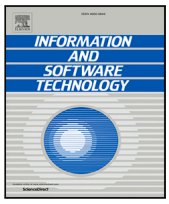
Information and Software Technology 177 (2025) 107597

Contents lists available at ScienceDirect

Information and Software Technology

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



Better together: Automated app review analysis with deep multi-task learning

Yawen Wang [a,b,c], Junjie Wang [a,b,c,*], Hongyu Zhang [d], Xuran Ming [a,b,c], Qing Wang [a,b,c,*]

[a] State Key Laboratory of Intelligent Game, Beijing, China
[b] Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, Beijing, China
[c] University of Chinese Academy of Sciences, Beijing, China
[d] Chongqing University, Chongqing, China

ARTICLE INFO

Keywords:
App review
Bug classification
Feature extraction
Deep multi-task learning

ABSTRACT

Context: User reviews of mobile apps provide an important communication channel between developers and users. Existing approaches to automated app review analysis mainly focus on one task (e.g., bug classification task, information extraction task, etc.) at a time, and are often constrained by the manually defined patterns and the ignorance of the correlations among the tasks. Recently, multi-task learning (MTL) has been successfully applied in many scenarios, with the potential to address the limitations associated with app review mining tasks.

Objective: In this paper, we propose MABLE, a deep MTL-based and semantic-aware approach, to improve app review analysis by exploiting task correlations.

Methods: MABLE jointly identifies the types of involved bugs reported in the review and extracts the fine-grained features where bugs might occur. It consists of three main phases: (1) data preparation phase, which prepares data to allow data sharing beyond single task learning; (2) model construction phase, which employs a BERT model as the shared representation layer to capture the semantic meanings of reviews, and task-specific layers to model two tasks in parallel; (3) model training phase, which enables eavesdropping by shared loss function between the two related tasks.

Results: Evaluation results on six apps show that MABLE outperforms ten commonly-used and state-of-the-art baselines, with the precision of 79.76% and the recall of 79.24% for classifying bugs, and the precision of 79.83% and the recall of 80.33% for extracting problematic app features. The MTL mechanism improves the F-measure of two tasks by 3.80% and 4.63%, respectively.

Conclusion: The proposed approach provides a novel and effective way to jointly learn two related review analysis tasks, and sheds light on exploring other review mining tasks.

1. Introduction

The mobile app development has been active for over a decade now, generating millions available apps for a wide variety of categories such as shopping, banking, and social interactions. The indispensability of mobile apps urges the development team to make every endeavor to carry out the quality assurance activities [1,2]. Users often write reviews of the mobile apps they are using on distribution platforms such as Apple Store and Google Play Store. These reviews are short texts that can provide valuable information to app developers, such as user experience, bug reports, and requests for new enhancement [3–6]. A good understanding of these reviews can help developers improve app quality and user satisfaction [7–9]. However, manually reading and analyzing each app review to gather useful feedback is very time-consuming, especially for popular apps that may receive hundreds of reviews every day.

In recent years, automated techniques for mining app reviews have attracted many research interests [1,3,10–13]. These techniques can help reduce the effort required to understand and analyze app reviews in many ways, such as classifying reviews according to specific topics [8,12,14], extracting insights from reviews [3,5,9,15],

* Correspondence to: No.4 South 4th Street, Zhongguancun, Haidian District, Beijing, China.
E-mail addresses: yawen2018@iscas.ac.cn (Y. Wang), junjie@iscas.ac.cn (J. Wang), hyzhang@cqu.edu.cn (H. Zhang), xuran2020@iscas.ac.cn (X. Ming), wq@iscas.ac.cn (Q. Wang).

https://doi.org/10.1016/j.infsof.2024.107597
Received 10 August 2023; Received in revised form 13 September 2024; Accepted 6 October 2024
Available online 10 October 2024

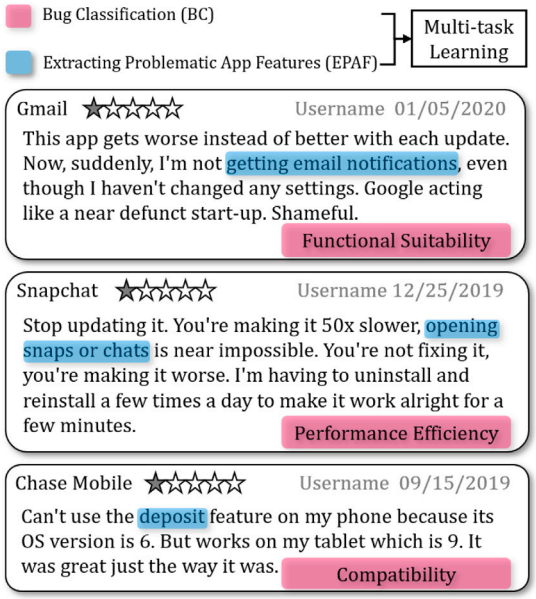


Fig. 1. Examples of BC task and EPAF task.

and summarizing reviews [4,7,13,16]. In our previous work, we proposed SIRA [17], a ***S***emantic-aware, f***I***ne-grained app ***R***eview ***A***nalysis approach, which can extract, cluster, and visualize the problematic features[1] of apps. However, SIRA has two major limitations. First, SIRA can only extract the features which users consider might have bugs from app reviews, but it cannot reveal the specific bug types involved in the problematic features. The knowledge about the types of bugs associated with problematic features could further facilitate the developers in obtaining a summarized view of issues and improving their apps. Second, although the performance of SIRA is acceptable benefiting from the model design to better understand semantics, it ignores the correlation between related tasks, e.g., certain features are prone to specific type of bugs. Such information can further improve the performance of related tasks, respectively.

In this study, we extend our previous work SIRA [17] by introducing multi-task learning (MTL) mechanism, which has led to success in many applications, from natural language processing, speech recognition to computer vision [20–22]. 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. This can result in improved efficiency and prediction accuracy, when compared to training the models separately. We mainly focus on two app review analysis tasks to facilitate the understanding of user feedback. One is Bug Classification (BC) task, which is a multi-class classification task for identifying the types of the involved bugs in the review. The other task is Extracting Problematic App Features (EPAF), which is a named entity recognition task for extracting the fine-grained problematic features (i.e., the phrases within the app review depicting the app features which might be bug-prone), as shown in Fig. 1. The fine-grained knowledge about problematic features and the involved bug types could facilitate developers in understanding the user concerns, identifying and localizing problematic modules, obtaining a summarized view of issues, thus conducting follow-up bug-fixing activities and improving their apps.

Most existing review mining approaches [3,5,7,12,13,23–27] employ pattern-based or learning-based techniques to explore valuable user opinions. Pattern-based approaches heavily rely on a set of heuristic rules pre-defined by human experts, and learning-based approaches employ the machine learning algorithms to train model for future predictions. There are four major limitations in existing approaches. **First**, for BC task, existing studies [12,24–27] typically model the reviews using bag-of-words which vectorizes the texts under the assumption that the words are independent. Although some transformer-based models [28] can be used to fully exploit the abundant semantics contained in review descriptions, they omit the information outside the text, e.g., app category and review description sentiment. The combination of review descriptions and review attributes can better model the semantics of reviews and further boost the performance of transformer-based models. **Second**, for EPAF task, manually-built patterns [3,5,7,13,23] are labor-intensive and difficult to rebuild across apps or categories in practices. **Third**, the labeled data for these two tasks is often limited. The app reviews on the distribution platforms are typically large in size, yet preparing the labeled training data is time- and effort-consuming. How to make the maximum use of the few labeled data to accurately classify and extract the knowledge in reviews becomes important. **Fourth**, previous approaches aim at using supervised training instances on a single task, but unfortunately, they ignore the rich correlation information among tasks. These correlations could also help take better advantage of the labeled data and boost the performance.

To address these challenges, we propose a deep ***M***ulti-t***A***sk learning-based approach (named MABLE) to automate the ***B***ug c***L***assification and ***E***xtracting problematic app features, which facilitate the understanding and fine-grained exploration of app reviews. MABLE consists of three main phases: (1) data preparation phase, for data sharing beyond single task learning; (2) model construction phase, which employs a BERT model as shared representation layer to capture the semantic meanings of the review descriptions, and task-specific layers to model the BC and EPAF tasks in parallel; and (3) model training phase, enabling eavesdropping by shared loss function between the two related tasks. Furthermore, during model construction phase, MABLE encodes the review attributes (i.e., app category and review description sentiment) to better model the semantics of reviews and boost the performance of the traditional model.

Evaluation results on six apps show that MABLE outperforms ten commonly-used and state-of-the-art baselines, with the precision of 79.76%, the recall of 79.24% on BC task, and the precision of 79.83% and the recall of 80.33% on EPAF task. Specifically, by jointly learning the two tasks, the performance of BC task increases F1 of 3.80%, and the EPAF task increases F1 of 4.63%. The inclusion of review attributes in the model can boost the F1 of 5.06% on BC task and 1.51% on EPAF task respectively. The proposed approach provides an effective way to jointly learn the above two related review analysis tasks, it can also be adapted/tailored to other review mining tasks to better support app review understanding and exploration.

The main contributions of this paper are as follows:

- The investigation of the correlation between two review analysis tasks, and the application of the deep MTL to automating and improving two tasks in parallel.
- MABLE, a deep MTL-based model with shared layers for sharing information and task-specific layers for jointly learning BC and EPAF tasks. MABLE also incorporates categorical attributes together with textual descriptions in the MTL-based model.
- The evaluation of MABLE on 3426 reviews from six apps spanning three categories, with affirmative results.
- Publicly accessible labeled dataset and source code to facilitate replication, and applications in other scenarios.[2]

[1] We refer to a feature as a distinctive, user visible characteristic of a mobile app [18,19], e.g., sending videos, viewing messages, etc.

[2] https://github.com/MeloFancy/MABLE.

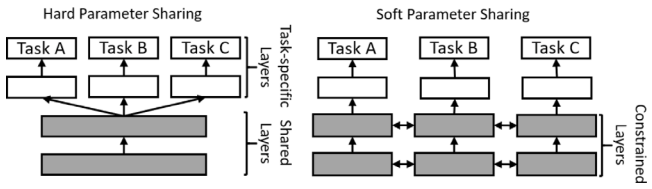


Fig. 2. Two typical approaches of MTL.

