A Deep Multitask Learning Approach for Requirements Discovery and Annotation from Open Forum

Mingyang Li[1,3], Lin Shi[1,3,*], Ye Yang[4], Qing Wang[1,2,3]
[1]Laboratory for Internet Software Technologies, [2]State Key Laboratory of Computer Sciences,
Institute of Software Chinese Academy of Sciences, Beijing, China;
[3]University of Chinese Academy of Sciences, Beijing, China; *Corresponding author;
[4]Stevens Institute of Technology, Hoboken, NJ, USA;
mingyang@itechs.iscas.ac.cn,shilin@iscas.ac.cn,wq@iscas.ac.cn,yyang4@stevens.edu

ABSTRACT

The ability in rapidly learning and adapting to evolving user needs is key to modern business successes. Existing methods are based on text mining and machine learning techniques to analyze user comments and feedback, and often constrained by heavy reliance on manually codified rules or insufficient training data. Multitask learning (MTL) is an effective approach with many successful applications, with the potential to address these limitations associated with requirements analysis tasks. In this paper, we propose a deep MTL-based approach, DEMAR, to address these limitations when discovering requirements from massive issue reports and annotating the sentences in support of automated requirements analysis. DEMAR consists of three main phases: (1) data augmentation phase, for data preparation and allowing data sharing beyond single task learning; (2) model construction phase, for constructing the MTL-based model for requirements discovery and requirements annotation tasks; and (3) model training phase, enabling eavesdropping by shared loss function between the two related tasks. Evaluation results from eight open-source projects show that, the proposed multitask learning approach outperforms two state-of-the-art approaches (CNC and FRA) and six common machine learning algorithms, with the precision of 91% and the recall of 83% for requirements discovery task, and the overall accuracy of 83% for requirements annotation task. The proposed approach provides a novel and effective way to jointly learn two related requirements analysis tasks. We believe that it also sheds light on further directions of exploring multitask learning in solving other software engineering problems.

KEYWORDS

Requirements discovery, Requirements annotation, Multitask learning, Deep learning

ACM Reference Format:
Mingyang Li[1,3], Lin Shi[1,3,*], Ye Yang[4], Qing Wang[1,2,3]. 2020. A Deep Multitask Learning Approach for Requirements Discovery and Annotation from Open Forum. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3324884.3416627

ASE '20, September 21–25, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-6768-4/20/09…$15.00
https://doi.org/10.1145/3324884.3416627

1 INTRODUCTION

Modern applications become increasingly dependent on knowledge learned from user community to improve and innovate services. In general, such knowledge reveals how users discuss their experience with a specific version/feature of the application, and propose feedback or new features. Dozens of data sources have been identified and used for requirements discovery and analysis, including chat messages, emails, forums, project digests, etc [48]. Such data sources contain large amount of textual data. For example, Cleland-Huang *et al.* reported that forums such as SourceForge and JIRA typically consist of thousands of requirements [8], and Vlas *et al.* reported that 16 projects from SourceForge average 146,716 words just in feature requests [63]. It would be an extremely labor-intensive process for manual extraction of requirements. In addition, such textual documents are frequently noisy, consisting of trivial, non-requirements information, such as social communications, slang, typos, etc [8]. Such noise needs to be appropriately handled in order to support efficient and effective requirements discovery and analysis [50, 62]. Meanwhile, intensive approaches have been proposed to support automatic requirements analysis from user comments and issue reports, employing techniques such as text mining, machine learning, social network analysis, and feedback mechanisms, etc.

For this study, we define two requirements analysis tasks, according to the underlying problem addressed in existing studies. One is requirements discovery (RD), which is a binary-classification task for identifying whether a new document is a valid requirement. The other is requirements annotation (RA), which is a multi-label classification task for annotating the semantic categories of the sentence(s) within a new document. Most existing RD and RA approaches are rule-based or learning-based techniques to explore valuable user requirements. Rule-based approaches rely heavily on a set of heuristic rules pre-codified by human experts [9]. While, learning-based approaches employ the machine learning algorithms, such as Random Forest [4] and convolution neural network [18], to train model for future predictions.Two example rule-based RA approaches include a fuzzy rule-based approach for annotating sentences in feature requests [51], and an ontology-based approach consisting of six levels of classification patterns [62].

There are four major limitations in existing approaches. First, in the RD task, existing studies typically model documents using bag-of-words [49] which vectorizes the texts under the assumption

that the words are independent. It ignores the abundant semantic information containing in requirements descriptions. Second, in the RA task, manually-built rules are labor-intensive and difficult to rebuilt across domains in practices. Third, some methods aim at using supervised training instances on a single task, but unfortunately, they ignore the rich correlation information between the tasks. Forth, the training data for RD and RA is often limited. The issue reports in the repositories are typically large in size. But only a few labeled ones are categorized into requirements types. How to make the maximum use of the few labeled requirements to accurately classify the unlabeled issues becomes an important problem. To address these challenges, this study proposes to adopt deep multitask learning (MTL) approach, which combines the deep learning and MTL techniques, which has led to successes in many applications of machine learning, from natural language processing [55], speech recognition [42], to computer vision [70]. MTL is a subfield of machine learning in which multiple learning tasks are solved at the same time, while exploiting commonalities and differences across tasks [44]. This can result in improved efficiency and prediction accuracy, when compared to training the models separately [6, 58, 72].

In this paper, we present a deep MTL-based approach, DEMAR (**DE**ep **M**ultitask Le**A**rning for **R**equirements Discovery and Annotation), to automated support for RD and RA. DEMAR consists of three main phases: (1) data augmentation phase, for data preparation and allowing data sharing beyond single task learning; (2) model construction phase, for constructing the MTL-based model for requirements discovery and requirements annotation; and (3) model training phase, enabling eavesdropping by shared loss function between the two related tasks. Evaluation results from eight open-source projects show that DEMAR outperforms two state-of-the-art approaches (CNC and FRA) and six common machine learning algorithms, with the precision of 91% , the recall of 83% for RD task, and overall accuracy of 83% for RA task. Specifically, by jointly learning the two tasks, the performance of the discovery task increases 4% of F1, and the annotation task increases 8% of accuracy. In summary, the key contributions of this paper are: 1) DEMAR, a novel paradigm to support multitask learning with two related requirements analysis tasks, including data augmentation, MTL-based model construction, and MTL-based model training phases. 2) A novel deep MTL-based model to enable sharing information through shared layers and task-specific layer for jointly learning RD and RA tasks; 3) A demonstration of the power of MTL in combination with deep learning for RD and RA tasks; 4) The publicly accessible materials[1] to facilitate its application in other contexts.

2 BACKGROUND

This section introduces a motivational example, and the related techniques used in our study.

2.1 A Motivational Example

In practices, a typical requirements process in online open-source communities goes like the following. First, project contributors manually browse through the list of newly opened issues and assign a "feature request" or "enhancement" label when discovering a new requirement. This manual process is very labor-intensive and time-consuming. The RD task aims at predicting whether a new issue is a requirement or not, which can expedite the process of issue labeling. Second, given a list of labeled issues or requirements, Release team review, annotate, and integrate important suggestions from a requirement to a future release. The RA task aims at expediting the process of requirements understanding by providing semantic annotations at sentence level. Thus, the release team members can obtain an initial understanding of the unstructured textual artifacts quickly. They could pay attentions to the relevant information while ignoring the less important parts from the submitted artifacts [51]. Figure 1 illustrates an example of the two related RD and RA tasks on a set of newly opened, unlabeled issue reports in Keras project from Github. The RD task focuses on the entire issue report, and the output is true or false. Meanwhile, the RA task focuses on the sentences in each issue report, and the output is a set of high-level annotations for each sentence, characterizing whether each sentence depicts the intent, benefit, example, explanation, drawback, or trivia aspect of a requirement. Generally speaking, sentences with "intent" and "benefit" annotations reflect more essential requirements, and sentences with "trivia" annotations may be largely ignored. As shown in Figure 1, the identified requirement from the RD task corresponds to seven more sentence-level annotation labels produced from the RA task, with the 2^{nd} and 6^{th} sentences highlighting the major intents of this requirement.

