20 times. While the time cost on the training phase is linearly increased all the time. Therefore, we consider that enlarging the training dataset 5 times might be a trade-off choice between effectiveness and efficiency.

Summary: FRMiner can better resolve the classification task than p-FRMiner by significantly improving the Precision, Recall, F1-score by 46.79%, 31.13%, 42.90% on average. The results confirm that it is easier for the model to recognize whether two dialogues belong to the same class rather than classifying the exact class directly when labeled dialogues are few. We consider 5 times to be a trade-off choice between effectiveness and efficiency because after enlarging the training dataset 5 times, the performances slowly increase but the time cost rises largely.

5.3 Answering RQ3

Figure 8 presents the average performances achieved by different approaches in the setting of cross-project validation, and Table 3 presents the detailed performances for each project. The best results of precision, recall, and F1-score are highlighted in bold. Note that the performances of CNC and FRA are the same with 3-fold intra-project validation due to the two approaches are not trained by feature dialogues. We repeat their results in Table 2 for comparison and analysis purposes.

We can see that FRMiner can also perform well in cross-project settings. Performance only slightly declines by 2.27% over average F1-scores compared with the result in intra-project validation. We consider that dialogues expressing feature-requests share common linguistic patterns across domains that are typically not relevant with domain-specific concepts. The results show that FRMiner can learn these common patterns and be generalized to other projects. It indicates that developers express feature requests in a similar way even in different communities and projects, and the feature dialogues of different projects share similar patterns. We note that p-FRMiner does not perform as good as within-project validation. The average F1-score declines 10.51%.

For text classification approaches, NB achieves the highest F1-score of 23%, and it only slightly declines 3.44% on average compared with within-project validation. None of the text classification approaches encounter an underfitting problem because the size of the training dataset for cross-project validation is larger. In addition, we notice that most text classification approaches perform better in cross-project evaluation than in intra-project evaluation for Angular and Chromium. It is mainly due to two reasons: (1) The cross-project training dataset involves two-project data while the intra-project only has 2/3 data of one project. Training with a much larger dataset results in a more robust classifier. (2) Cross-project evaluation imports two projects for training while intra-project evaluation only has one project. The wider scope of training dataset would increase the generalizability of the classifier due to the biased knowledge introduced by different projects.

Summary: FRMiner can also achieve high performance on unfitted projects, which indicates that FRMiner is generalizable to other projects. We also observe that NB is the best classifier among all the text classification baselines towards mining chat messages.

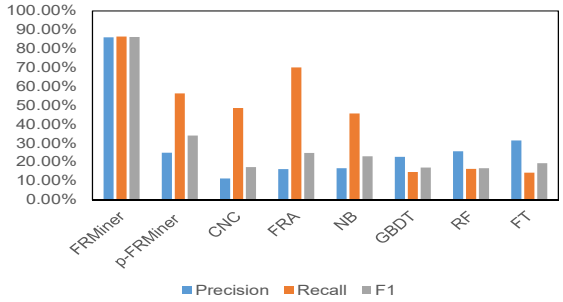

Figure 8: Average performances in cross-project validation

6 DISCUSSIONS

Applicability. Our work can benefit in gathering crowd requirements by conveniently integrating FRMiner into the workflow of the release-team members. First, the release team could build a chat-message monitor or a crawler to collect the textual conversation from the organization's chatting platform periodically. Then an automatic script [33] is applied to preprocess and disentangle the raw chatting text. After that, by performing the inference process mentioned in section 3.4 on the disentangled dialogues, FRMiner can record all the dialogues that are likely to request features. The release team could also subscribe to the monitoring results as an RSS feed to receive hidden feature requests periodically. Besides, due to people expressing feature dialogues in a relatively consistent way, FRMiner users do not need to retrain the model quite frequently. The retraining mainly need to be performed when the amount or the quality of the dataset changes extraordinarily.

Extendibility. We notice that people express feature dialogues in a relatively consistent way, for example, other than chatting messages, people also use similar expressions such as "need implement sth. in next release version" and "sth. will be a solution/improvement", to indicate feature requests in other open-source platforms including Github Issues and development emails. Hence, we argue that our approach can be extended to other data sources. In addition, FRMiner can be applied to the other languages since our deep contextual dialog model has a strong ability in capturing semantic patterns. When switching to other languages, FRMiner users need to adapt the pre-trained word embedding model to the specific language. They also need to consider to apply extra preprocessing according to the specific language, e.g., apply word segmentation to Chinese corpus. Another limitation of FRMiner on language switching is related to the pre-trained dialogues disentanglement model. A new dataset needs to be annotated for retraining the model, which may involve a relatively high cost.