2. Background

2.1. Multi-task learning

Multi-task Learning (MTL) is an approach which can improve generalization on main task by sharing the domain-specific information contained in the training signals of related tasks [29]. Many recent deep learning approaches have used MTL either explicitly or implicitly as part of their model. For example, MTL has been applied successfully from natural language processing (NLP), speech recognition to computer vision [20–22].

So far, there are two most typical ways to perform MTL in the context of deep learning: hard or soft parameter sharing of hidden layers [30]. We demonstrate the structure of two paradigms in Fig. 2. Hard parameter sharing [31] is the most commonly-used MTL paradigm in deep neural networks. This paradigm consists of two parts in constructing the network, i.e., shared layers and task-specific layers. Through the shared layers, the general hidden vectors for input could be obtained. The hidden vectors are used as the input of the task-specific layers for many related tasks. After that, the shared information among tasks is learned by optimizing model parameters using the joint loss function. This structure can greatly reduce the risk of over-fitting. In soft parameter sharing paradigm, each task has its own model with its own parameters, where the distance between the parameters of the model is regularized by regularization techniques [32,33] to make the parameters similar. In our context, since the BC and EPAF tasks share the same inputs, i.e., the review descriptions, they are naturally suitable to the hard parameter sharing, i.e., sharing the same representation layer and processing multiple tasks in the task-specific layers. Therefore, we choose the hard parameter sharing to jointly learn the BC and EPAF tasks.

2.2. Named entity recognition

In this study, we model the EPAF task as a Named Entity Recognition (NER) task, which is a classic NLP task of sequence tagging [34,35]. Given a sequence of words, NER aims to predict whether a word belongs to named entities, e.g., names of people, organizations, locations, etc. NER task can be solved by linear statistical models, e.g., Maximum Entropy Markov models [36,37], Hidden Markov Models [38] and Conditional Random Fields (CRF) [39]. Deep learning-based techniques would use a deep neural network to capture textual semantics and a CRF layer to learn sequence-level tag rules. Typical network structures include convolutional neural network with CRF (Conv-CRF) [40], Long Short-Term Memory network with CRF (LSTM-CRF), and bidirectional LSTM network with CRF (BiLSTM-CRF) [35]. By taking advantage of the bidirectional structure, BiLSTM-CRF model can use the past and future input information and can usually obtain better performance than Conv-CRF and LSTM-CRF.

Language model pre-training techniques have been shown to be effective for improving many NLP tasks [41,42]. BERT (Bidirectional Encoder Representations from Transformers) [42] is a Transformer-based [43] representation model that uses pre-training to learn from the raw corpus, and fine-tuning on downstream tasks such as NER task. Employing BERT to replace BiLSTM (short for BERT-CRF) could lead to further performance boosts [44]. We integrate the structure of BERT-CRF in our approach to tackle the EPAF task, because it can benefit from the pre-trained representations on large general corpora after fine-tuning.

3. Approach

This paper proposes an approach for jointly conducting the BC task and EPAF task on app reviews based on deep MTL. The BC task is to identify the types of the involved bugs in the review, e.g., *Compatibility*, *Functional Suitability*, etc. The EPAF task is to extract the fine-grained problematic features from app reviews (i.e., the phrases in app reviews depicting the app features which tend to have specific type of bugs.) See the examples in Fig. 1 for details. The BC task is treated as a multi-class classification task, and the EPAF task is modeled as a NER task.

Fig. 3 presents the overview of MABLE, consisting of three main phases. ***(1) Data preparation phase***. It preprocesses the app reviews crawled from online app marketplace, in order to obtain the cleaned review descriptions and the review attributes (i.e., the category of the belonged app c and the review description sentiment s); it also prepares data to share labels across two underlying tasks beyond single task learning. ***(2) Model construction phase***. It constructs a MTL-based model for BC task and EPAF task, where a shared BERT layer captures the semantic meanings of the review descriptions, and two task-specific layers model the BC and EPAF tasks, respectively; it further combines the two review attributes to better model the semantics of reviews and boost the performance of two tasks. ***(3) Model training phase***. It enables eavesdropping by a shared loss function between the two related tasks. We will present details on each phase next.

3.1. Data preparation phase

Data preparation mainly includes three steps: data cleaning, review attribute preparation and data label sharing.

3.1.1. Data cleaning

The raw app reviews are often submitted via mobile devices and typed using limited keyboards. This situation leads to the frequent occurrences of massive noisy words, such as repetitive words, misspelled words, acronyms and abbreviations [7,13,23,45]. We treat each app review as the input unit rather than a bag of sentences, so we do not split sentences as our previous work SIRA [17], which is a big difference. This is because the BC task is inferring the bug type of the whole app reviews instead of single review sentence, and we expect to solve BC and EPAF tasks simultaneously with MTL. We first filter all non-English reviews with Langid.[3]We then tackle the noisy words problem with the following steps:

- **Lowercase:** We convert all the words in the review descriptions into lowercase.
- **Lemmatization:** We perform lemmatization with Spacy[4] to alleviate the influence of word morphology.
- **Formatting:** We replace all numbers with a special symbol <*number*> to help the model unify its understanding. Besides, we build a list containing all the app names crawled from Google Play Store, and replace them with a uniform special symbol <*appname*>.

[3] https://github.com/saffsd/langid.py.
[4] https://spacy.io.

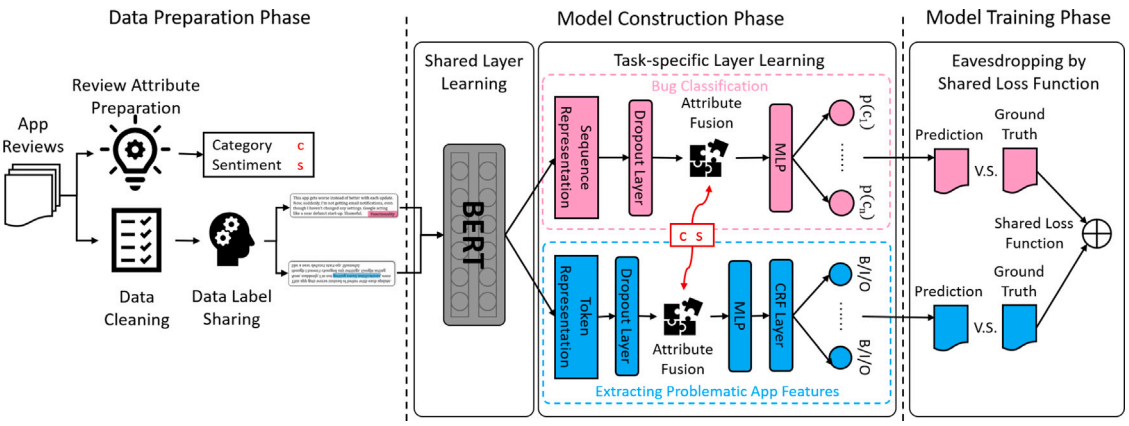


Fig. 3. The overview of MABLE.

3.1.2. Review attributes preparation

Some attributes related with the review or the belonged app can facilitate the performance of two tasks. This subsection prepares these attributes, i.e., the category of the belonged app c and the review description sentiment s, as shown in Fig. 3. The reason why we include the app category is considering the fact that apps from different categories would exert unique nature in terms of functionalities and topics [46]. For example, features that are prone to certain types of bugs in social apps might tend to be bug-free or other types of bugs for financial apps. Furthermore, review description with negative sentiment would be more likely to contain certain types of problematic features, compared with the description with positive sentiment. Hence, we include the second attribute, i.e., review description sentiment, in our model.

For app categories, we can collect them directly when crawling data from Google Play Store. For review description sentiment, many sentiment analysis tools currently available (e.g., SentiStrength [47], Stanford NLP [48]) are created and trained using data from non-technical social networking platforms (e.g., Twitter posts, forum discussions, movie reviews). However, when applied in a technical field like software engineering, their accuracy significantly decreases primarily because of domain-specific variations in the interpretations of commonly-used technical terms. Therefore, we employ SentiStrength-SE [49], a domain-specific sentiment analysis tool especially designed for software engineering text, to get the review sentiment. SentiStrength-SE would assign a positive integer score in the range 1 (not positive) to 5 (extremely positive) and a negative integer score in the range −1 (not negative) to −5 (extremely negative) to each review. Employing two scores is because previous research from psychology [50] has revealed that the human beings process the positive and negative sentiment in parallel. Following previous work [46,51], if the absolute value of the negative score multiplied by 1.5 is larger than the positive score, we assign the review the negative sentiment score; otherwise, the review is assigned with the positive sentiment score. To evaluate the accuracy of SentiStrength-SE on app reviews, we randomly sample 100 reviews to apply the tool, and manually review the positive/negative results. SentiStrength-SE achieves an accuracy of about 75%, which is comparable to the results reported in the original paper.

3.1.3. Data label sharing

Designed based on supervised learning, each app review comes with the ground-truth labels for BC task and EPAF task, respectively.

While BC task is a multi-class classification task, we identify eight types of bugs for each review description, based on the product quality model of ISO/IEC 25 010 standard.[5] We exclude *Maintainability*, which can be hardly perceived by app users, and apply other seven of them.

Table 1
Bug types and definitions.

Bug type	Definition
Compatibility (Com)	Degree to which a product, system or component can work on different platforms including the hardware and firmware, devices, operating system, and any other services available to applications. e.g., app has problems on a specific device or OS version.
Experience (Exp)	Degree to which the functions cover or exceed the user objectives. e.g., Users want to remove or expect to add certain features.
Functional Suitability (Fun)	Degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. e.g., Unexpected behavior or failure on certain features.
Performance Efficiency (Per)	Performance relative to the amount of resources used under stated conditions. e.g., The app drains battery or memory, or is slow to respond.
Portability (Por)	Degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another. e.g., Issues on app install, uninstall and update.
Reliability (Rel)	Degree to which a system, product or component performs specified functions under specified conditions for a specified period of time. e.g., Issues about app crashing/freezing.
Security (Sec)	Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization. e.g., Issues about privacy, permission, data leakage, etc.
Usability (Usa)	Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. e.g., Issues about the layout, visuals, or controls.
None	Reviews not related to bugs.

We additionally include the new type (i.e., *Experience*), because app users usually express their opinions about adding or removing of certain features. We use *None* to denote reviews which do not involve bugs. We summarize these eight bug types (sorted alphabetically) plus *None* in Table 1.