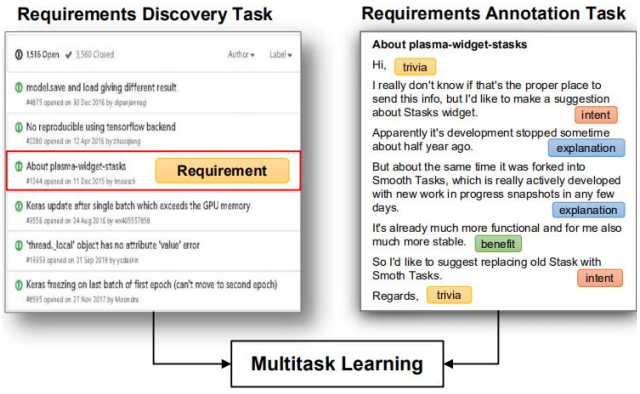


Figure 1: An Example of RD and RA

As introduced earlier, existing RD and RA methods mostly employ rule-based or learning-based approaches to resolve these two tasks independently. But unfortunately, they ignore the rich correlation information between tasks and are often constrained by limited amounts of training data. In fact, the RD task is related to the RA task as they share the same inputs of the textual requirements, and they have shared information relevant to prediction. Specifically, the sequences of sentence annotations may comply with certain patterns when expressing requirements, which can benefit RA task of judging whether an issue report is requirement or not. Taken the requirement in Figure 1 as an example, the user first writes

[1]https://github.com/DEMAR-requirements/DEMAR

an intent-sentence to directly express a request, and then gives some explanations as well as what the benefits the request could deliver. Such "intent-explanation-benefit" pattern might largely exist in requirements rather than in non-requirements like bug reports. Conversely, the category information of an issue report may also benefit to determining the annotations of its sentences. For example, if an issue report is predicted as a requirement, it would be more likely to contain "intent" annotation. If an issue report is predicted as a non-requirement report, it would be more likely to contain "drawback" and "current behavior" annotations. Taking all the above into consideration, we make the first attempt to propose an MTL-based approach that can jointly learn the two tasks, aiming to improve learning efficiency and prediction accuracy for both tasks.

2.2 Multitask Learning

Multitask learning (MTL) is a widely-used approach to inductive transfer, which can be used in combination with machine learning algorithms. Given m related learning tasks, MTL takes the correlation information into consideration, and boosts the performance of each task by sharing the knowledge contained in the m tasks [7, 72]. It achieves this by learning multiple tasks in parallel while using a shared representation between the related tasks to produce models with better generality.

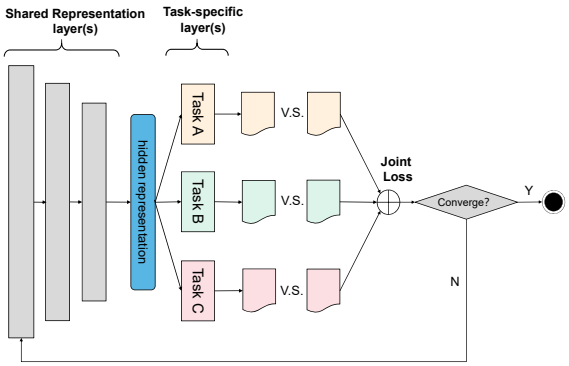


Figure 2: Paradigm of MTL with hard parameter sharing

In the context of supervised deep learning, there are two common ways to perform MTL: *hard parameter sharing* and *soft parameter sharing* of hidden layers. MTL with **hard parameter sharing**, as illustrated in Figure 2, has been widely used in many applications such as natural language processing [55], speech recognition [42], and computer vision [70], etc. This paradigm consists of two parts in constructing the network, i.e., shard presentation layer(s) and task-specific layer(s). Through the shared representation layer(s), the general hidden vectors for input could be obtained. The hidden vectors are used as the input of the task-specific layer(s) for many related tasks. After that, the shared information among tasks is learned by optimizing the model parameters using joint loss. After several iterations, the training stops until the model parameters stabilize. Another paradigm is MTL with **soft parameter sharing**, where each task has its own model with its own parameters. Since the RA and RD tasks share the same inputs of the textual issue reports, they are naturally suitable to the hard parameter mode that sharing the same representation layer(s) and process multiple tasks in the task-specific layer(s). Therefore, we choose the hard parameter sharing to jointly learn the RD and RA tasks.

3 APPROACH

As shown in Figure 3, the proposed approach, DEMAR, consists of three main phases: (1) data augmentation phase, for data preparation and allowing data sharing beyond single task learning; (2) model construction phase, for constructing the MTL-based model for requirements discovery and requirements annotation tasks; and (3) model training phase, enabling eavesdropping by shared loss function between the two related tasks. We will present details on each of the three phases next.

3.1 Data Augmentation

In this step, each issue report will be introduced with augmented information beyond the scope of single task learning.

3.1.1 ***Data Pre-processing.*** A typical issue report may contain two types of information: textual content of the issue (i.e., title and description), and embedded source information (e.g., code snippets, patches, stack traces). In this study, we focus on the textual contents to avoid noise associated with the embedded source information. Specifically, the following steps are conducted to pre-process the issue report data: (1) Retaining textual contents and exclude embedded source information using a text-parsing tool [36]; (2) Splitting issue description text into individual sentences; and (3) Applying standard data cleaning pipeline including special character removal, tokenization, and lowercase conversion. After data pre-processing, each issue report, denoted as D, is considered as an instance for the learning tasks.

3.1.2 ***Instance Label Sharing.*** Designed based on supervised learning, each issue report, D, comes with ground truth labels for RD and RA tasks, respectively. While RD task is a binary-classification task (i.e., Yes/No), DEMAR also employs a set of seven sentence categories for RA task, as summarized in Table 1. The first six are adopted from on Shi *et al.*'s work [51], which are identified from practice guidelines [66]. Other than the six categories, a new category of "current behavior" is introduced, and highlighted in bold in Table 1. The reason for this addition is that, while the majority of Shi *et al*'s categories and their fuzzy rules are still effective, refinement on the explanation category is needed to highlight more semantic meanings in requirements.

At the outset, DEMAR allows to share instance labels across related tasks. Thus, the issue data is explicitly augmented in the following two aspects:

- **Augmented RD data.** By sharing labels, the data used to train the RD model is automatically augmented by including additional information on each sentence. The original RD data is the plain issue reports, and the augmented RD data is the issue report with its sentences being tagged to one of the seven categories in Table 1.
- **Augmented RA data.** Similarly, the data used to train the RA model is augmented by adding the category information of the issue report that each sentence belongs to.

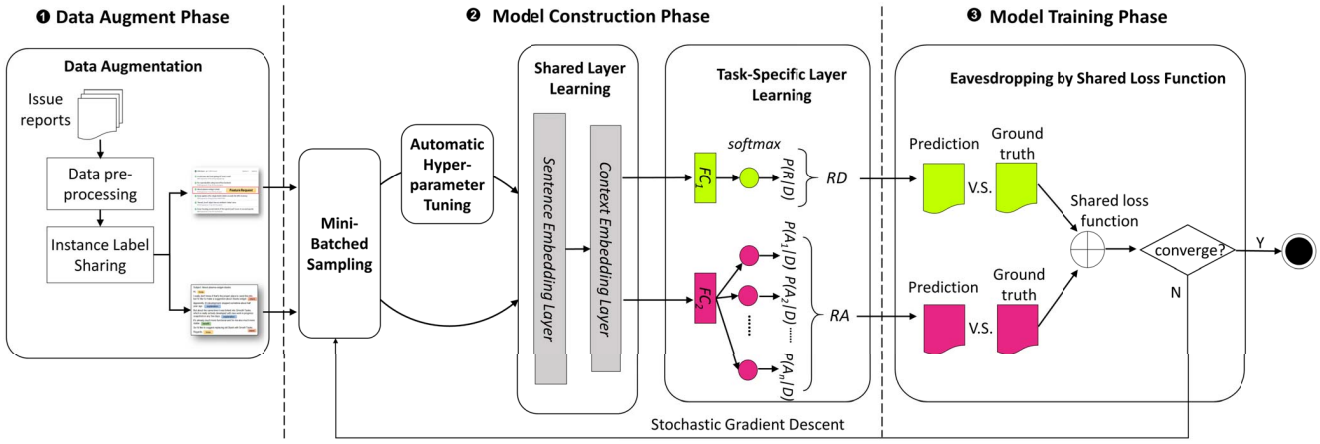


Figure 3: An Overview of DEMAR

Table 1: Description of RA categories.