7 THREATS TO VALIDITY

External Validity. The external threats relate to the generalizability of the proposed approach. All the three systems examined in this work were open-source projects, which might not be representative of closed-source projects. It is also possible that we accidentally chose systems that have better or worse than average cross-project feature requests detection performance. However, the cross-project evaluation results show that our approach is generalizable on the three studied projects, which largely alleviates the threat.

Internal Validity. The internal threats relate to experimental errors and biases. Threats to internal validity may come from the results of conversation disentanglement. The accuracy of disentangled conversations has impact on our results. To reduce the threat, when separating single conversations in a stream of chat messages, we employed the state-of-the-art technique proposed by Kummerfeld et al. [33], which outperforms previous studies by achieving 73.5% F1-score and 91.5 VI[2].

Construct Validity. The construct threats relate to the suitability of evaluation metrics. We utilize precision and recall to evaluate the performance, in which we use the manually labeled dialogues as ground-truth when calculating the performance metrics. The threats might come from the process of manual inspection and tagging. We understand that such a process is subject to mistakes. To reduce that threat, we build two inspection teams to reach agreements on different options.

[2]Variation of Information (VI) is a measure of information gained or lost when going from one clustering to another

8 RELATED WORK

Our work is related to previous studies that focused on (1) detection of feature requests; and (2) mining development communication artifacts. We briefly review the recent works in each category.

8.1 Detection of Feature Requests

The amount of research on gathering and analyzing information from a crowd to derive validated user requirements/feature requests has increased significantly in the last years.

Di Sorbo et al. [58] proposed a taxonomy of intentions to classify sentences in developer mailing lists into six categories: feature request, opinion asking, problem discovery, solution proposal, information seeking, and information giving. Although the taxonomy has been shown to be effective in analyzing development emails and user feedback from app reviews [48], Huang et al. [25] found that it cannot be generalized to discussions in issue tracking systems, and they addressed the deficiencies of Di Sorbo et al's taxonomy by proposing a convolution neural network (CNN)-based approach. Arya et al. [6] identified 16 information types including potential new feature requests through quantitative content analysis of 15 issue discussion threads. They also provided a supervised classification solution by using Random Forest with 14 conversational features that can classify sentences expressing new feature requests with 0.66 F1-score. Morales-Ramirez et al. [43, 44] identified requirement-related information in OSS issue discussion using 20 speech-act rules supported by NLP and linguistic parsing techniques. Merten *et al.* [40] investigated natural language processing and machine learning features to detect software feature requests in issue tracking systems. Their results showed that software feature requests detection can be approached on the level of issues and data fields with satisfactory results. Merten *et al.* [41] also investigated how requirements communicated in issue tracking systems by manually reviewing 200 issues. They categorized the text and reported on the distribution of issue types and information types. Their results showed that information with respect to prioritization and scheduling can be found in natural language data. Herzig *et al.* [22] manually examined more than 7,000 issue reports, and discussed the impact of misclassification of bugs in the bug databases of five open source projects. Their results showed that 39% of files marked as defective actually new features, updates to documentation, or internal refactoring. The authors suggested that human should always be involved when dealing with the posted issues. Antoniol *et al.* [5] investigated whether the text of the issues posted in bug tracking systems is enough to classify them into corrective maintenance and other kinds of activities. They alternated among various machine learning approaches such as decision trees, naive Bayes classifiers, and logistic regression to distinguish enhancement apart from other issues posted in the system. Shi et al. [55] proposed 81 fuzzy rules that can classify sentences in issues into six categories: intent, benefit, drawback, example, explanation, and

trivia. Their work designed to help understanding and analyzing real intents of feature requests, which can also benefit the detection of feature requests. Rodeghero *et al.* [39] presented an automated technique that extracted useful information from the transcripts of developer-client spoken conversations to construct user stories. They used machine learning classifiers to determine whether a conversation contains user story information or not. Maalej and Nabil [37] leveraged probabilistic techniques as well as text classification, natural language processing, and sentiment analysis techniques to classify app reviews into bug reports, feature requests, user experiences, and ratings. Their results showed that the classification can reach the precision between 70-95% and recall 80-90% actual results. Other studies have been found to also capture user needs from app reviews automatically [13, 26, 46, 62]. Vlas and Robinson [63] proposed a grammar-based design of software automation for the discovery and classification of natural language requirements found in open source projects repositories. Cledland-Huang *et al.* [12] designed an automated forum management (AFM) system, which was used to automated detect duplicated feature requests that have been already posted in the issue tracking systems. Shi *et al.* [54] proposed an initial approach to automated identify feature requests that ask for features that have been already implemented by applying feature tree model. Summing up, previous approaches differ from our work as it: identified feature requests from development emails [25, 58]; identified feature requests from issue tracking systems [5, 6, 22, 40, 41, 43, 44, 55]; identified user stories from spoken conversations [39]; identified feature requests from app reviews [13, 37, 46, 48, 62]; identified feature requests from project repositories [63] detected duplicated feature requests [12] [54].