Since EPAF task is a NER task, we use the BIO tag format [52,53] to tag each review description, where

- **B-label (Beginning):** The word is the beginning of the target phrase.

[5] https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en.

- **I-label (Inside):** The word is inside the target phrase but not the beginning of it.
- **O-label (Outside):** The word is outside the target phrase.

For example, the tag sequence of "whenever I go to send a video it freezes up" is "$< O >< O >< O >< O >< B >< I >< I >< O >< O >< O >$", which indicates the tagged problematic feature is "send a video".

At the outset, MABLE allows to share labels across related tasks. Thus, the review data is explicitly organized in the following two aspects:

- **BC data preparation:** By sharing labels, the data used to train the BC task layers is automatically prepared by including additional information on problematic features. The original BC data is the plain reviews and class label, and the prepared BC data also includes its problematic features being tagged by BIO format.
- **EPAF data preparation:** Similarly, the data used to train the EPAF task layers is prepared by adding the bug type information of reviews that problematic features belong to.

3.2. Model construction phase

We model the BC task as a multi-class classification task, where each review is classified as belonging to a specific bug type. Meanwhile, we model the EPAF task as a NER task, where we treat the problematic features depicted in the review descriptions as the named entity. The model consists of two parts: a shared layer and two task-specific layers. The shared layer is to encode the review descriptions, where we employ a BERT model to better capture the semantics of the app reviews. Then we solve two tasks with task-specific layers (i.e., the Multi-layer Perceptron layer for BC task, and the CRF layer for EPAF task), respectively. Furthermore, we fuse the review attributes in our model to further boost two tasks. Two attributes, i.e., category of the belonged app c and review description sentiment s (see Section 3.1.2), are utilized in our model.

3.2.1. Shared layer learning

To better capture the semantics of reviews, we employ BERT as the shared layer to embed reviews into semantic vectors. The BERT model has been pre-trained on a large volume of natural-language texts (BooksCorpus [54] containing 800M words and English Wikipedia containing 2500M words). Since app reviews are short texts, and the involved vocabulary is relatively small, we use the pre-trained model $BERT_{BASE}$,[6] which has 12 layers, 768 hidden dimensions and 12 attention heads, and retrain it with fine-tuning technique. At the input, each review is represented by word tokens with a special starting symbol $[CLS]$. BERT will output a sequence representation and a token representation. The former is fed into an output layer for classification tasks (i.e., BC task), and the latter is fed into an output layer for token level tasks (i.e., EPAF task) [42].

3.2.2. Task-specific layer learning

BC Task Layers. As we mentioned above, the sequence representation obtained from BERT is fed into BC task layers. Each sequence representation (denoted as v) are 768 dimensions, representing the semantic vector of the input reviews. In detail, the sequence representation is first fed into a dropout layer to avoid over-fitting. To jointly capture the underlying meaning of review description, we fuse the review attributes into the vector v, which is demonstrated in Fig. 4 in detail. The review attributes (i.e., c and s) extracted in Section 3.1.2 are discrete values. We first feed them into the embedding layer, which can convert discrete values into continuous vectors (denoted as h_c and h_s). Then we concatenate h_c, h_s and v ($h_c \bigoplus h_s \bigoplus v$) to obtain a new vector (denoted as v'). Taking attribute s as an example, it can take ten values (-5 to -1, and 1 to 5). It is first represented by a continuous vector h_s via the embedding layer. Then h_s is concatenated with the continuous vector of app category h_c and the sequence representation v to get v'. The vector v' incorporates both the review semantics and the attribute information, and can be tuned jointly with the parameters of the whole network. The concatenated vector (v') is fed into a Multi-layer Perceptron (MLP), which computes the probability vector (denoted as p) of bug types:

$$p = f(W \cdot v') \tag{1}$$

where $f(\cdot)$ is the activation function, and W is trainable parameters in MLP. Finally, we perform softmax operation on p to normalize the probabilities and determine the most likely bug type based on the probability distribution. The softmax is specified as:

$$Softmax(p_i) = \frac{e^{p_i}}{\sum_{j=1}^{n} e^{p_j}} \tag{2}$$

where $p = (p_1, p_2, \ldots, p_n)$, and n is the number of bug types.

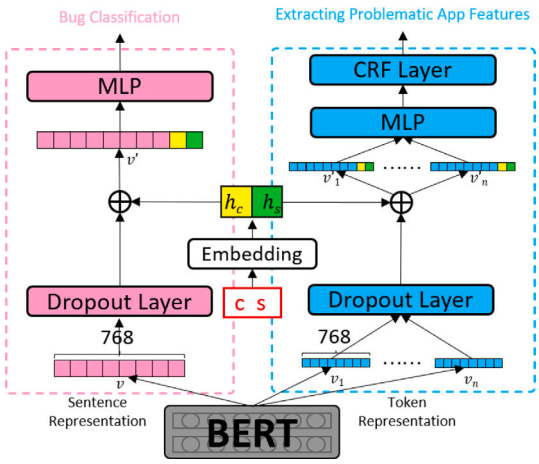


Fig. 4. Details of attribute fusion.

EPAF Task Layers. As mentioned in Section 3.2.1, the token representation obtained from BERT is fed into EPAF task layers. The dimensions of the token representation is $n \times 768$, where n is the length of the input review, and 768 is the dimension of each vector (denoted as v_i) corresponding to each input word i. In detail, the token representation is first fed into a dropout layer to avoid over-fitting. Following similar process in BC task, we fuse the review attributes into the vector v_i for better modeling the review semantics. As illustrated in Fig. 4, the review attributes (c and s) first go through the embedding layer to obtain h_c and h_s. We then concatenate h_c, h_s and v_i ($h_c \bigoplus h_s \bigoplus v_i$) to obtain a new vector (denoted as v'_i) for each input word. The concatenated vectors ($v'_1 \sim v'_n$) then go through an MLP, which computes the probability vector (denoted as p_i) of BIO tags for each word:

$$p_i = f(W \cdot v'_i) \tag{3}$$

where $f(\cdot)$ is the activation function, and W is trainable parameters in MLP. Finally, p_i of all words are input into the CRF layer to determine the most likely tag sequence based on Viterbi Algorithm [55]. With the derived tag sequence, we can obtain the phrases of problematic features based on BIO format.

3.3. Model training phase

[6] https://huggingface.co/bert-base-uncased.

3.3.1. Eavesdropping by shared loss function

The shared loss function consists of two parts, respectively for BC task and EPAF task. Since the BC task is a classification task, we use the commonly-used cross entropy as the loss function, which is specified as:

$$Loss_{BC}(\theta) = -\sum_{i} y_i \cdot log(p_i(\theta)) \tag{4}$$

where θ is the parameters from MLP, and y_i is the ground-truth bug type label. $p_i(\theta)$ is the predicted probability distribution of review i, which is the output of MLP with parameters θ.

For EPAF task, the loss function should measure the likelihood of the whole true tag sequence, instead of the likelihood of the true tag for each word in the sequence. Therefore, the commonly-used cross entropy is not suitable in this context. Following existing studies [35], the loss function consists of the emission score and the transition score as below:

$$Loss_{EPAF}([x]_1^T, [l]_1^T, \widetilde{\theta}) = \sum_{t=1}^{T}([A]_{[l]_{t-1},[l]_t} + [f_\theta]_{[l]_t,t}) \tag{5}$$

where $[x]_1^T$ is the token sequence of length T, and $[l]_1^T$ is the tag sequence of length T. $[A]_{i,j}$ is the transition score, which is obtained with the parameters from CRF layer. It models the transition from the ith state to the jth state in the CRF layer. $f_\theta([x]_1^T)$ is the emission score, which is the output of MLP with parameters θ. $\widetilde{\theta} = \theta \cup \{[A]_{i,j} \forall i, j\}$ is the new parameters for the whole network. The loss of a token sequence $[x]_1^T$ along with a tag sequence $[l]_1^T$ is derived by the sum of emission scores and transition scores.

Following previous work [56], we employ a principled, shared loss function to take the weighted sum of the two individual losses, which is specified as:

$$Loss = \lambda Loss_{BC} + (1 - \lambda) Loss_{EPAF} \tag{6}$$

where λ is harmonic factor in the range $(0, 1)$. The shared loss function can allow the model to eavesdrop information between two related tasks, and learn some features which are difficult to learn from itself but easy from the other task. The hyper-parameter λ is determined by a greedy strategy. For example, if the candidate values of λ are $\{0.1, 0.2, \ldots, 1\}$, we train MABLE under each candidate value, and evaluate it on the validation dataset. After 10 iterations, we choose the values which leads to the best performance as the tuned value of λ.

3.3.2. Training details

All the hyper-parameters in MABLE are tuned carefully with a greedy strategy to obtain best performance. Given a hyper-parameter P and its candidate values $\{v_1, v_2, \ldots, v_n\}$, we perform automated tuning for n iterations, and choose the values which lead to the best performance as the tuned value of P. After tuning, the harmonic factor λ is set as 0.6, and the learning rate is set as 10^{-4}. The optimizer is Adam algorithm [57]. We use the mini-batch technique for speeding up the training process with batch size 16. The drop rate is 0.1, which means 10% of neuron cells will be randomly masked to avoid over-fitting.

We implement MABLE based on an open-source Pytorch library, named Transformers.[7] Our implementation and experimental data are available online.[8]

4. Experimental design

4.1. Research questions

We answer the following three research questions:

- **RQ1:** What is the performance of MABLE on BC task and EPAF task respectively compared to existing approaches?
- **RQ2:** What is the benefit of multi-task learning applied in MABLE compared to single task learning?
- **RQ3:** Is each type of the review attributes employed in MABLE necessary?

4.2. Data preparation

We select six apps from three categories (two in each category) as our experimental object. All selected apps and their belonging categories are commonly-used, which ensure the richness of reviews in practice. We first crawl the app reviews from Google Play Store submitted during August 2019 to January 2020, with the tool google-play-scraper.[9] Then we retain the reviews with one-star rating, because they are more potentially to contain problematic features, and we filter the non-English ones and uninformative ones (e.g., short reviews only containing emoticons, etc.). Finally, we obtain around 550 reviews for each app and label them for further experiments.