Category	Definition
Intent (INT)	Ideas or needs to improve system functionalities.
Benefit (BE)	Results or effects if a proposed feature is deliver.
Drawback (DR)	Disadvantages or negative aspects of the current system.
Example (EXA)	Examples or references in support of a proposed feature.
Explanation (EXP)	Detailed information about the proposed feature.
Trivia (TRI)	Information not directly related to a feature or system.
Current Behavior (CB)	The current behavior of a system.

3.2 Model Construction

As illustrated in Figure 3, the model construction phase starts with a mini-batched sampling component, and a two-step MTL component, i.e., shared layer learning and task-specific layer learning, to train the MTL-based model. An elaborated architecture of the MTL-based model is shown in Figure 4. There lower layers - the sentence embedding layer and the context embedding layer- are the globally shared layers for RD and RA tasks. The shared layers take inputs from the mini-batched sampling component, and produce the hidden representation for each input sentence. Then, the task-specific layer is to map the hidden representation into two groups of outputs. Next, we introduce the details of this phase.

3.2.1 ***Mini-Batched Sampling.*** Mini-batching is a very popular technique in machine learning, due to its ability to boost performances and accelerate the training process [39]. Following this convention, DEMAR employs mini-batched sampling in its training process. Rather than sending one training instance into the model at a time, DEMAR equally splits all the samples into batches with the same size (batch_size). All the training samples are divided into $\lceil \frac{number_s}{batch\_size} \rceil$ batches, where $number_s$ is the number of all the training samples. More details on the tuning of the batch_size parameter will be presented in the later section (Section 3.3.2). For one training iteration, each batch containing batch_size instances is sent into the model for training, which could avoid the bias of a single instance.

3.2.2 ***Shared Layer Learning.*** The shared layers consist of two layer: (1) sentence embedding layer where DEMAR embeds each sentence to a semantic vector; (2) context embedding layer where DEMAR encodes the contextual information for each sentence. Following introduces two layers in detail.

Sentence Embedding Layer: This layer aims to embed each sentence into a semantic vector. For a given issue report, we denote all sentences in it as $[s_1, s_2, ..., s_n]$ (n is the number of sentences in the issue report). Traditionally, a sentence is vectorized using a vector space model [47], and represented by the weighted words within the sentence. In this way, the dimension of the sentence vector is the vocabulary size. However, due to the large size of vocabulary, it is very expensive to learn such kind of models. Unlike the traditional vector space model, DEMAR introduces a pre-trained BERT model [19] to embed the sentence. The BERT model used in DEMAR has been pre-trained on a large volume of natural language texts (BooksCorpus [73] containing 800M words and English Wikipedia

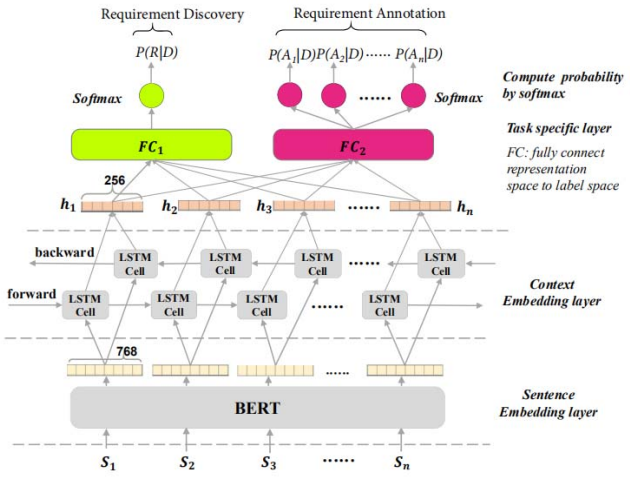


Figure 4: The MTL-based model

containing 2,500M words), and has been successfully adopted in many natural language process tasks such as text classification [32], automatic question answering [20] and named entity recognition [53]. Given a sentence, the BERT model returns the corresponding sentence vector with the dimension of 768 (the default value in the BERT model), which carries the semantic information. Please note that there are different sizes of sentences in issue reports, which leads to different lengths of instances. To ensure the consistency of input length, DEMAR uses *padding* and *truncation* to align input length, which is one of the common practices in deep learning [12]. Due to 90% sampled issue reports (see Section 4.2 for details) have less than 10 sentences, we set the fixed input length n as 10. If an instance has less than 10 sentences, DEMAR adds zero vectors to the end of the instance for padding. If it has more than 10 sentences, DEMAR truncates the ending sentences.

Context Embedding Layer: The purpose is to encode the contextual information of each sentence. Existing rule-based approaches typically capture high-level contextual information and express them using rules, derived based on direct observation from empirical data or expert experience. For example, if a sentence describes an intent, the next sentence is likely to belong to the "benefit". On the contrary, for an issue report, if an "intent" is followed by "benefit" sentence(s), it is likely a requirement. With the consideration, we employ a Bi-LSTM layer to embed the contextual information and enable richer information sharing in MTL. Given the semantic vectors produced by the sentence embedding layer, the context embedding layer learns and converts them into the corresponding vector sequence $[h_1, h_2 ...., h_n]$, where h_i is the hidden representation for the input sentence s_i. The dimension of these hidden vector, denoted as *LSTM dimension*, is a key hyper-parameter. More details of tuning this hyper-parameter are presented in Section 3.3.2. The shared hidden representation is used as the input for the following task-specific layer learning.

3.2.3 ***Task-Specific Layer Learning.*** As shown in Figure 4, the task-specific layer is to map the hidden vector representations produced by the shared layers into two groups of classification outputs, i.e., predictions for the RD and RA tasks respectively.

Classification for the RD task. As RD is a binary-classification task, DEMAR employs an fully connected (FC) layer, which is popularly used as the final layer in deep learning for classification tasks [24, 26, 71].The FC layer takes the hidden vector, i.e., $[h_1, h_2, ..., h_n]$ as the input, and outputs 2-dimension vector corresponding to the two values, indicating whether the issue report is a requirement or not. Then, using the softmax function [75], DEMAR normalizes the 2-dimension vector into a probability score $P(R \mid D)$, and classify whether the issue report D is a requirement.

Classification for the RA task. As introduced earlier, RA is a multi-label classification task to assign each sentence into the annotation categories, as illustrated in Table 1. Similarly, DEMAR leverages the FC layer to map each hidden representation into one of the seven categories. The output of the FC layer is a vector $[P(A_1 \mid D)$, $P(A_2 \mid D)$, ..., $P(A_n \mid D)]$, where $P(A_i \mid D)$ $(1 \leq i \leq n)$ is also a vector of probability across all seven sentence categories for an input sentence s_i of the issue report D. The probability vector $P(A_i \mid D)$ is then processed through the softmax function to generate the corresponding sentence labels of the issue report D.

3.3 Model Training

3.3.1 ***Eavesdropping by shared loss function.*** One of the main advantages of MTL is that it allows the model to eavesdrop when learning from related tasks. For the RD task, the loss is calculated by the cross-entropy of using the following equation:

$$Loss_{RD} = -y_i \cdot \log(P(R \mid D)) \tag{1}$$

where y_i is the ground-truth requirements label for each instance D, and $P(R \mid D)$ is the predicted probability of the issue report being a requirement.

Similarly, the loss of the RA task is calculated by the average cross-entropy using the following equation:

$$Loss_{RA} = \frac{1}{N} \sum_{j=1}^{N} -y_j \cdot \log P(A_j \mid D) \tag{2}$$

where N is the number of sentences in the instance D (covering requirements and non-requirements issue reports), y_j is the true sentence label of the sentence s_j, $P(A_j \mid D)$ is the predicted classification of each sentence s_j. Then DEMAR employs a principled, shared loss function to take the weighted sum of the two individual losses, using the following equation:

$$Loss = \lambda Loss_{RD} + (1 - \lambda) Loss_{RA} \tag{3}$$

where λ is harmonic parameter in the range (0, 1). In this study, the hyper-parameter λ is determined by the automatic hyper-parameter tuning introduced in Section 3.3.2. With the loss function, the training is conducted in an iterative manner. In each iteration, by sharing the loss, one task could eavesdrop the information from the other, and learn some features which are difficult to learn from itself but easy from the other task.