Our work differs from existing researches in that we focus on detecting hidden feature requests from chat messages which post different challenges as chat messages are informal, unstructured, noisy and typically have insufficient labeled data than the previously analyzed documents. In addition, our work complements to the existing studies on automated feature requests detection.

8.2 Mining Development Communication Artifacts

Previous researches on development communication artifacts reported that the usage of online chatting play an increasingly significant role in software development, and chat messages are a rich source for valuable information about the software system. Lin et al. [35] conducted an exploratory study on how developers use Slack, which is a popular workplace chat app, and how they benefit from it. Their research revealed that developers use Slack for personal, team-wide, and community-wide purposes, and Slack plays an increasingly significant role in software development, replacing email in some cases. Shihab et al. [56, 57] analyzed the usage of developer IRC meeting channels of two large open source projects from several dimensions: meeting content, meeting participates, their contribution, and meeting styles. Their results showed that IRC meetings are gaining popularity among open source developers, and highlighted the wealth of information that can be obtained from developer chat messages. Yu et al. [66] analyzed the usage of two communication mechanisms in global software development projects, which are synchronous (IRC) and asynchronous (mailing list). Their results showed that developers actively use both communication mechanisms in a complementary way. Chatterjee et al. [9] conducted an exploratory study to investigate the usefulness and challenges of mining developer conversations for supporting software maintenance and evolution. They observed that developers are likely to share opinions and interesting insights on tool usage, best practices, and various technologies via instant conversations. They also reported that it is feasible to achieve high accuracy in disentangling conversations by adapting the techniques and training sets. Alkadhi et al. [3, 4] identified five rationale elements which are issue, alternative, pro-argument, con-argument, and decision from chat messages that collected from three student projects. They developed two supervised classifiers to automated detect rationale elements on the manually labeled data. Wood et al. [65] discovered 26 speech act types in the chat conversations during bug repair, and trained a supervised classifier to automatically detect these speech acts. Chowdhury and Hindle [11] implemented machine learning techniques to filter out off-topic discussions in programming IRC channels by engaging StackOverflow discussions as positive examples and YouTube video comments as off-topic discussion examples.

The findings of previous work motivates the work presented in this paper. Our study is different from the previous work as we focus on detecting feature requests hidden in massive chat messages that would be important and valuable information for OSS developers to enhance their software.

9 CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel approach, named FRMiner, which can detect feature dialogues from chat messages via deep Siamese network. In FRMiner, we incorporated two BiLSTM-based dialog models with the Siamese network to learn the similarity between a pair of dialogues rather than the class of a specific dialog. We evaluated FRMiner on a small sample of 1,035 dialogues taken from the high-volume chat messages of three popular open-source projects. The experimental results showed that our approach significantly outperformed two sentence-wise classifiers and four traditional text classification approaches with average precision, recall and F1-score of 88.52%, 88.50%. and 88.51%. FRMiner can also achieve high performance on unfitted projects, which indicated that FRMiner is generalizable to other projects. The experimental results confirmed that our approach could effectively detect hidden feature requests from chat messages. We also observed that NB seems to be the best classifier for chat messages among the four text classification baselines. In the future, we plan to employ NLP summarization technologies with our approach to extract a brief summary for developers, which can reduce the effort on reading feature dialogues. Moreover, we plan to extend this work by not only classifying but also recording feature requests in a well-designed and structured format from chat messages.

10 ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China under grant No.2018YFB1403400, the National Science Foundation of China under grant No.61802374, No.61432001, No.61602450.

REFERENCES

[1] 2019. echelog. https://echelog.com/.

[2] Explosion AI. 2019. Spacy. https://spacy.io/.

[3] Rana Alkadhi, Teodora Lata, Emitza Guzmany, and Bernd Bruegge. 2017. Rationale in Development Chat Messages: an Exploratory Study. *Mining Software Repositories* (2017), 436–446.

[4] Rana Alkadhi, Manuel Nonnenmacher, Emitza Guzman, and Bernd Bruegge. 2018. How do developers discuss rationale?. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 357–369.