Three authors then manually label the app reviews to serve as the ground-truth in verifying the performance of MABLE. This labeling process involves assigning labels for both BC and EPAF tasks. For BC task, annotators need to assign one of bug types in Table 1 to each review according to the semantics of reviews and ISO guidelines [58–60]. For EPAF task, annotators need to mark the begin and end position of the problematic features in each review, which can be converted to BIO format easily. Considering both the working mechanism and the usefulness of MABLE, the guidelines for feature labeling are as follows:

- Label the phrases related to specific functionality as features, while the phrases related to user experience, compatibility, or portability are not considered as features, e.g., "read the message" is a feature, whereas "app install/uninstall/update" is not considered as a feature.
- For sentences like "*I cannot/hardly/... to PhraseX when/since/... PhraseY.*", label PhraseX after "cannot" as feature instead of PhraseY after "when", e.g., for the review sentence "*I cannot send email when I click on the button.*", "send email" is regarded as a feature, not "click on the button".
- Focus on the functionality aspect, ignore the quality aspect and exclude adverbs, e.g., "upload the photo quickly" becomes "upload the photo".
- Simultaneously include the action (verb) and the function module (noun), e.g., "voice message" becomes "receive voice message".
- Consecutive chunks of words, avoid splitting and deriving, e.g., "search or like the post" rather than "search the post" and "like the post".

To guarantee the accuracy of the labeling outcomes, first, the first two authors independently label the app reviews of an app. Second, the fourth author compares the labeling results, finds the differences, and organizes a face-to-face discussion among them three to determine the final labels. The annotation of all the six apps follows the same process. For the first labeled app (*Instagram*), the Cohen's Kappa is 0.78 between the two annotators, while for the last labeled app (*Gmail*), the Cohen's Kappa is 0.86. After two rounds of labeling, common consensus is reached for each review.

Table 2 elaborates the statistics of the experimental dataset in terms of apps and bug types in detail. The bugs related to *Fun*, *Exp* and *Rel* are mostly complained about by app users, while the reviews referring to *Com*, *Sec* and *Usa* bugs are relatively rare. Our dataset contains 3426 reviews in total. Note that there are some reviews that are not categorized as *None*, but do not mention any feature. It is common when the reviews belong to *Experience*, *Compatibility* and *Portability*, as required by the first guideline for feature labeling. The proportion of such reviews in the labeled dataset is 768/3426.

[7] https://github.com/huggingface/transformers.
[8] https://github.com/MeloFancy/MABLE.
[9] https://github.com/facundoolano/google-play-scraper.

Table 2
Experimental dataset.

	Instagram	Snapchat	BPI Mobile	Chase Mobile	Yahoo Mail	Gmail	Total
Com	8	20	6	16	3	2	55
Exp	191	107	69	68	103	165	703
Fun	189	186	265	131	179	234	1184
Per	9	23	13	17	97	20	179
Por	14	41	51	84	11	14	215
Rel	87	73	140	166	90	65	621
Sec	14	7	0	20	9	22	72
Usa	15	31	3	6	17	21	93
None	55	97	41	35	33	43	304
Total	582	580	584	544	540	586	3426

4.3. Experimental setup

To answer RQ1, we conduct nested cross-validation [61] on the experimental dataset. The inner loop is for selecting optimal hyper-parameters, which are used for evaluating performance in the outer loop. In the outer loop, we randomly divide the dataset into ten folds, use nine of them for training, and utilize the remaining one fold for testing the performance. The process is repeated for ten times, and the average performance is treated as the final performance. In the inner loop, we use eight folds for training and one fold for validation. We run each baseline (see Section 4.4) to obtain its performance following the same experimental setups. To verify the generalization capabilities of MABLE, we additionally conduct cross-app validation. Specifically, one app's reviews would be held out for testing, while the model is trained on reviews from other apps.

For RQ2, we first configure MABLE into three working modes. One is MTL mode (i.e., MABLE). The other two are its single task learning modes: single BC task (MABLE_{BC}) and single EPAF task (MABLE_{EPAF}), where the former is a transformer-based classification network and the latter is namely SIRA, respectively. 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., BC view with only bug type labels and EPAF view with only problematic features, to train the two corresponding models, respectively. We use the same nested cross-validation setups as RQ1.

For RQ3, we design three variants of MABLE to demonstrate the necessity of employed review attributes in our context. In detail, MABLE_{base}, MABLE_{senti} and MABLE_{cate} respectively represent the model without review attributes (i.e., only with text), the model without app category (i.e., with text and review description sentiment), and the model without review description sentiment (i.e., with text and app category). We reuse the same nested cross-validation setups as RQ1.

The experimental environment is a desktop computer equipped with a NVIDIA GeForce RTX 2060 GPU, intel core i7 CPU, 16 GB RAM, running on Windows 10, and training the MABLE takes about 4 h for each fold of nested cross-validation.

4.4. Baselines

4.4.1. Baselines for bug classification

We do not select transformer-based baselines, because its network architecture is the same as MABLE_{BC} in RQ2, and the analysis would be presented in Section 5.2. The baselines for BC task are as below:

Machine Learning-Based Classifiers: Some existing studies [24,27] employed machine learning-based classifiers to solve the bug classification problem. Following their practice, we first adopt the Term Frequency–Inverse Document Frequency (TF–IDF) model [62] to convert the review descriptions into vectors, then classify the TF–IDF vectors into bug types. Four typical classifiers are utilized to comprehensively examine the classification performance, i.e., ***Naive Bayesian (NB)*** [63], ***Support Vector Machine (SVM)*** [64], ***Logistic Regression (LR)*** [65] and ***Random Forest (RF)*** [66].

TextCNN [67]: The convolutional neural network for text classification which has been proven to be useful in many NLP tasks such as sentiment analysis and question classification.

TextRNN [68]: The recurrent neural network for text classification, which uses BiLSTM and attention mechanism to capture bidirectional semantics of a text sequence.

4.4.2. Baselines for extracting problematic app features

We select methods that can extract target phrases from app reviews as baselines for extracting problematic app features. To the best of our knowledge, existing methods are mainly pattern-based, which can be classified into three types based on the techniques: Part-of-Speech (PoS) pattern (e.g., SAFE [3] and PUMA [23]); dependency parsing plus PoS pattern (e.g., Caspar [5] and SUR-Miner [7]); pattern-based filter plus text classification (e.g., KEFE [69]). We select the representative method from each type as baselines, i.e., KEFE, Caspar and SAFE. In addition, since we model the feature extraction task as an NER task, we also include BiLSTM-CRF [35], a commonly-used technique in NER tasks, as a baseline. We do not include SIRA as a baseline, because it is exactly the same as MABLE_{EPAF} in RQ2, and we would present the analysis in Section 5.2. We introduce the four baselines in detail below:

BiLSTM-CRF [35]: A commonly-used algorithm in sequence tagging tasks such as NER. Being a deep learning-based technique, it utilizes a BiLSTM to capture textual semantics and a CRF layer to learn sequence-level tags.

KEFE [69]: An approach for identifying key features from app reviews. A key feature is referred as the features that are highly correlated to app ratings. It employs a pattern-based filter to obtain candidates, and a BERT-based classifier to identify the features. Since its patterns are designed for Chinese language, we replace them with the patterns in SAFE [3] to handle English reviews.

Caspar [5]: An approach for extracting and synthesizing user-reported mini stories regarding app problems from reviews. We treat its first step, i.e., events extraction, as a baseline. An event is referred as a phrase that is rooted in a verb and includes other attributes related to the verb. It employed Part-of-Speech (PoS) tagging and dependency parsing on review sentences to address this task. We use the implementation provided by the original paper.[10]

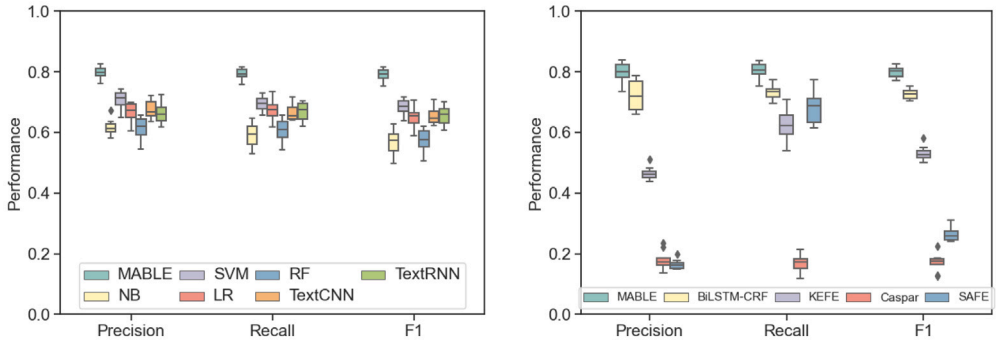
SAFE [3]: A pattern-based approach to extract feature-related phrases from reviews with 18 PoS patterns. For example, the pattern *Verb-Adjective-Noun* can extract features like "delete old emails". We implement all 18 patterns to extract the phrases based on the NLP toolkit NLTK.[11]

4.5. Evaluation metrics

We use commonly-used metrics, i.e., precision, recall, and F1-Score, to evaluate the performance of MABLE for both BC task and EPAF task. Since BC is a multi-class classification task, when calculating metrics on multiple labels, we first calculate metrics for each label, and then obtain their weighted average by considering the number of ground-truth samples for each bug type following existing practices [70,71]. In EPAF task, for a review description of an app, we consider a problematic feature is correctly predicted if the predicted phrase from MABLE is the same as the ground-truth one. Three metrics are computed as:

- **Precision** is the ratio of the number of correct predictions to the total number of predictions.
- **Recall** is the ratio of the number of correctly predictions to the total number of ground-truth samples.

[10] https://hguo5.github.io/Caspar.
[11] https://github.com/nltk/nltk.



(a) Performance comparison on BC task.

(b) Performance comparison on EPAF task.

Fig. 5. Performance comparison of MABLE and baselines (RQ1).

Table 3
Performance of MABLE on BC task across types (RQ1).

Bug type	Metric		
	Precision	Recall	F1
Compatibility	67.20%	52.90%	54.95%
Experience	72.17%	71.86%	71.70%
Functional Suitability	82.31%	85.99%	84.03%
Performance Efficiency	82.62%	80.72%	81.05%
Portability	72.43%	78.64%	74.84%
Reliability	85.72%	87.88%	86.72%
Security	63.48%	67.15%	64.01%
Usability	62.86%	51.46%	56.03%
None	84.08%	68.07%	74.77%
Overall	79.76%	79.24%	79.04%

Table 4
Performance of MABLE on BC and EPAF tasks across apps (RQ1).