Then, all mini-batches are fed into the model one by one, and the loss is calculated using the loss function *Loss*. Next, with the loss *Loss* being back propagated through the model layers, the model parameters in the task-specific layer and context embedding layer are updated using the Stochastic Gradient Descent algorithm [41]. The training process repeats the above steps and stops if the loss function does not decrease obviously on 10 consecutive batches.

3.3.2 ***Hyper-parameter tuning strategy.*** Previous studies have shown that hyper-parameter tuning is important for a prediction model to achieve better performance [10]. There are three hyper-parameters in DEMAR, i.e., the dimension of the hidden vector in MTL model (refer to Section 3.2.2), batch_size (refer to Section 3.2.1), and the harmonic factor λ for the loss function (refer to Section 3.3.1). DEMAR adopts a greedy strategy to tune these hyper-parameters. Given n hyper-parameters $\{P_1, P_2, ..., P_n\}$ and the corresponding n sets of candidate tuning values $\{V_1, V_2, ..., V_n\}$, we perform n iterations of automated tuning. In each iteration, we randomly select one P_i without replacement, enumerate all its corresponding candidate values v_j in $V_i = \{v_1, v_2, ..., v_m\}$, choose the relatively good performances, as the determined value of P_i. More details and results on hyper-parameter tuning will be presented in Section 4.3.3 and Section 5.3, respectively.

4 EXPERIMENTAL DESIGN

This section describes the research questions, the subject dataset, the designed experiments to be conducted, the baseline approaches,

and the metrics for performance measurement, in order to verify the performance of the proposed DEMAR approach.

4.1 Research Questions

Three research questions are formulated as following:

RQ1: **What are the benefits of DEMAR compared to single task learning?** This question aims at evaluating the effectiveness of the multitask learning approach in requirements discovery and annotation. Specifically, in order to investigate the benefits of multitask learning approach, we conduct experiments to compare DEMAR with its two single task modes, respectively.

RQ2: **How does the performance of DEMAR compare to existing approaches?** This question aims at evaluating the advantages of DEMAR. Specifically, we compare DEMAR with existing state-of-the-art approaches, as well as a set of widely-adopted machine learning classifiers.

RQ3: **How sensitive is the performance of DEMAR to the hyper-parameter values?** This question aims at evaluating the performance concerning different values of its hyper-parameters, and tuning towards improved performance. Specifically, we employ a greedy strategy for automatic hyper-parameter tuning, in order to determine optimal values of hyper-parameters for achieving promising results.

4.2 Dataset

The dataset used in this study contains issue report data extracted from eight open-source projects, as summarized in Table 2. Specifically, we construct our RD and RA datasets and label ground truth, through the following steps: (1) For each project, we randomly retrieve the equal amount of positive instances (issue reports which are labeled as "feature request" or "enhancement" in project repositories) and negative instances, and remove the instances without any natural language descriptions; (2) We manually inspect the labels, and correct 29 mis-classified requirements from negative instances into positive instances; (3) For each issue report, we manually label the sentences with the pre-defined seven categories. To guarantee the correctness of the labeling results, an inspection team with two senior researchers and two PhD students jointly work in this process. Once different labeling opinions arise, the final result is determined based on a team discussion and majority voting mechanism. In total, there are 1,067 issue reports (IR), including 573 requirements (R), and 494 non-requirements (NR) issues in the RD and RA dataset. The RA dataset includes the sentences taken from the 1,067 issue reports, as well as their corresponding categories introduced earlier in Table 1.

Table 2: The Statistics of RD Dataset and RA Dataset

project	#IR	RD dataset	RA dataset						
		#R/NR	#INT	#CB	#BE	#DR	#EXA	#EXP	#TRI
ActiveMq	125	65/60	56	80	16	18	13	116	29
Archiva	83	42/41	65	63	3	8	6	70	7
Aspectj	191	97/94	126	147	22	20	19	301	44
HDFS	162	104/58	112	90	34	20	11	132	8
Hibernate ORM	175	89/86	108	99	18	6	13	259	21
Log4j	120	61/59	67	84	18	6	32	150	15
Mopidy	114	65/49	56	55	18	3	15	71	9
SWT	97	50/47	50	74	4	5	9	120	12
TOTAL	**1067**	**573/494**	**640**	**692**	**133**	**86**	**118**	**1219**	**145**

4.3 Experiment Design

4.3.1 Benefit analysis of MTL (RQ1). To answer RQ1, we first configure DEMAR into three working modes. One is multitask learning mode (i.e., DEMAR). Two are its single task learning modes, i.e., single RD task ($DEMAR_{SRD}$) and single RA task ($DEMAR_{SRA}$). Then, we conduct experiments to compare the performances among the three modes. To train the two single task learning modes, we use the single task view of the dataset, i.e., RD-view with only RD labels and RA-view with only RA labels, to train the two corresponding $DEMAR_{SRD}$ model and $DEMAR_{SRA}$ model, respectively. Then, we conduct 10-fold cross validation on the RD and RA dataset. Finally, the performances are compared among the three modes to investigate whether multitask learning mode outperform either single task learning mode. Note that, for the RA task, the 10-fold cross validation is conducted on the whole dataset, i.e., including sentences in both requirements and non-requirements issues. However, in order to evaluate performances on RA task, we only calculate the evaluation metrics on the subset of sentences in requirements. This is to avoid the side-effect RA prediction of DEMAR on non-requirements instances, and we will discuss more in Section 6.2.

4.3.2 Baseline Comparison (RQ2). To answer RQ2, we select a set of eight comparison baselines, including two state-of-art approaches, i.e., CNN-based Classifier (CNC) and Rule-based Classifier (FRA), as primary baselines, and six popular text classification approaches to compare their performances with DEMAR. Specifically, CNC is used to compare the performance of DEMAR in the RD task, and FRA is used to compare in RA task. In addition, we implement six additional classifiers to support both RD and RA tasks, so that they can be used to compare with DEMAR in both RD and RA tasks. The details of the baselines are presented in Section 4.4. The performance for each baseline is evaluated using 10-fold cross validation. Similar to RQ1, in order to evaluate performances on the RA task, we only calculate the evaluation metrics on the subset of sentences in requirements.

4.3.3 Hyper-parameter Sensitivity Analysis (RQ3). As introduced in Section 3.3.2, there are three hyper-parameters in DEMAR. To answer RQ3, we design a step-wise tuning experiment to automatically tune their values respectively. First, we tune the dimension of hidden vector, with the other two hyper-parameters fixed at default values (i.e., 64 for batch_size and 0.5 for λ). Specifically, we enumerate the dimension of hidden vector h_i with values in the range of [32, 64, 96, 128, 160, 192, 224, 256, 512, 1024], and train the model. Among all the candidate dimensions, we choose the one whose corresponding model achieves the best performance on the test set. Second, tune the batch size hyper-parameter. Following a similar enumeration approach, we experiment each value in the set [1, 2, 4, 8, 16, 32, 64, 128, 256, 512], with the optimal dimension of hidden vector identified from the previous step, as well as the default value of λ. Among all the candidate batch sizes, we choose the one whose corresponding model achieves the best performance on the test set. Last, tune λ. The optimal values for the other two hyper-parameters are applied in this step, and we experiment values of λ from the range of 0.1 to 0.9 with an increased delta of 0.1 in each round. The value which leads to a corresponding model that achieves the best performance will be selected as the optimal

value. Following this greedy strategy, the performance of DEMAR under different hyper-parameter combinations is evaluated, and the hyper-parameter values producing the best performance are selected and used in DEMAR.

4.4 Baselines

To evaluate the performance of DEMAR, we choose the following approaches as comparison baselines.