[5] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2008. Is It a Bug or an Enhancement?: a Text-based Approach to Classify Change Requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. ACM, 23.

[6] Deeksha Arya, Wenting Wang, Jin L. C. Guo, and Jinghui Cheng. 2019. Analysis and detection of information types of open source software issue discussions. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. 454–464.

[7] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13, 1 (2012), 281–305.

[8] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1994. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*. 737–744.

[9] Preetha Chatterjee, Kostadin Damevski, Lori Pollock, Vinay Augustine, and Nicholas A Kraft. 2019. Exploratory study of slack Q&A chats as a mining source for software engineering tools. In *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 490–501.

[10] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Wang, and Jia-Bin Huang. 2019. A Closer Look at Few-shot Classification. In *International Conference on Learning Representations*.

[11] Shaiful Alam Chowdhury and Abram Hindle. 2015. Mining StackOverflow to filter out off-topic IRC discussion. (2015), 422–425.

[12] Jane Cleland-Huang, Horatiu Dumitru, Chuan Duan, and Carlos Castro-Herrera. 2009. Automated Support for Managing Feature Requests in Open Forums. *Commun. ACM* 52, 10 (2009), 68–74.

[13] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What Would Users Change in My App? Summarizing App Reviews for Recommending Software Changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 499–510.

[14] Facebook. 2019. FastText. https://fasttext.cc/.

[15] Facebook. 2019. PyTorch. https://pytorch.org/.

[16] Allen Institute for Artificial Intelligence. 2019. AllenNLP. https://allennlp.org/.

[17] Google. 2019. AngularJS. https://angularjs.org/.

[18] Google. 2019. Chromium. https://www.chromium.org/.

[19] Google. 2019. Scikit-Learn. https://scikit-learn.org/.

[20] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6645–6649.

[21] Eduard C Groen, Norbert Seyff, Raian Ali, Fabiano Dalpiaz, Joerg Doerr, Emitza Guzman, Mahmood Hosseini, Jordi Marco, Marc Oriol, Anna Perini, et al. 2017. The Crowd in Requirements Engineering: The Landscape and Challenges. *IEEE Software* 34, 2 (2017), 44–52.

[22] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's not a bug, it's a feature: How Misclassification Impacts Bug Prediction. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 392–401.

[23] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[24] Anna Huang. 2008. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, Vol. 4. 9–56.

[25] Qiao Huang, Xin Xia, David Lo, and Gail C. Murphy. 2018. Automating Intention Mining. *IEEE Transactions on Software Engineering* PP, 99 (2018), 1–1.

[26] Nishant Jha and Anas Mahmoud. 2017. Mining User Requirements from Application Store Reviews Using Frame Semantics. (2017), 273–287.

[27] Thorsten Joachims. 1996. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization*. Technical Report. Carnegie-mellon univ pittsburgh pa dept of computer science.

[28] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).

[29] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. 3146–3154.

[30] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[31] Ron Kohavi. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. 1137–1145.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[33] Jonathan K. Kummerfeld, Sai R. Gouravajhala, Joseph Peper, Vignesh Athreya, Chulaka Gunasekara, Jatin Ganhotra, Siva Sankalp Patel, Lazaros Polymenakos, and Walter S. Lasecki. 2019. A Large-Scale Corpus for Conversation Disentanglement. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

[34] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[35] Bin Lin, Alexey Zagalsky, Margaret-Anne D. Storey, and Alexander Serebrenik. 2016. Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing*. 333–336.

[36] Charles X Ling and Chenghui Li. 1998. Data mining for direct marketing: Problems and solutions.. In *Kdd*, Vol. 98. 73–79.

[37] Walid Maalej and Hadeer Nabil. 2015. Bug report, Feature request, or Simply Praise? on Automatically Classifying App Reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)*. IEEE, 116–125.

[38] Andrew McCallum, Kamal Nigam, et al. 1998. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, Vol. 752. Citeseer, 41–48.

[39] Collin Mcmillan, Collin Mcmillan, Collin Mcmillan, and Collin Mcmillan. 2017. Detecting User Story Information in Developer-client Conversations to Generate Extractive Summaries. In *Ieee/acm International Conference on Software Engineering*. 49–59.

[40] Thorsten Merten, Matúš Falis, Paul Hübner, Thomas Quirchmayr, Simone Bürsner, and Barbara Paech. 2016. Software Feature Request Detection in Issue Tracking Systems. In *Requirements Engineering Conference (RE), 2016 IEEE 24th International*. IEEE, 166–175.