App	Metric					
	BC task			EPAF task		
	Precision	Recall	F1	Precision	Recall	F1
Instagram	83.66%	83.30%	83.46%	80.09%	79.89%	79.89%
Snapchat	80.09%	77.62%	78.83%	76.01%	81.72%	78.58%
BPI Mobile	82.21%	81.05%	81.61%	83.14%	78.85%	80.80%
Chase Mobile	79.62%	76.75%	78.15%	78.79%	78.44%	78.42%
Yahoo Mail	82.35%	79.47%	80.87%	73.71%	84.22%	78.49%
Gmail	79.72%	77.55%	78.59%	85.26%	79.91%	82.43%
Overall	79.76%	79.24%	79.04%	79.83%	80.33%	80.02%

- **F1-Score** is the harmonic mean of precision and recall.

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{7}$$

5. Results and analysis

5.1. Answering RQ1

Performance of BC task. Table 3 presents the overall performance of MABLE on BC task, as well as the performance in terms of each bug type. For BC task, the overall precision, recall and F1 reach 79.76%, 79.24% and 79.04% respectively, which indicates that our proposed approach can accurately classify the bug types of app reviews. Furthermore, MABLE can reach relatively promising performance on *Reliability* (86.72%) and *Functional Suitability* (84.03%) in F1, while the performances of *Compatibility* (54.95%) and *Usability* (56.03%) are relatively low. *Reliability* achieves the best performance because the related reviews usually share similar expressions (e.g., "crash"/"be frozen"). Yet for low-performance types like *Compatibility*, the related reviews are more diversified, and difficult to be identified in nature. We also present the performance of MABLE on BC task across apps in Table 4. The performance difference among six apps is small. MABLE obtains the best F1 on *Instagram* (83.46%), and the worst F1 on *Chase Mobile* (78.15%).

Performance of EPAF task. We then turn to the EPAF task in Table 4. The overall precision, recall, and F1 on EPAF task reach 79.83%, 80.33%, and 80.02% respectively, which is also promising. Note that the performance of MABLE on EPAF task does not outperform that of our previous work SIRA. It is because the input of MABLE is app reviews rather than review sentences as SIRA. This difference means that the input texts are getting longer, and each of them is more likely to contain multiple problematic feature phrases, which increases the difficulty of training the model. And the performance of SIRA on review-level in this context is presented by MABLE_{EPAF} in RQ2. When it comes to the performance in terms of each app, MABLE reaches the highest precision 85.26% and the highest F1 82.43% on *Gmail*, and the highest recall is 84.22% on *Yahoo Mail*. Its lowest precision is 73.71% on *Yahoo Mail*, the lowest recall 78.44%, and the lowest F1 78.42% on *Chase Mobile*. We can see that even with its worst performance, an acceptable precision and recall can be achieved.

We then examine the extracted problematic features in detail, and find that there are indeed some observable patterns associated with the problematic features. For example, users would use some negative words (e.g., "cannot", "hardly") or temporal conjunctions (e.g., "as soon as", "when") before mentioning the problematic features. This could probably explain why the pattern-based techniques [3,5,16] could work sometimes. However, these techniques highly rely on the manually defined patterns and have poor scalability in a different dataset. Furthermore, there are many circumstances when the pattern-based approach can hardly work. For example, it is quite demanding to design patterns for the following review text: "this update takes away my ability to view transactions", where the problematic feature is "view transactions". These circumstances further prove the advantages and flexibility of our approach.

Generalization capability of MABLE. Table 5 presents the performance of MABLE on the cross-app validation. For BC task, the overall precision, recall and F1 reach 74.77%, 73.75% and 74.25% respectively, 78.61%, 74.90%, and 76.55% for EPAF task respectively. Compared to the performance of nested cross-validation as shown in Table 4, the performance decreases within 5%. The experiments under two setups share some similar results, e.g., *Chase Mobile* obtains the worst performance (F1) on both tasks. The changes are where the best performance (F1) is obtained by *BPI Mobile* on both tasks on cross-app validation, rather than *Instagram* and *Gmail* on nested cross-validation. This is because the unbalanced label distribution among different apps would influence the performance when conducting the

Table 5
Performance of MABLE on cross-app validation (RQ1).

App	Metric					
	BC task			EPAF task		
	Precision	Recall	F1	Precision	Recall	F1
Instagram	74.99%	74.40%	74.69%	80.00%	72.57%	76.11%
Snapchat	74.29%	73.79%	74.04%	78.10%	76.07%	77.07%
BPI Mobile	78.17%	77.05%	77.61%	81.94%	78.99%	80.44%
Chase Mobile	68.50%	66.73%	67.60%	75.58%	69.94%	72.65%
Yahoo Mail	77.31%	76.11%	76.70%	71.49%	80.94%	75.92%
Gmail	75.36%	74.40%	74.88%	84.57%	70.87%	77.12%
Overall	74.77%	73.75%	74.25%	78.61%	74.90%	76.55%

Table 6
The benefit (F1) of MTL (RQ2).

App	Method			
	BC task		EPAF task	
	MABLE	MABLE_{BC}	MABLE	MABLE_{EPAF}
Instagram	**83.46%**	78.89%	**79.89%**	77.06%
Snapchat	**78.83%**	74.63%	**78.58%**	77.39%
BPI Mobile	**81.61%**	79.18%	**80.80%**	76.94%
Chase Mobile	**78.15%**	72.17%	**78.42%**	74.27%
Yahoo Mail	**80.87%**	79.67%	**78.49%**	74.72%
Gmail	78.59%	**79.12%**	**82.43%**	77.45%
Overall	**79.04%**	76.15%	**80.02%**	76.48%

cross-app validation. For example, the reviews of *BPI Mobile* and *Gmail* occupies 42% of the total *Functional Suitability* reviews. Overall, a performance (F1) of around 75% on cross-app validation can prove the generalization capability of MABLE to some extent.

Comparison with baselines. Fig. 5 presents the comparison of overall performance between MABLE and baselines on both tasks. We first look at BC task in Fig. 5(a), MABLE outperforms all baselines on all metrics significantly (p-value <0.05 of Mann–Whitney test). Among all baselines, SVM gets the second best performance on all metrics (precision 70.54%, recall 69.49%, and F1 68.37%), which indicates that SVM is more powerful than other machine learning-based classifiers. The performance of TextCNN and TextRNN (F1 65.49% and 65.60% respectively) is lower than SVM, because the massive number of parameters cannot be trained adequately on such small dataset when lacking pre-training on external corpus. MABLE outperforms the two deep learning-based methods because of the pre-training and fine-tuning techniques from BERT and the mechanism of MTL.

Fig. 5(b) presents the overall performance of MABLE and four baselines on EPAF task. MABLE outperforms all baselines on all metrics significantly (p-value <0.05 of Mann–Whitney test). The comparison between MABLE and BiLSTM-CRF indicates the advantage of BERT and the mechanism of MTL. The comparison between two deep learning-based methods (i.e., MABLE and BiLSTM-CRF) and other three pattern-based baselines (i.e., KEFE, Caspar and SAFE) indicates that the pattern-based baselines are far from effective in extracting problematic features. Among three pattern-based methods, SAFE achieves the highest recall but the lowest precision, i.e., 16.40% precision and 68.25% recall. This is because it defines 18 PoS patterns to extract feature-related phrases, and can retrieve large number of potential problematic features (i.e., high recall). KEFE achieves the promising performance, indicating that it can filter away many low-quality phrases with the BERT classifier. However, the classification is still conducted based on candidate features extracted by a pattern-based method, which limits its performance. By comparison, Caspar can only extract events from reviews containing temporal conjunctions and key phrases (e.g., "when", "every time", etc.) which can hardly work well in this context.

5.2. Answering RQ2

Table 6 presents the F1 of MTL mode (MABLE) and two single task learning modes (MABLE_{BC} and MABLE_{EPAF}). We can observe that MABLE outperforms both MABLE_{BC} and MABLE_{EPAF} on overall F1. The improvement on BC task and EPAF task reach 3.80% and 4.63% respectively, which indicates that the shared features learned from two related tasks indeed improve classification, and help retrieve more correct problematic features simultaneously. Note that, the model design of MABLE_{EPAF} is exactly the same as the problematic feature extraction model in our previous work SIRA, while the performance gets lower because the input turns into the whole review from single sentences. For the performance across apps on both tasks, MABLE obtains the best F1 on most apps, except for *Gmail* on BC task where the F1 of MABLE (78.59%) is a bit lower than MABLE_{BC} (79.12%).

Overall, compared with its single task learning modes, the MTL mode of MABLE can achieve higher performance. The performance improvement of MTL is attributed to the internal shared learning mechanism of MABLE (shared BERT layer and shared loss function), which can share the features between two related tasks. The improvement for EPAF task is slightly more than BC task, which indicates that BC task can share more informative features with EPAF task.

5.3. Answering RQ3

Table 7 presents the F1 of MABLE and its three variants, respectively. On BC task, the overall F1 of MABLE is higher than all the three variants. When compared with the base approach MABLE_{base}, adding the app category and the sentiment attributes (MABLE) noticeably increases the F1 by 5.06%. This indicates that the two attributes are helpful in classifying bugs. Among the two added review attributes, the sentiment attribute (MABLE_{senti}) contributes slightly more to F1 improvement (+1.99%) than the app category attribute (MABLE_{cate}, +1.16%). Then we turn to EPAF task. We can find that MABLE reaches the best overall F1. Compared with the base MABLE_{base}, adding both review attributes (MABLE) improves the F1 by 1.51%. Only adding the sentiment attribute (MABLE_{senti}) increases the F1 by 0.85%, and only adding the category attribute (MABLE_{cate}) increases the F1 by 0.66%. For the performance across apps, MABLE obtains the best F1 on most apps (4/6), while MABLE_{senti} obtains the best F1 on the other two apps.

From the results above, we have the following observations. First, the contribution of these two attributes overlaps to some extent, i.e., the increased performance by each attribute is not simply added up to the overall performance. This is reasonable considering the fact that words expressing the user sentiment in the textual descriptions could be encoded semantically and captured by the BERT model. Second, the added review attributes are more useful for BC task than EPAF task, and the sentiment attribute contributes more than the category attribute. Nevertheless, the F1 achieved by adding both of the attributes is the highest, further indicating the contribution of two review attributes.