CNN-based Classifier (CNC). This is the latest state-of-art approach for intent mining, proposed by Huang *et al.* [18]. CNC is based on the convolution neural network (CNN) to automatically classify sentence label to be one of seven pre-defined intent categories ("Information Giving", "Information Seeking", "Feature Request", "Solution Proposal", "Problem Discovory", "Aspect Evaluation", and "Meaningless"). If a sentence is classified into the "Feature Request" category, the issue report which the sentence belongs to is considered as a requirement. It is used as the primary baseline for comparison in the RD task. In our experiment, we predict the label of each sentence using the trained model [59], and classify an issue report as a requirement if at least one of its sentences has a label of "Feature Request".

Rule-based Classifier (FRA). This is the state-of-art approach to requirements annotation, proposed by Shi *et al.* [51]. FRA is built upon a fuzzy method and natural language processing to automatically classify the content of feature requests according to six pre-defined categories, including "Intent", "Benefit", "Drawback", "Example", "Explanation", and "Trivia". It is used as the primary baseline for comparison in the RA task. We apply the FRA classifier by using the source code provided by the authors.

Machine-learning-based Classifiers. Besides the above two primary baselines, we additionally introduce six text classification algorithms as baselines to provide more comprehensive perspectives of comparison. These classification algorithms are also widely used in previous studies [13, 18, 23]. The six classification algorithms are applied for RD and RA tasks separately. *Logistic Regression (LR)* [28] is a statistical method that is employed to conduct appropriate regression analysis when the dependent variable is binary. *Naive Bayesian (NB)* [33] 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)* [30] is an ensemble machine learning method that is constructed with several trees, and each tree can contribute to the final classification result. *Support Vector Machine (SVM)* [60] is a supervised machine learning algorithm that can be used for both classification and regression purposes, which is a non-binary linear classifier. *TextCNN* [25] is the convolutional neural network for text classification, and it is a useful deep learning algorithm for many natural language process tasks such as sentiment analysis and question classification. *TextRNN* [26] is the recurrent neural network for text classification, where the connections between nodes form a directed graph along a text sequence.

4.5 Evaluation Metrics

We use four commonly-used measurements to evaluate the performance, i.e., *Accuracy, Precision, Recall, F1.* (1) *Accuracy* is the proportion of sentences which are correctly classified among all sentences. It is used as the overall evaluation metric for the RA task. (2) *Precision*, which refers to the ratio of the number of correct predictions to the total number of predictions; (3) *Recall*, which refers to the ratio of the number of correct predictions to the total number of samples in the golden test set; and (4) *F1*, which is the harmonic mean of precision and recall.

5 RESULTS

This section presents the evaluation results in order to answer the research questions.

5.1 Benefit of MTL (RQ1)

Figure 5 (a) illustrates the performance comparison between DEMAR and its single task learning mode ($DEMAR_{SRD}$) for RD task, in terms of precision, recall, and F1. It is clear that DEMAR outperforms its single task mode across all metrics. The improvements on the precision, recall, and F1-score are 5%, 4%, and 4% respectively. These increases are directly attributed to the benefits associated with the internal jointly learning setup of DEMAR, i.e., learning RD task simultaneously with a counterpart RA task.

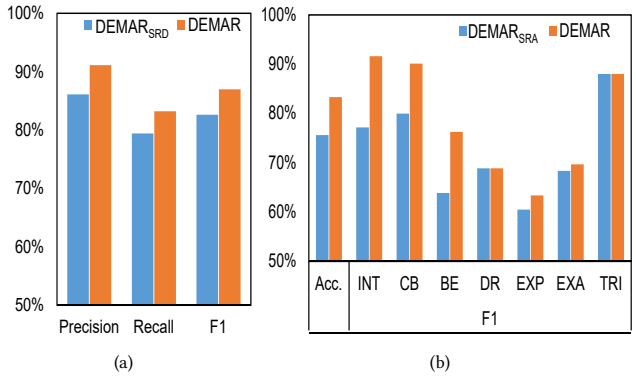


Figure 5: Comparison of different learning modes

Figure 5 (b) shows the performance comparison results between DEMAR and its single task learning mode, i.e., $DEMAR_{SRA}$. As introduced earlier, the RA task is a multi-label classification task containing seven categories of sentences. Therefore, we compare the overall accuracy (Acc.) and F1 across the seven sentence categories, as indicated in the horizontal axis. Significant increases in the performance of DEMAR can be observed in 4 of the 8 metrics, i.e., accuracy and F1 in three categories including "Intent (INT)", "Current Behavior (CB)" and "Benefits (BE)". More specifically, the overall accuracy is increased by 7.7%, i.e., from 75.6% to 83.3%, and the F1 in the three categories including "Intent (INT)", "Current Behavior (CB)" and "Benefits (BE)" are increased by 14%, 12%, and 10%, respectively. Compared to the three categories, the F1 values for the "Explanation (EXP)" and "Example (EXA)" categories have

less improvement, and remains the same for the other two categories, i.e., "Drawback (DR)" and "Trivia (TRI)". The reason of better performances in INT, CB, and BE categories is that, these categories are more discriminative between requirement and non-requirement reports. Thus, incorporating the prediction results from the RD task would contribute potential confidences to annotate those tags in the RA task by joint learning.

Answers to RQ1: Compared with its alternative single task learning modes, the multitask learning mode of DEMAR can achieve higher performance. On average, DEMAR increases the performance of the single RD task by 4% in F1, and the performance of the single RA task by 8% of in overall accuracy.

5.2 Baseline Comparison (RQ2)

As introduced in Section 4.4, two sets of baseline approaches are selected for comparing with the proposed DEMAR in RD and RA tasks. We will present the comparison results next.

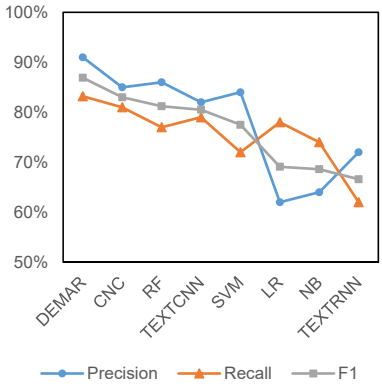


Figure 6: Comparison in RD task

Performance comparison in RD task: Figure 6 shows the precision, recall, and F1 achieved by DEMAR, compared with the state-of-art approach CNC, and six other popular classifiers. It is easy to observe that DEMAR achieves the highest performance in all three metrics. CNC is the second-best performer, only its precision slightly worse than the random forest (RF) classifier. The results indicate that DEMAR outperforms all seven baseline approaches, including the state-of-art CNC approach and six popular text classifiers. This is attributed to the benefits of multitask learning, i.e., by leveraging additional information that can only be learned in a jointly learning setting. More specifically, the model learns a better-shared representation (the hidden vector h_i) by multitask learning, which implicitly boosts the performance of the RD task, even compared with the latest state-of-art CNC approach.

Performance comparison in RA task: Table 3 summarizes the comparison results of overall accuracy and F1 across seven categories among DEMAR, FRA, and six popular classifiers. It shows that DEMAR achieves the highest overall accuracy, i.e., with a 1.8% increase than that of FRA baseline. In addition, DEMAR also has the highest performances in four categories including "Intent (INT)", "Current Behavior (CB)", "Explanation (EXP)", and "Trivia (TRI)". It is observed that FRA also achieves relatively good performance

Table 3: Comparison in RA task

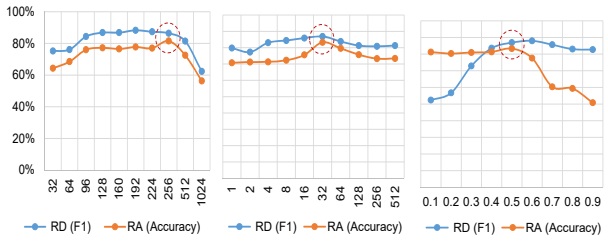
Approach	F1							Accuracy
	INT	BE	DR	EXP	EXA	TRI	CB	
DEMAR	**91.4%**	76.2%	68.9%	**87.2%**	69.7%	**88.0%**	**91.5%**	**83.3%**
FRA	87.5%	**84.9%**	**83.0%**	83.0%	**80.5%**	82.0%	NA	81.5%
LR	70.5%	48.6%	56.8%	57.4%	71.0%	79.2%	77.4%	68.6%
NB	70.0%	45.7%	57.6%	46.7%	73.5%	79.0%	76.8%	69.4%
RF	75.9%	64.4%	72.5%	54.0%	60.4%	80.0%	78.5%	72.0%
SVM	75.8%	59.4%	69.4%	57.4%	71.8%	74.1%	77.9%	75.7%
TEXTCNN	74.9%	51.5%	61.7%	67.2%	54.9%	83.8%	78.0%	73.2%
TEXTRNN	57.7%	38.0%	58.5%	61.0%	53.4%	82.0%	47.3%	58.3%

in the RA task. This is because it is a rule-based approach, which may be more effective, but building those fuzzy rules require extra involvement of requirements experts. Considering DEMAR is a fully automated, learning-based approach, we believe the comparison result demonstrates DEMAR is sufficiently effective in comparing with and potential as replacement of manual expert effort. Compared with the other six learning-based classifiers, DEMAR is 7.6%-25% higher in terms of accuracy. Such improvement may benefit from the data augmentation and eavesdropping mechanism, thus making the learning process more effective.