[41] Thorsten Merten, Bastian Mager, Paul Hübner, Thomas Quirchmayr, Barbara Paech, and Simone Bürsner. 2015. Requirements Communication in Issue Tracking Systems in Four Open-Source Projects.. In *REFSQ Workshops*. 114–125.

[42] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[43] Itzel Morales-Ramirez, Fitsum Meshesha Kifetew, and Anna Perini. 2017. Analysis of Online Discussions in Support of Requirements Discovery. In *Advanced Information Systems Engineering*. 159–174.

[44] Itzel Morales-Ramirez, Fitsum Meshesha Kifetew, and Anna Perini. 2018. Speech-acts based analysis for requirements discovery from online discussions. *Information Systems* (2018).

[45] Jonas Mueller and Aditya Thyagarajan. 2016. Siamese recurrent architectures for learning sentence similarity. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[46] Fabio Palomba, Mario Linares Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2015. User Reviews Matter! Tracking Crowdsourced Reviews to Support Evolution of Successful Apps. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*. 291–300.

[47] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359.

[48] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C Gall. 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. (2015), 281–290.

[49] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[50] Antônio Mauricio Pitangueira, Paolo Tonella, Angelo Susi, Rita Suzana Pitangueira Maciel, and Márcio de Oliveira Barros. 2017. Minimizing the Stakeholder Dissatisfaction Risk in Requirement Selection for Next Release Planning. *Information & Software Technology* 87 (2017), 104–118.

[51] Germán Poo-Caamaño, Eric Knauss, Leif Singer, and Daniel M German. 2017. Herding cats in a FOSS ecosystem: a tale of communication and coordination for release management. *Journal of Internet Services and Applications* 8, 1 (2017), 12.

[52] Lutz Prechelt. 1998. Early stopping-but when? In *Neural Networks: Tricks of the trade*. Springer, 55–69.

[53] Scrapinghub. 2019. Scrapy. https://scrapy.org/.

[54] Lin Shi, Celia Chen, Qing Wang, and Barry W. Boehm. 2016. Is It a New Feature or Simply "Don't Know Yet"?: On Automated Redundant OSS Feature Requests Identification. In *24th IEEE International Requirements Engineering Conference, RE 2016, Beijing, China, September 12-16, 2016*. 377–382.

[55] Lin Shi, Celia Chen, Qing Wang, Shoubin Li, and Barry W Boehm. 2017. Understanding feature requests by leveraging fuzzy method and linguistic analysis. *automated software engineering* (2017), 440–450.

[56] Emad Shihab, Zhen Ming Jiang, and Ahmed E. Hassan. 2009. On the use of Internet Relay Chat (IRC) meetings by developers of the GNOME GTK+ project. In *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009 (Co-located with ICSE), Vancouver, BC, Canada, May 16-17, 2009, Proceedings*. 107–110.

[57] Emad Shihab, Zhen Ming Jiang, and Ahmed E. Hassan. 2009. Studying the use of developer IRC meetings in open source projects. In *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*. 147–156.

[58] Andrea Di Sorbo, Sebastiano Panichella, Corrado Aaron Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. 2015. Development Emails Content Analyzer: Intention Mining in Developer Discussions (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*. 12–23.

[59] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[60] Bootstrap Team. 2019. Bootstrap. https://getbootstrap.com/.

[61] José Apolinário Teixeira and Helena Karsten. 2019. Managing to release early, often and on time in the OpenStack software ecosystem. *Journal of Internet Services and Applications* 10, 1 (2019), 7.

[62] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2016. Release Planning of Mobile Apps based on User Reviews. In *Proceedings of the 38th International Conference on Software Engineering*. 14–24.

[63] Radu E Vlas and William N Robinson. 2012. Two Rule-based Natural Language Strategies for Requirements Discovery and Classification in Open Source Software Development Projects. *Journal of management information systems* 28, 4 (2012), 11–38.

[64] Yaqing Wang and Quanming Yao. 2019. Few-shot learning: A survey. *arXiv preprint arXiv:1904.05046* (2019).

[65] Andrew Wood, Paige Rodeghero, Ameer Armaly, and Collin McMillan. 2018. Detecting speech act types in developer question/answer conversations during bug repair. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE*. 491–502.

[66] Liguo Yu, Srini Ramaswamy, Alok Mishra, and Deepti Mishra. 2011. Communications in Global Software Development: An Empirical Study Using GTK+ OSS Repository. In *Proceedings of the 2011th Confederated International Conference on the Move to Meaningful Interest Systems, OTM'11*. 218–227.