6. Discussion

6.1. Why does MTL work

The reason why we apply MTL is that BC and EPAF tasks are correlated with each other naturally. We indeed observe this correlation when examining the results of two tasks produced by MABLE. For each app, we cluster all predicted problematic features into twenty topics using topic model LDA [72], and count the numbers of problematic features of each bug type on each topic, then normalize them into ratios as shown in Fig. 6.

We can observe the trend that certain problematic features are related to specific bug types. For example, the problematic feature which is most related to *Reliability* is "log in" in all six investigated apps (i.e., bug type 6 and topic 9). In *Yahoo Mail*, the feature "load

Table 7
The ablation experiment (F1) of MABLE (RQ3).

App	Method							
	BC task				EPAF task			
	MABLE	$MABLE_{base}$	$MABLE_{senti}$	$MABLE_{cate}$	MABLE	$MABLE_{base}$	$MABLE_{senti}$	$MABLE_{cate}$
Instagram	**83.46%**	79.74%	80.07%	78.97%	79.89%	79.63%	**81.23%**	80.49%
Snapchat	**78.83%**	75.05%	76.24%	76.14%	**78.58%**	77.11%	78.28%	78.37%
BPI Mobile	**81.61%**	79.60%	78.13%	79.04%	80.80%	81.03%	**81.57%**	81.44%
Chase Mobile	**78.15%**	69.86%	74.71%	74.00%	**78.42%**	75.72%	76.00%	76.81%
Yahoo Mail	**80.87%**	78.56%	78.61%	77.98%	**78.49%**	76.78%	76.83%	77.54%
Gmail	78.59%	76.68%	**78.82%**	77.78%	**82.43%**	80.76%	81.40%	80.26%
Overall	**79.04%**	75.23%	76.73%	76.10%	**80.02%**	78.83%	79.50%	79.35%

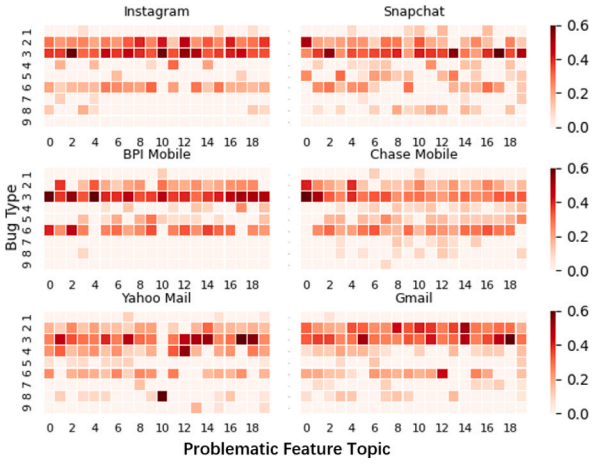


Fig. 6. The correlations between two tasks.

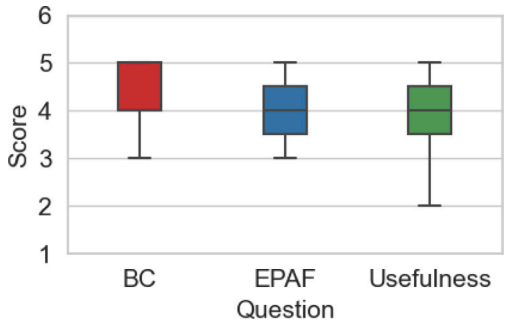


Fig. 7. Feedback of user study.

emails" is highly associated with the bug type *Performance* (i.e., bug type 4 and topic 12), because many users complain that loading emails becomes so slow after the recent update. Reviews which belong to *Compatibility* (i.e., bug type 1) and *Security* (i.e., bug type 7) have a high probability of containing no problematic features, because these types of bugs tend to relate with the whole system instead of certain features. These correlations indicate the potential of jointly learning these two related tasks via MTL, which can share information between them and boost the performance of each task.

6.2. The generality of MABLE

In order to assess the generality and usefulness of MABLE, we conduct a user survey on three popular apps: *Weibo*, *QQ* and *Alipay*. We first crawl the reviews of three apps submitted during the first week of May 2021. Note that the apps and the time period are different from the experimental dataset shown in Table 2. After filtering the low-quality reviews and preprocessing, we construct a new dataset for user study, which contains 177 reviews from *Weibo*, 149 from *QQ*, and 177 from *Alipay*. We invite 15 respondents (5 from each company) in total, including 2 product managers, 5 requirement analysts, and 8 developers, who are familiar with the app reviews of their own company. Each respondent examines the bug type and extracted features obtained by MABLE, and answer the following three questions: (1) (BC) Can MABLE classify the bug type accurately? (2) (EPAF) Can MABLE extract problematic app features accurately? (3) (Usefulness) Can MABLE help understand user requirements from app reviews? We provide five options for each question from 1 (strongly disagree) to 5 (strongly agree). The first two questions concern the performance of MABLE on BC task and EPAF task respectively, when analyzing app reviews in real-world practice. The last question concerns the usefulness of MABLE, i.e., whether MABLE can save effort for analyzing large-scale app reviews.

Fig. 7 shows the box plot statistics of respondents' feedback. There are respectively 12, 11 and 11 (out of 15) respondents give the score over 3 for Q1, Q2, and Q3. Most of them (over 73%) are satisfied (score over 3) with the usefulness of MABLE, and think MABLE can help them obtain a fine-grained understanding on bug type and problematic features. The average score of Q1, Q2, and Q3 are 4.13, 4, and 3.93 respectively. Besides, three of them heard about or tried existing review analysis tools such as INFAR [16] and SUR-Miner [7], and they admit the advantages of MABLE as its derived bug type and extracted features are more accurate and more meaningful. We also interviewed the respondents about the possible enhancement of MABLE. They said there were still some cases where MABLE does not work well, such as some extracted phrases contain two or more features. This can be solved in future work by exploring the patterns of such tangled features and deconstructing them into separate ones.

6.3. The application of MABLE

Computational Requirements. Training MABLE on a dataset of about 3000+ data takes about 4 h per fold of nested cross-validation. Considering that the performance of the server we used for training is not particularly superior (see detailed experimental environment in Section 4.3), and that the nested cross-validation setup is cumbersome, we consider such computational overhead to be acceptable. During the evaluation process, we did not perceive a significant difference in computational overhead compared to the learning-based baselines, neither in the training phase nor in the prediction phase. In addition, since our method learns two tasks simultaneously, its computational overhead is even lower than the sum overhead of learning two tasks separately. From the view of computational requirements, we believe that MABLE can be adapted to most cases.

Applicability. We provide some insights about the applicability of MABLE. First, although the evaluation of MABLE is only conducted on BC and EPAF tasks using app reviews, it is potential to be adapted and applied to handling other data sources and tasks. Our approach can be applied directly to other data sources (e.g., bug reports), when people expect to classify bug types and extract feature-related phrases simultaneously. More specifically, the architecture of MABLE can be adapted to other classification tasks (e.g., sentiment analysis) and information extraction tasks based on specific contexts, as long as people annotate their own dataset and customize their own attributes just like app category and review sentiment in our context. Actually, even MABLE_{base} without any attributes can obtain promising results in our experiment.

Second, we do not treat the BC task as a multi-label problem, mainly to simplify the problem. This is because the focus of this paper is to demonstrate the performance can be further improved by combining two single tasks with some correlations via MTL (Section 6.1 illustrates the correlation between BC and EPAF task and explains why MTL works). Therefore, we simply treat the BC task as a multi-class problem to avoid the trouble of complicated linguistic structures that require advanced NLP techniques to handle (e.g., the refinement and splitting of composite features) and the costs associated with manual annotation. For the potential multi-label problem in data annotation, we simply reach an agreement through discussion and assign the most likely labels. We leave the MABLE application on other combinations of tasks as future work, such as the combination of the multi-label BC task with other tasks.

Besides, MABLE can also be applied to handle different languages since MABLE does not make use of language-specific features to train the model. When switching to other languages, users only need to apply additional pre-processing according to the specific language, e.g., switching to the BERT pre-trained by the specific language corpus. However, MABLE is not equipped to handle mixed languages now.

Finally, the performance of MABLE on certain bug types or apps is not stable. This is because the distribution of labels is not balanced across bug types or apps. For example, the type *Functional Suitability* accounts for approximately 34.6% of the total data. This can lead to biased learning, i.e., the model tend to predict the majority of types because this gives better overall performance, ignoring the minority of types. These limitations can be addressed by designing sensible sampling techniques, or performing data augmentation to increase the number of minority types. We would explore the solutions to these issues in the future.

Practical Implications. We also provide some insights about the practical implications of MABLE. First, the results (bug type and problematic features) of MABLE can be further processed (e.g., clustering, visualizing) to provide a summarized view of issues for single app or the comparison across apps. This enables the developers to acquire where the app is prone to issues, and where other apps are also likely to have issues.

Second, developers can leverage reviews from similar apps for quality assurance activities, rather than only focus on the limited set of reviews of its own app. This is especially the case for the less popular apps which only have few reviews regarding app issues. The results provided by MABLE can assist the developers to arrange the follow-up problem solving and allocate the testing activity for subsequent versions. More than that, these information can also assist the developers in the competitive analysis of apps, e.g., acquire the weakness of their app compared with similar apps.

Finally, MABLE can facilitate app testing and refine the testing techniques. For example, it can recommend problematic features to similar apps in order to prioritize the testing effort, or recommend related descriptions (mined from app reviews) to similar apps to help bug detection. In addition, the automated graphical user interface (GUI) testing techniques can be customized and the testing contents can be prioritized. If one could know the detailed problematic features of other similar apps in advance, the explored pages can be re-ranked so that the bug-prone features can be explored earlier to facilitate the bugs being revealed earlier.

6.4. Threats to validity

The **external threats** concern the generality of the proposed approach. We train and evaluate MABLE on the dataset consisting of six apps from three categories. The selected apps and their belonging categories are all the commonly-used ones with rich reviews in practice. The time span of the data we select is one year (2019.8 $\sim$ 2020.8), which avoids the impact of a mass of similar reviews in a short period on the generality. These setups relatively reduce this threat.

Regarding **internal threats**, the loss function of MABLE is the weighted sum of losses of each single task. It is necessary to ensure that one single task cannot dominate the whole training process. Since we tuned the value of λ carefully, and obtained a relatively high performance on both tasks, we believe MABLE can alleviate the threat to some extent. Moreover, we reuse the source code from the original paper (e.g., *KEFE* and *Caspar*), or the open-source implementations (e.g., *TextCNN*, *TextRNN*, *BiLSTM-CRF*, and *SAFE*) for the baselines, which ensure the correctness of the experiments.