Performance comparison across eight individual projects: Table 4 summarizes the performance comparison results across eight individual projects, among DEMAR and all baselines. The comparison is conducted in the RD and RA tasks, respectively. The "approach" column lists the corresponding baselines, as mentioned above and in Section 4.4. For RD task, DEMAR achieves the highest F1 in most (5/8) of the individual projects, and the highest average of F1 (85%) among all the eight projects. For RA task, DEMAR achieves the highest accuracy (83%) on average, and also the highest accuracy on 4 of the 8 projects, while FRA is the highest on other three project, and TextCNN is the highest on one project.

Answers to RQ2: On average, DEMAR outperforms all selected baselines in RD and RA tasks, which indicates the advantages of the proposed approach.

5.3 Hyper-parameter Sensitivity Analysis(RQ3)



(a) LSTM Dimension (b) Batch size (c) Lamda

Figure 7: Results of hyper-parameter tuning

Figure 7 (a) shows the variation of DEMAR's performance under different values for the first hyper-parameter, i.e., the dimension of the hidden vector. It is clear that DEMAR produces the best performance for both tasks, when the value of this hyper-parameter h_i is set as 256. Therefore, this is identified as the optimal value

Table 4: Summary of baseline comparison across eight projects

Task Name	Approach	ActiveMq	Archiva	Aspectj	HDFS	Hibernate	Log4j	Mopidy	SWT	Avg.
RD (F1)	*DEMAR*	**84%**	72%	**78%**	**88%**	**95%**	83%	**94%**	82%	**85%**
	CNC	80%	65%	73%	85%	89%	85%	88%	**83%**	81%
	LR	71%	72%	55%	80%	82%	80%	75%	73%	74%
	NB	69%	74%	52%	70%	75%	70%	73%	79%	70%
	RF	75%	**77%**	75%	83%	82%	**86%**	84%	68%	79%
	SVM	79%	**77%**	76%	85%	92%	78%	82%	75%	81%
	TEXTCNN	78%	60%	68%	80%	85%	83%	85%	80%	77%
	TEXTRNN	65%	63%	62%	73%	76%	84%	61%	66%	69%
RA (Accuracy)	*DEMAR*	80%	**82%**	79%	**88%**	86%	76%	**82%**	**88%**	**83%**
	FRA	**83%**	77%	**80%**	85%	**87%**	72%	73%	86%	80%
	LR	67%	52%	69%	72%	64%	59%	66%	79%	66%
	NB	62%	58%	60%	80%	59%	55%	61%	72%	63%
	RF	78%	70%	65%	77%	68%	68%	78%	75%	72%
	SVM	82%	68%	70%	82%	59%	57%	69%	79%	71%
	TEXTCNN	72%	70%	79%	77%	65%	**79%**	76%	87%	76%
	TEXTRNN	53%	55%	62%	70%	72%	65%	60%	72%	64%

of h_i, which is used in training DEMAR. Figure 7 (b) shows the variation of DEMAR's performance under different values for the second hyper-parameter, i.e., batch_ size. It is observed that the value of 32 leads to the highest performances of DEMAR on both tasks, we chose 32 as the value of the batch_size parameter. Figure 7 (c) shows that the imbalanced sharing of the two loss functions ($Loss_{RD}$ and $Loss_{RA}$) could result in performance decreasing on either RA task or RD task. The curves show that if the λ is smaller than 0.5, the performance of the RD task decreases. If the λ is larger than 0.5, the performance of the RA task decreases. It indicates that in this study, a balanced weight sharing loss function could achieve the best performance on both related tasks. Therefore, we choose the value 0.5 as the parameter value of λ. Finally, the combination of the three values is selected as the tuned parameter values and used to train DEMAR, whose promising performance has been demonstrated in previous sections for answering RQ1 and RQ2.

Answers to RQ3: DEMAR employs a greedy strategy and enables automatic tuning-up of its hyper-parameters, which enables the identification of the best parameter values to achieve its promising performance.

6 DISCUSSION

6.1 Advantages of Deep MTL

Results in the previous section demonstrate that, leveraging deep MTL, DEMAR outperforms existing baselines as well as its single task learning modes, in both requirements analysis tasks (i.e., RD and RA). This confirms findings in most studies on MTL's applications [6, 72]. Besides the explicit data augmentation phase in DEMAR, we believe that other implicit augmentation characteristics of MTL also contributes to the increased prediction performance. First, the shared layer learning, as detailed in Section 3.2.2 and Section 3.2.3, allows simultaneously processing related tasks and introducing extra information to each single task. Second, considering that there are different noise patterns in RD and RA dataset, DEMAR can leverage the MTL-based model to mitigate and balance extreme noise and focus more on shared features that matter. Third, the principled loss function in DEMAR allows the MTL model to eavesdrop the information from other tasks and learns features which are easy to capture by other tasks but difficult by itself. All these characteristics of MTL jointly contribute to the increased performance when simultaneously learning RD and RA classification tasks.

6.2 Side-effect Prediction

Existing RA methods focus on requirements annotation, therefore, only take requirements (R) as inputs. Unlike these, DEMAR can also take non-requirements (NR) types of issues as inputs, and produce a set of predicted RA labels. In another word, even if an issue is a NR, i.e., a negative RD instance, DEMAR can still learn and generate its sentence-level annotations. This kind of side-effect prediction may sound chaotic or confusing, as NRs are not valid requirements and can not get sentence-level RA labels. (Recall in Section 4.3, this study only takes requirements instances into account when evaluating the performance of RA task.)

However, there are certain frequent cases that such side information may be beneficial to developer activities such as bug resolution or information sharing. Consider the large variety of topics developers/users are communicating through issue repositories. Besides submitting a requirement, they often discuss other types of NR issues, such as reporting a bug, fixing a bug, asking for help, scheduling a task, etc [15, 74]. Based on our previous observations and experiences from the eight open-source projects, the annotation categories in DEMAR, as listed in Table 1, seems widely generalizable in covering and representing sentences in NR issues. For example, most bug reports contain sentences describing the "current behavior", "drawback", or "intent" (a.k.a expected behavior); typical ask-for-help issues may contain sentences related to some "example" and/or "explanation". In these cases, the side-effect prediction in DEMAR can be used as issue annotation (IA) to highlight important sentences with corresponding labels, and improves the efficiency in issue resolution.

We conduct a further analysis to compare and understand the overall performance of DEMAR in not only RA, but also IA. In this analysis, we take into consideration of the side-effect prediction of the NR instances. More specifically, we first calculate the F1 and accuracy using NR instances from the test data, and then do the

Table 5: Performance with side-effect prediction

TASK	F1							Accuracy
	INT	BE	DR	EXP	EXA	TRI	CB	
RA (R)	91.4%	76.2%	68.9%	87.2%	69.7%	88.0%	91.5%	83.3%
IA (NR)	83.6%	72.9%	88.6%	86.2%	72.5%	91.0%	92.1%	83.5%
IA (R+NR)	88.7%	75.3%	81.3%	86.9%	70.2%	89.3%	91.7%	83.4%

same using both R and NR instances from the test data. As shown in Table 5, DEMAR can achieve 83.6% and 92.1% F1 on the "Intent" and "Current Bahevior" sentences respectively. Besides, DEMAR also achieves relatively good performances on other types, with the overall accuracy of 83.5%. Moreover, the F1 of "Drawback" increases 20% when applying on the NR dataset, mainly due to there are more "Drawback" sentences in bug reports. The results indicate that DEMAR has the potential to benefit the issue resolution process.

6.3 Adaptability for Different Data Sources

Although the evaluation of DEMAR is only conducted in RD and RA tasks using issue reports, the potential to adapt and apply it to handling other data sources is abundant. No matter what sources or format the data comes from, people tend to express their desired features in a relatively consistent way. For example, people frequently use similar expressions such as "need implement sth. in next release" and "sth. will be a solution/improvement", to indicate requirements in other resources such as chat messages, development emails, etc. The proposed DEMAR approach can be adapted to other data sources containing similar representative requirements. Besides, DEMAR can also be applied to handle different languages since DEMAR does not make use of language-related features to train the model. When switching to other languages, DEMAR users only need to apply additional pre-processing according to the specific language, e.g., switch to the BERT trained by the specific language corpus.

6.4 Relevance to Industry Practices

As a fully automated approach, DEMAR can be applied in support different issue review and resolution scenarios. Our ongoing work includes the implementation of DEMAR and develops a dialog bot to continuously monitor Github issue reports, and automatically analyze, classify, and alert developers about important requirements. Such automated tool can be easily integrated with typical workflow of software development teams.

6.5 Extension to Other MTL Applications

Current consensus is that MTL is applicable if there is considerable understanding of task relatedness [6], i.e., their similarity, relationship, etc. The authors believe that, in the field of software engineering (SE), there are a few related tasks that can readily take advantage of MTL.

Bug Prediction Analysis. Bug prediction is one of the classic SE topics, and existing research has proposed intensive methods, models, and tools to address total bug prediction [17, 29, 46, 67], bug classification [11, 16, 57], bug triage [3, 5, 21, 21, 27, 56], bug fixing effort prediction [69], and so on. In these related tasks, textual bug description documents are common key inputs. Nonetheless, each different type of tasks may require additional information, which can be shared in a multitask learning setting to avoid individual representation bias and improve joint learning performance. This would be an potential topic for further exploration.

Commit Message Analysis. Another possible extension is to explore MTL in commit message analysis, which is essential to many SE tasks such as commit classification [45, 68], commit-issue linking [43, 54], commit review recommendation [1, 14, 65], etc. In this case, the common textual inputs are commit messages, followed by other commit meta-data such as author, files included, and the commit diff, etc. Similarly, the SE field has developed intensive understanding of each individual task, using most of the leading machine learning techniques, the additional exploration of MTL seems to be promising.

6.6 Limitations

Nonetheless, there are several limitations to this study. The first limitation is the generalizability of the proposed approach. It is only evaluated using eight open-source projects, which might not be representative of closed-source projects and other open-source projects. The results may be different if applied to other projects. However, the dataset comes from four communities in eight different fields. The variety of projects relatively reduce this threat.

The second limitation may come from data pre-processing to remove the code snippets, patches, stack trace, and other technical information from the natural language text. It is possible that the natural language content is also removed from the text, which leads to errors when evaluating the actual performance. To mitigate this, we use the state-of-the-art tool [36] whose performance could reach 82% F1, which relatively alleviates the threats.

The third limitation relates to the fixed instance length. DEMAR sets instance length as the fixed value 10, and truncates the ending sentences if a instance has more than 10 sentences. It means that DEMAR would discard the predictions to the truncated sentences in the RA task. However, nearly all the instances have more than 10 sentences in our dataset. Thus, the not-available predictions caused by truncation could be a small minority. Furthermore, when applying our approach into other dataset, the instance length could be variable and determined by statistics to minimize the threat.

The forth limitation relates to the suitability of evaluation metrics. We utilize accuracy, precision, recall, and F1 to evaluate the performance, in which we use the document labels and sentence labels that are manually labeled as ground truth when calculating the performance metrics. The threats can be largely relieved as all the instances are reviewed and only the instances which reach the agreement on are kept in our dataset.

7 RELATED WORK

Our work is related to previous studies that focused on (1) requirements discovery; and (2) requirements annotation. We briefly review the recent studies in each category.

7.1 Requirements Discovery

This section reviews two typical categories of approaches (rule-based approaches and learning-based approaches) for automated requirements discovery.

7.1.1 Rule-based approaches. Di Sorbo *et al.* [52] 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". They codified 36 linguistic rules for discovering feature requests from emails, e.g., [someone] wants to have [something]. Morales-Ramirez *et al.* [37, 38] identified requirements-related information in OSS issue discussion using 20 speech-act rules supported by natural language processing techniques. Vlas and Robinson [64] proposed a grammar-based design of software automation for the discovery and classification of natural language requirements found in open source projects repositories. Previous rule-based approaches highly rely on the involvement of human expertise, which is labor-intensive and difficult to rebuild in practice. While our work focuses on providing an advanced learning-based solution to fully automated the discovery task.

7.1.2 Learning-based approaches. Arya *et al.* [4] 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. Rodeghero *et al.* [34] 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. Merten *et al.* [35] 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. Antoniol *et al.* [2] 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. Maalej and Nabil [31] 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 [9, 22, 40, 61]. Huang *et al.* [18] found that Di Sorbo's work 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. In summary, most learning-based approaches rely on a set of predefined features to train machine learning models for specific tasks. While Huang *et al.* [18] took the first attempt to utilize deep learning techniques to avoid explicitly extract features, and yielding significant improvements against Di Sorbo *et al.*'s rule-based approach and other nine commonly-used classification approaches. Our work differs from existing learning-based approaches in that, instead of using supervised training objectives on a single task, we focus on exploring and utilizing the rich correlation information between RD and its related RA tasks.

7.2 Requirements Annotation

Shi *et al.* [51] 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 analyzing real intents of feature requests, which can also expedite the process of feature requests understanding. Our work extends their categories by distilling "Current Behavior" from the "Explanation" category. Moreover, we proposed a multitask learning approach that can automated classify feature-request sentences in a better performance. Vlas *et al.* [62] defined six levels of classification patterns based on ontology rules, e.g., they used a subject-action-object assertion to extract operability and expandability requirements. Although the rules worked well with the 16 projects taken from Sourceforge, it requires new rules to work effectively with new datasets. Moreover, The ontology rules are codified by human expertise, which is labor-intensive and difficult to rebuild in practice.

In summary, existing requirements annotation studies are all rule-based solutions, our work provides an effective learning-based solution by jointly learning the RA task with the RD task that results in better generalization performance than learning separately.

8 CONCLUSION AND FUTURE WORK

In this paper, we proposed a deep multitask learning approach, DEMAR, for requirements discovery and annotation. DEMAR consists of three main phases: (1) data augmentation phase, for data preparation and allowing data sharing beyond single task learning; (2) model construction phase, for constructing the MTL-based model for requirements discovery and requirements annotation tasks; and (3) model training phase, enabling eavesdropping by shared loss function between the two related tasks.

Evaluation results from eight open-source projects show that: (1) DEMAR outperforms all state-of-the-art approaches, i.e., with a precision of 91% and a recall of 83% for requirement discovery task, and an overall accuracy of 83.3% for requirement annotation task; (2) By jointly learning the two tasks, the performance of discovery task increases 4% of F1, and annotation task increases 8% of accuracy;

DEMAR provides a novel and effective way of learning two related requirements analysis tasks. We believe that it also sheds light on further directions in exploring the application of multitask learning in solving other related software engineering problems. Future directions include, but not limited to, further evaluation DEMAR with more intensive data, implementation and integration of DEMAR prototype with modern development environment such as Github; applying and extending MTL-based approaches to not only requirements tasks, but other types of software engineering tasks, e.g., issue analysis, and so on.

ACKNOWLEDGMENT

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

REFERENCES

[1] Toufique Ahmed, Amiangshu Bosu, Anindya Iqbal, and Shahram Rahimi. 2017. SentiCR: a customized sentiment analysis tool for code review interactions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*. 106–111.