Construct validity mainly questions the evaluation metrics. We utilize precision, recall, and F1-Score to evaluate the performance of both tasks. In BC task, we use the weighted precision, recall, and F1 to evaluate the performance on multiple labels following existing practices [70,71]. In EPAF task, we consider that a problematic feature is correctly extracted when it is the same as the ground-truth, which is a rather strict measure.

7. Related work

7.1. App review classification

Existing studies have proposed various methods for app review classification. These studies classified app reviews into different types based on different perspectives, such as quality aspects [60], user actions [73], categories relevant to software maintenance and evolution [8], types of user issues [24], whether or not they include bug information, feature request or simply praise [12], whether they are informative [14], and whether reviews across different languages and platforms are similar [74]. These studies only focused on single classification task. Our proposed approach has the ability to share features from other related tasks (i.e., information extraction task) with MTL paradigm, which improves the generalization and boosts the performance.

7.2. App review information extraction

Many studies focused on the information extraction from app reviews considering the fact that reading through the entire reviews is impractical [3,5,7,9,13,15,16,23,51,75,76]. Some work [77,78] conducted empirical study, replicating the studies related to extracting features from app reviews, to confirm their results and to provide benchmarks. There are also studies [79,80] using existing feature extraction methods to conduct competitive analysis, e.g., identifying competing apps. Other information that can facilitate review understanding also included the types of complaints [9], user rationale [81] and summaries for guiding release planning [15], etc.

Some studies are similar to our work, but they do not support the extraction of fine-grained features well. For example, SAFE [3] proposed a simple method to extract feature-related phrases from app descriptions and app reviews with PoS patterns. PUMA [23] proposed a phrase-based approach to extract user opinions in app reviews with PoS templates. INFAR [16] mined insights from app reviews and generated summaries after classifying sentences into pre-defined topics. The discovered topics from INFAR were more coarse-grained (e.g., GUI, crash, etc.). Our method can highlight the fine-grained features (e.g., "push notification") that users complained about. SUR-Miner [7] and Caspar [5] used techniques, such as dependency parsing and Part-of-Speech pattern, to extract some aspects from app reviews. Guzman

et al. [51] proposed a method, which can only extract features consisting of two words (i.e., collocations) from the reviews based on word co-occurrence patterns, which is not applicable in our context, because the problematic features might contain multiple words. There are also some work [75,76] utilizing CRF-based models to extract features in the app reviews, but they did not combine any deep neural networks to capture textual semantics.

These pattern-based and CRF-based methods to extract the target phrases, did not consider the review semantics sufficiently, and had bad tolerance to noise. By comparison, we model the semantics of the review descriptions and review attributes to enable semantic-aware information extraction.

7.3. Multi-task learning in SE

Some studies have used MTL to simultaneously address two related tasks, or to improve target task by pre-training language model and vector representations on auxiliary tasks. For example, DEMAR [56] proposed a deep MTL-based model to jointly conduct requirements discovery task and requirements annotation task. MTFuzz [82] employed MTL to learn a compact embedding of program input spaces to guide the mutation process. Other studies about code analysis with MTL also included modeling source code with pre-trained language model for code understanding and generation [83], enabling knowledge sharing between related tasks for code completion [84], leveraging method name generation and method name informativeness prediction to improve code summarization [85], learning unified representation space for source code understanding tasks [86], learning a language model for auto code completion that combines learned structure and naming information [87], and automating seven commit-level vulnerability assessment tasks simultaneously [88], etc. This work addresses bug classification task and problematic app feature extraction task with MTL. Meanwhile, our approach is not a simple and direct application of MTL, the incorporation of review attributes and the modeling of textual semantics improve the MTL model further.

8. Conclusion

The true benefits of app review mining have largely been attributed to the types of knowledge classified and extracted to the developers. This paper proposes MABLE, a deep ***M***ulti-t***A***sk learning-based approach to automate the ***B***ug c***L***assification task and ***E***xtracting problematic app features. We evaluate MABLE on 3426 reviews from six apps, and the results confirm the effectiveness of the proposed approach on two related tasks, which significantly outperforms ten commonly-used and state-of-the-art baselines. Results also demonstrate the benefits of multi-task learning mechanism. The proposed approach provides an effective way to jointly learn two related review analysis tasks, and addresses the limitations of existing methods with improved performance. Our code and data are publicly available online at: https://github.com/MeloFancy/MABLE.

CRediT authorship contribution statement

Yawen Wang: Methodology, Software, Writing – original draft, Writing – review & editing. **Junjie Wang:** Supervision, Writing – review & editing. **Hongyu Zhang:** Supervision, Writing – review & editing. **Xuran Ming:** Software. **Qing Wang:** Supervision, Writing – review & editing.

Declaration of competing interest

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

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China Grant No. 62232016 and No. 62072442, Youth Innovation Promotion Association Chinese Academy of Sciences, Basic Research Program of ISCAS Grant No. ISCAS-JCZD-202304, and Major Program of ISCAS Grant No. ISCAS-ZD-202302.

Data availability

Data will be made available on request.

References

[1] E. Noei, D.A. da Costa, Y. Zou, Winning the app production rally, in: Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018, 2018, pp. 283–294.

[2] W.J. Martin, F. Sarro, Y. Jia, Y. Zhang, M. Harman, A survey of app store analysis for software engineering, IEEE Trans. Softw. Eng. 43 (9) (2017) 817–847.

[3] T. Johann, C. Stanik, A.M.A. B., W. Maalej, SAFE: A simple approach for feature extraction from app descriptions and app reviews, in: 25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017, 2017, pp. 21–30.

[4] A. Di Sorbo, S. Panichella, C.V. Alexandru, J. Shimagaki, C.A. Visaggio, G. Canfora, H.C. Gall, What would users change in my app? summarizing app reviews for recommending software changes, in: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016, 2016, pp. 499–510.

[5] H. Guo, M.P. Singh, Caspar: extracting and synthesizing user stories of problems from app reviews, in: ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020, 2020, pp. 628–640.

[6] Y. Man, C. Gao, M.R. Lyu, J. Jiang, Experience report: Understanding cross-platform app issues from user reviews, in: 27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, on, Canada, October 23-27, 2016, 2016, pp. 138–149.

[7] X. Gu, S. Kim, "What parts of your apps are Loved by users?", in: 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015, 2015, pp. 760–770.

[8] S. Panichella, A. Di Sorbo, E. Guzman, C.A. Visaggio, G. Canfora, H.C. Gall, How can i improve my app? Classifying user reviews for software maintenance and evolution, in: 2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015, 2015, pp. 281–290.

[9] H. Khalid, E. Shihab, M. Nagappan, A.E. Hassan, What do mobile app users complain about?, IEEE Softw. 32 (3) (2015) 70–77.

[10] M. Harman, Y. Jia, Y. Zhang, App store mining and analysis: MSR for app stores in: 9th IEEE Working Conference of Mining Software Repositories, MSR 2012, June 2-3, 2012, Zurich, Switzerland, 2012, pp. 108–111.

[11] F. Palomba, M.L. Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, A.D. Lucia, 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, 2015, pp. 291–300.

[12] W. Maalej, H. Nabil, Bug report, feature request, or simply praise? On automatically classifying app reviews in: 23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, on, Canada, August 24-28, 2015, 2015, pp. 116–125.

[13] P.M. Vu, T.T. Nguyen, H.V. Pham, T.T. Nguyen, Mining user opinions in mobile app reviews: A keyword-based approach in: 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015, 2015, pp. 749–759.

[14] N. Chen, J. Lin, S.C.H. Hoi, X. Xiao, B. Zhang, AR-miner: mining informative reviews for developers from mobile app marketplace in: 36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014, 2014, pp. 767–778.

[15] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, M. Di Penta, Release planning of mobile apps based on user reviews in: Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, 2016, pp. 14–24.

[16] C. Gao, J. Zeng, D. Lo, C. Lin, M.R. Lyu, I. King, INFAR: insight extraction from app reviews in: Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018, 2018, pp. 904–907.

[17] Y. Wang, J. Wang, H. Zhang, X. Ming, L. Shi, Q. Wang, Where is your app frustrating users? in: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022, IEEE, 2022, pp. 2427–2439.

[18] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[19] W. Zhang, H. Mei, H. Zhao, Feature-driven requirement dependency analysis and high-level software design Requir. Eng. 11 (3) (2006) 205–220.

[20] R. Collobert, J. Weston, A unified architecture for natural language processing: deep neural networks with multitask learning in: Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008, in: ACM International Conference Proceeding Series, vol. 307, ACM, 2008, pp. 160–167.

[21] J. Rao, F. Türe, J. Lin, 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, ACM, 2018, pp. 636–645.

[22] T. Zhang, B. Ghanem, S. Liu, N. Ahuja, 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, IEEE Computer Society, 2012, pp. 2042–2049.

[23] P.M. Vu, H.V. Pham, T.T. Nguyen, T.T. Nguyen, Phrase-based extraction of user opinions in mobile app reviews in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016, ACM, 2016, pp. 726–731.

[24] S. McIlroy, N. Ali, H. Khalid, A.E. Hassan, Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews Empir. Softw. Eng. 21 (3) (2016) 1067–1106.

[25] P. Terdchanakul, H. Hata, P. Phannachitta, K. Matsumoto, Bug or not? Bug report classification using N-gram IDF in: 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017, IEEE Computer Society, 2017, pp. 534–538.

[26] Y. Zhou, Y. Tong, R. Gu, H.C. Gall, Combining text mining and data mining for bug report classification in: 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014, IEEE Computer Society, 2014, pp. 311–320.

[27] G. Catolino, F. Palomba, A. Zaidman, F. Ferrucci, Not all bugs are the same: Understanding, characterizing, and classifying bug types J. Syst. Softw. 152 (2019) 165–181.

[28] P. Devine, Y.S. Koh, K. Blincoe, Evaluating software user feedback classifier performance on unseen apps, datasets, and metadata Empir. Softw. Eng. 28 (1) (2023) 26.

[29] R. Caruana, Multitask learning in: Learning To Learn, Springer, 1998, pp. 95–133.

[30] S. Ruder, An overview of multi-task learning in deep neural networks 2017, CoRR arXiv:1706.05098.