[2] 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.

[3] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug?. In *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*. 361–370.

[4] 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.

[5] Gerald Bortis and André van der Hoek. 2013. PorchLight: a tag-based approach to bug triaging. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. 342–351.

[6] R Caruana. 1997. Multitask Learning. *Machine Learning* 28, 1 (1997), 41–75.

[7] Rich Caruana. 1998. Multitask Learning. In *Learning to Learn*. 95–133.

[8] 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.

[9] 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.

[10] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Inf. Softw. Technol.* 76 (2016), 135–146. https://doi.org/10.1016/j.infsof.2016.04.017

[11] Baljinder Ghotra, Shane McIntosh, and Ahmed E. Hassan. 2015. Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. 789–800.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press.

[13] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann.

[14] Christoph Hannebauer, Michael Patalas, Sebastian Stünkel, and Volker Gruhn. 2016. Automatically recommending code reviewers based on their expertise: an empirical comparison. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*. 99–110.

[15] 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.

[16] LiGuo Huang, Vincent Ng, Isaac Persing, Ruili Geng, Xu Bai, and Jeff Tian. 2011. AutoODC: Automated generation of Orthogonal Defect Classifications. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*. 412–415.

[17] Qiao Huang, Xin Xia, and David Lo. 2019. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empirical Software Engineering* 24, 5 (2019), 2823–2862.

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

[19] huggingface. 2020. https://github.com/huggingface/transformers.

[20] Eric Hulburd. 2020. Exploring BERT Parameter Efficiency on the Stanford Question Answering Dataset v2.0. *CoRR* abs/2002.10670 (2020).

[21] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009*. 111–120.

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

[23] Tian Jiang, Lin Tan, and Sunghun Kim. 2013. Personalized defect prediction. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*. 279–289.

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

[25] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 1746–1751.

[26] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent Convolutional Neural Networks for Text Classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. 2267–2273.

[27] Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. 926–931.

[28] Stanley Lemeshow, David W Hosmer, and Wiley. 2000. *Applied logistic regression*. J. Wiley.

[29] Zhiqiang Li, Xiao-Yuan Jing, Xiaoke Zhu, Hongyu Zhang, Baowen Xu, and Shi Ying. 2019. Heterogeneous defect prediction with two-stage ensemble learning. *Autom. Softw. Eng.* 26, 3 (2019), 599–651.

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

[31] 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.

[32] Harish Tayyar Madabushi, Elena Kochkina, and Michael Castelle. 2020. Cost-Sensitive BERT for Generalisable Sentence Classification with Imbalanced Data. *CoRR* abs/2003.11563 (2020).

[33] 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.

[34] 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.

[35] 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.

[36] Thorsten Merten, Bastian Mager, Simone Bürsner, and Barbara Paech. 2014. Classifying unstructured data into natural language text and technical information. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, Premkumar T. Devanbu, Sung Kim, and Martin Pinzger (Eds.). ACM, 300–303.

[37] 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.

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

[39] Li Mu. 2014. Efficient Mini-batch Training for Stochastic Optimization. In *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining*.

[40] 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.

[41] Paras. 2014. Stochastic Gradient Descent. *Optimization* (2014).

[42] Jinfeng Rao, Ferhan Türe, and Jimmy Lin. 2018. Multi-Task Learning with Neural Networks for Voice Query Understanding on an Entertainment Platform. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. 636–645.

[43] Hang Ruan, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2019. DeepLink: Recovering issue-commit links based on deep learning. *J. Syst. Softw.* 158 (2019).

[44] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[45] Antonino Sabetta and Michele Bezzi. 2018. A Practical Approach to the Automatic Classification of Security-Relevant Commits. In *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. 579–582.

[46] Ripon K. Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne E. Perry. 2013. Improving bug localization using structured information retrieval. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*. 345–355.

[47] G. Salton, A. Wong, and C. S. Yang. 1975. A vector-space model for information retrieval. *Communications of the Acm* 18, 11 (1975), 13–620.

[48] Walt Scacchi. 2002. Understanding the requirements for developing open source software systems. *IEE Proceedings - Software* 149, 1 (2002), 24–39.

[49] Abhinav Sethy and Bhuvana Ramabhadran. 2008. Bag-of-word normalized n-gram models. In *Ninth Annual Conference of the International Speech Communication Association*.

[50] Lin Shi, Celia Chen, Qing Wang, Shoubin Li, and Barry W. Boehm. 2017. Understanding feature requests by leveraging fuzzy method and linguistic analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*. 440–450. https://doi.org/10.1109/ASE.2017.8115656

[51] 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.

[52] 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.

[53] Cong Sun and Zhihao Yang. 2019. Transfer Learning in Biomedical Named Entity Recognition: An Evaluation of BERT in the PharmaCoNER task. In *Proceedings of The 5th Workshop on BioNLP Open Shared Tasks, BioNLP-OST@EMNLP-IJNCLP 2019, Hong Kong, China, November 4, 2019*. 100–104.

[54] Yan Sun, Qing Wang, and Ye Yang. 2017. FRLink: Improving the recovery of missing issue-commit links by revisiting file relevance. *Inf. Softw. Technol.* 84 (2017), 33–47.

[55] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 3104–3112.

[56] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar M. Al-Kofahi, and Tien N. Nguyen. 2011. Fuzzy set-based automatic bug triaging. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*. 884–887.

[57] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*. 321–332.

[58] Sebastian Thrun. 1996. Is learning the n-th thing any easier than learning the first?. In *Advances in neural information processing systems*. 640–646.

[59] tkdsheep. 2020. https://github.com/tkdsheep/Intention-Mining-TSE.

[60] Vladimir Naumovich Vapnik. 2000. *The Nature of Statistical Learning Theory, Second Edition*. Springer.

[61] 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.

[62] Radu Vlas and William N. Robinson. 2011. A Rule-Based Natural Language Technique for Requirements Discovery and Classification. In *in Open-Source Software Development Projects. Proceedings of the 44th Hawaii International Conference on Systems Science*.

[63] Radu E. Vlas and William N. Robinson. 2011. A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. In *44th Hawaii International International Conference on Systems Science (HICSS-44 2011), Proceedings, 4-7 January 2011, Koloa, Kauai, HI, USA*. 1–10.

[64] 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.

[65] Min Wang, Zeqi Lin, Yanzhen Zou, and Bing Xie. 2019. CoRA: Decomposing and Describing Tangled Code Changes for Reviewer. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. 1050–1061.

[66] Werner. [n.d.]. How do I Write a Good Feature Request. https://meta.stackexchange.com/questions/7656/how-do-i-write-a-good-answer-to-a-question.

[67] Tingting Yu, Wei Wen, Xue Han, and Jane Huffman Hayes. 2019. ConPredictor: Concurrency Defect Prediction in Real-World Applications. *IEEE Trans. Software Eng.* 45, 6 (2019), 558–575.

[68] Sarim Zafar, Muhammad Zubair Malik, and Gursimran Singh Walia. 2019. Towards Standardizing and Improving Classification of Bug-Fix Commits. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2019, Porto de Galinhas, Recife, Brazil, September 19-20, 2019*. 1–6.

[69] Hongyu Zhang, Liang Gong, and Steven Versteeg. 2013. Predicting bug-fixing time: an empirical study of commercial software projects. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. 1042–1051.

[70] Tianzhu Zhang, Bernard Ghanem, Si Liu, and Narendra Ahuja. 2012. Robust visual tracking via multi-task sparse learning. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. 2042–2049.

[71] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. 649–657.

[72] Yu Zhang and Qiang Yang. 2018. An overview of multi-task learning. *National Science Review* v.5, 1 (2018), 34–47.

[73] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. *CoRR* abs/1506.06724 (2015).

[74] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schröter, and Cathrin Weiss. 2010. What Makes a Good Bug Report? *IEEE Trans. Software Eng.* 36, 5 (2010), 618–643.

[75] Rodolfo Zunino and Paolo Gastaldo. 2002. Analog implementation of the SoftMax function. In *IEEE International Symposium on Circuits & Systems*.