[31] R. Caruana, Multitask learning: A knowledge-based source of inductive bias in: Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993, Morgan Kaufmann, 1993, pp. 41–48.

[32] L. Duong, T. Cohn, S. Bird, P. Cook, Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers, The Association for Computer Linguistics, 2015, pp. 845–850.

[33] Y. Yang, T.M. Hospedales, Trace norm regularised deep multi-task learning in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings, OpenReview.net, 2017.

[34] Y. Zhang, H. Chen, Y. Zhao, Q. Liu, D. Yin, Learning tag dependencies for sequence tagging in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, ijcai.org, 2018, pp. 4581–4587.

[35] Z. Huang, W. Xu, K. Yu, Bidirectional LSTM-CRF models for sequence tagging 2015, CoRR arXiv:1508.01991.

[36] A. McCallum, D. Freitag, F.C.N. Pereira, Maximum entropy Markov models for information extraction and segmentation in: Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000, Morgan Kaufmann, 2000, pp. 591–598.

[37] A. Tang, D. Jackson, J. Hobbs, W. Chen, J.L. Smith, H. Patel, A. Prieto, D. Petrusca, M.I. Grivich, A. Sher, P. Hottowy, W. Dabrowski, A.M. Litke, J.M. Beggs, A maximum entropy model applied to spatial and temporal correlations from cortical networks in vitro J. Neurosci. 28 (2) (2008) 505–518.

[38] J. Felsenstein, G.A. Churchill, A Hidden Markov Model approach to variation among sites in rate of evolution Mol. Biol. Evol. 13 (1) (1996) 93–104.

[39] J.D. Lafferty, A. McCallum, F.C.N. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data in: Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001, Morgan Kaufmann, 2001, pp. 282–289.

[40] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P.P. Kuksa, Natural language processing (almost) from scratch J. Mach. Learn. Res. 12 (2011) 2493–2537.

[41] J. Howard, S. Ruder, Universal language model fine-tuning for text classification in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers, Association for Computational Linguistics, 2018, pp. 328–339.

[42] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), Association for Computational Linguistics, 2019, pp. 4171–4186.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 5998–6008.

[44] L. Xu, Q. Dong, C. Yu, Y. Tian, W. Liu, L. Li, X. Zhang, CLUENER2020: Fine-grained name entity recognition for Chinese 2020, arXiv preprint arXiv:2001.04351.

[45] C. Gao, J. Zeng, M.R. Lyu, I. King, Online app review analysis for identifying emerging issues in: Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, ACM, 2018, pp. 48–58.

[46] C. Gao, J. Zeng, X. Xia, D. Lo, M.R. Lyu, I. King, Automating app review response generation in: 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019, IEEE, 2019, pp. 163–175.

[47] M. Thelwall, K. Buckley, G. Paltoglou, Sentiment strength detection for the social web J. Assoc. Inf. Sci. Technol. 63 (1) (2012) 163–173.

[48] R. Socher, A. Perelygin, J. Wu, J. Chuang, C.D. Manning, A.Y. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, a Meeting of SIGDAT, a Special Interest Group of the ACL, ACL, 2013, pp. 1631–1642.

[49] M.R. Islam, M.F. Zibran, SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text J. Syst. Softw. 145 (2018) 125–146.

[50] R. Berrios, P. Totterdell, S. Kellett, Eliciting mixed emotions: a meta-analysis comparing models, types, and measures Front. Psychol. 6 (2015) 428.

[51] E. Guzman, W. Maalej, How do users like this feature? A fine grained sentiment analysis of app reviews in: IEEE 22nd International Requirements Engineering Conference, RE 2014, Karlskrona, Sweden, August 25-29, 2014, IEEE Computer Society, 2014, pp. 153–162.

[52] L. Ratinov, D. Roth, Design challenges and misconceptions in named entity recognition in: Proceedings of the Thirteenth Conference on Computational Natural Language Learning, CoNLL 2009, Boulder, Colorado, USA, June 4-5, 2009, ACL, 2009, pp. 147–155.

[53] H. Dai, P. Lai, Y. Chang, R.T. Tsai, Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization J. Cheminform. 7 (S-1) (2015) S14.

[54] Y. Zhu, R. Kiros, R.S. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, S. Fidler, Aligning books and movies: Towards story-like visual explanations by watching movies and reading books in: 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015, IEEE Computer Society, 2015, pp. 19–27.

[55] Viterbi algorithm 2014, https://en.wikipedia.org/wiki/Viterbi_algorithm.

[56] M. Li, L. Shi, Y. Yang, Q. Wang, A deep multitask learning approach for requirements discovery and annotation from open forum in: 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020, IEEE, 2020, pp. 336–348.

[57] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization in: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[58] J. Estdale, E. Georgiadou, Applying the ISO/IEC 25010 quality models to software product in: Systems, Software and Services Process Improvement - 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings, in: Communications in Computer and Information Science, vol. 896, Springer, 2018, pp. 492–503.

[59] J. Estdale, App stores & ISO/IEC 25000: Product certification at last? in: SQM XXIV: Systems Quality: Trends and Practices, 2016.

[60] E.C. Groen, S. Kopczynska, M.P. Hauer, T.D. Krafft, J. Dörr, Users - the hidden software product quality experts?: A study on how app users report quality aspects in online reviews in: 25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017, IEEE Computer Society, 2017, pp. 80–89.
[61] R. Kohavi, 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, 1995, pp. 1137–1145.
[62] G. Salton, C. Buckley, Term-weighting approaches in automatic text retrieval Inf. Process. Manag. 24 (5) (1988) 513–523.
[63] A. Mccallum, K. Nigam, A comparison of event models for naive Bayes text classification in: AAAI-98 Workshop on Learning for Text Categorization, 1998, pp. 41–48.
[64] V.N. Vapnik, The Nature of Statistical Learning Theory, Second Edition, Statistics for Engineering and Information Science, Springer, 2000.
[65] D.W. Hosmer, S. Lemeshow, Applied Logistic Regression, Second Edition, Wiley, 2000.
[66] A. Liaw, M. Wiener, A. Liaw, Classification and regression with random forest R News 23 (23) (2002).
[67] Y. Kim, 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, ACL, 2014, pp. 1746–1751.
[68] S. Lai, L. Xu, K. Liu, J. Zhao, 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, AAAI Press, 2015, pp. 2267–2273.
[69] H. Wu, W. Deng, X. Niu, C. Nie, Identifying key features from app user reviews in: 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021, IEEE, 2021, pp. 922–932.
[70] X. Luo, A.N. Zincir-Heywood, Evaluation of two systems on multi-class multi-label document classification in: Foundations of Intelligent Systems, 15th International Symposium, ISMIS 2005, Saratoga Springs, NY, USA, May 25-28, 2005, Proceedings, in: Lecture Notes in Computer Science, vol. 3488, Springer, 2005, pp. 161–169.
[71] D.M.W. Powers, Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation 2020, CoRR arXiv:2010.16061.
[72] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent Dirichlet allocation in: Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada], MIT Press, 2001, pp. 601–608.
[73] H. Liu, M. Shen, J. Jin, Y. Jiang, Automated classification of actions in bug reports of mobile apps in: ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020, 2020, pp. 128–140.
[74] E. Oehri, E. Guzman, Same same but different: Finding similar user feedback across multiple platforms and languages in: 28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020, IEEE, 2020, pp. 44–54.
[75] M. Sänger, U. Leser, S. Kemmerer, P. Adolphs, R. Klinger, SCARE the sentiment corpus of app reviews with fine-grained annotations in german in: Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, PortoroŽ, Slovenia, May 23-28, 2016, European Language Resources Association (ELRA), 2016.
[76] F.A. Shah, K. Sirts, D. Pfahl, Simulating the impact of annotation guidelines and annotated data on extracting app features from app reviews in: Proceedings of the 14th International Conference on Software Technologies, ICSOFT 2019, Prague, Czech Republic, July 26-28, 2019, SciTePress, 2019, pp. 384–396.
[77] J. Dabrowski, E. Letier, A. Perini, A. Susi, Mining and searching app reviews for requirements engineering: Evaluation and replication studies Inf. Syst. 114 (2023) 102181.
[78] J. Dabrowski, E. Letier, A. Perini, A. Susi, Mining user opinions to support requirement engineering: An empirical study in: Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings, in: Lecture Notes in Computer Science, vol. 12127, Springer, 2020, pp. 401–416.
[79] F.A. Shah, Y. Sabanin, D. Pfahl, Feature-based evaluation of competing apps in: Proceedings of the International Workshop on App Market Analytics, WAMA@SIGSOFT FSE, Seattle, WA, USA, November 14, 2016, ACM, 2016, pp. 15–21.
[80] F.A. Shah, K. Sirts, D. Pfahl, Using app reviews for competitive analysis: tool support in: Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics, WAMA@ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 27, 2019, ACM, 2019, pp. 40–46.
[81] Z. Kurtanovic, W. Maalej, On user rationale in software engineering Requir. Eng. 23 (3) (2018) 357–379.
[82] D. She, R. Krishna, L. Yan, S. Jana, B. Ray, Mtfuzz: fuzzing with a multi-task neural network in: ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020, ACM, 2020, pp. 737–749.
[83] F. Liu, G. Li, Y. Zhao, Z. Jin, Multi-task learning based pre-trained language model for code completion in: 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020, IEEE, 2020, pp. 473–485.
[84] F. Liu, G. Li, B. Wei, X. Xia, Z. Fu, Z. Jin, A self-attentional neural architecture for code completion with multi-task learning in: ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020, ACM, 2020, pp. 37–47.
[85] R. Xie, W. Ye, J. Sun, S. Zhang, Exploiting method names to improve code summarization: A deliberation multi-task learning approach 2021, CoRR arXiv:2103.11448.
[86] D. Wang, Y. Yu, S. Li, W. Dong, J. Wang, Q. Liao, MulCode: A multi-task learning approach for source code understanding in: 28th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021, Honolulu, HI, USA, March 9-12, 2021, IEEE, 2021, pp. 48–59.
[87] M. Izadi, R. Gismondi, G. Gousios, CodeFill: Multi-token code completion by jointly learning from structure and naming sequences in: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022, ACM, 2022, pp. 401–412.
[88] T.H.M. Le, D. Hin, R. Croft, M.A. Babar, DeepCVA: Automated commit-level vulnerability assessment with deep multi-task learning in: 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021, IEEE, 2021, pp. 717–729.