Evaluating Pre-Trained Models for User Feedback Analysis in Software Engineering: A Study on Classification of App-Reviews

Mohammad Abdul Hadi
Dept. of Computer Science
University of British Columbia
Kelowna, Canada
mohammad.hadi@ubc.ca

Fatemeh H. Fard
Dept. of Computer Science
University of British Columbia
Kelowna, Canada
fatemeh.fard@ubc.ca

***Abstract*—*Context:* Mobile app reviews written by users on app stores or social media are significant resources for app developers. Analyzing app reviews have proved to be useful for many areas of software engineering (e.g., requirement engineering, testing). Automatic classification of app reviews requires extensive efforts to manually curate a labeled dataset. When the classification purpose changes (e.g. identifying bugs versus usability issues or sentiment), new datasets should be labeled, which prevents the extensibility of the developed models for new desired classes/tasks in practice. Recent pre-trained neural language models (PTM) are trained on large corpora in an unsupervised manner and have found success in solving similar Natural Language Processing problems. However, the applicability of PTMs is not explored for app review classification.**

***Objective:* We investigate the benefits of PTMs for app review classification compared to the existing models, as well as the transferability of PTMs in multiple settings.**

***Method:* We empirically study the accuracy and time efficiency of PTMs compared to prior approaches using six datasets from literature. In addition, we investigate the performance of the PTMs trained on app reviews (i.e. domain-specific PTMs). We set up different studies to evaluate PTMs in multiple settings: binary vs. multi-class classification, zero-shot classification (when new labels are introduced to the model), multi-task setting, and classification of reviews from different resources. The datasets are manually labeled app review datasets from Google Play Store, Apple App Store, and Twitter data. In all cases, Micro and Macro Precision, Recall, and F1-scores will be used and we will report the time required for training and prediction with the models.**

***Index Terms*—pre-trained neural language models, app review classification**

I. Introduction

Mobile app users write about their experience on App Stores which can reveal important information about the app for software engineers. App reviews are studied extensively [1] and shown to be a helpful resource for requirement engineering [2], release planning [3], software maintenance [4], change file localization [5], and testing [6]. Filtering informative reviews [7], identifying bugs [8], classifying usability and quality concerns [2], and finding security and privacy issues [9] are among the topics that have been investigated. The list of topics of interest are numerous and there are several studies that use supervised machine learning to extract useful software engineering information from app reviews [1]. Supervised techniques require to be trained on a labeled dataset. Although app review analysis is shown to be useful for software developers in practice, the manual effort required to label datasets seems to be a barrier in many studies [1]. Recent works try to alleviate the problem by incorporating semi-supervised learning [10] or active learning [11]. However, when the classes of interest change, e.g. privacy issues [9], a new labeled dataset is required and previous curated datasets cannot be used.

Recently, Pre-Trained Models (PTM) are used in many applications from Natural Language Processing (NLP) to Computer Vision. In NLP, PTMs are large language models trained using a deep neural network in an unsupervised manner, which are used for many downstream tasks [12]. For example, BERT [13] is a PTM which is used for question answering and sentiment classification. Because the PTMs are trained on large corpora, they learn contextual information which can be fine tuned in the downstream tasks; meaning that PTMs eliminate the need to train models for downstream tasks from scratch [14]. In addition, PTMs reduce the amount of effort to build models for each task separately, and they reduce the amount of required labeled dataset [15]. As a result, PTMs are used for many of the transfer learning tasks and in settings where the model have not seen an example during training (referred to as zero shot learning) [16]. PTMs are used extensively and led to many advances in natural language processing, however, their applicability for software engineering is barely known. Only a few studies exist that explore the use of PTMs for sentiment analysis in software engineering [12], [17].

However, domain specific PTMs that are trained on domain specific data, show significant improvements over general purpose PTMs such as BERT (i.e. PTMs that are trained on general purpose data) for NLP tasks in fields such as science or law [18], [19]. These domain specific PTMs leverage unsupervised training on a large domain specific corpus to compensate the lack of high-quality, large-scale labeled data

This study was accepted at MSR 2021 Registered Reports Track.

in the specified domains. Similarly, the classification of app reviews aims to extract useful information for software engineers/app developers, which are domain specific information. In addition, app reviews are short text and many reviews are noisy and unrelated, which can introduce more complications. However, to what extent the PTMs can be useful in the context of app review classification, and whether domain specific PTMs trained on app reviews can improve the results, is unknown. Moreover, deep learning approaches using Convolutional Neural Networks (CNN) to implement transfer learning from non-contextual word embedding is shown to have comparable results with traditional machine learning approaches for classification of requirements related app reviews [20]. In contrast, another study using CNN shows improvements over previous approaches for app review classification [21]. Though these studies do not use PTMs, they use a deep learning approach which sheds doubt on whether PTMs can be useful.

In this study, we aim to explore the benefits of PTMs compared to the existing approaches for app review analysis, specifically app issue-classification tasks. We define the app issue-classification as the task of extracting useful information from users' feedback which can be requirements related, release planning related, and software maintenance related. The extracted information helps identifying different aspects, including feature requests, aspect evaluations (e.g., feature strength, feature shortcoming, application performance), problem reports, usability, portability, reliability, privacy and security, energy related, appraisals, inquiries about the application, etc. from app reviews. Our goal is to investigate accuracy and time efficiency of PTMs for classification of different datasets with various labels and multiple tasks (i.e. issue classification and sentiment analysis of app reviews). Therefore, experiments will be conducted in different settings. These experiments will provide baselines on the applicability of PTMs for app review analysis including cost of using them (in term of required time for predictions), and their capability to reduce the manual effort required for labeling large datasets. We expect that PTMs can achieve at least the same performance as the current approaches, as well as being beneficial to be used for multiple classification tasks. The contributions of this study are:

- This is the first study that explores the applicability of PTMs for automatic app issue classification tasks compared to the existing tools.
- We will conduct an extensive comparison between four PTMs and four existing tools/approaches on six different app review datasets with different sizes and labels.
- We are the first to explore the performance of general versus domain-specific pre-trained PTMs for app review classification.
- This is the first empirical study to examine the accuracy and efficiency of PTMs in four different settings: binary vs. multi-class classification, zero shot setting, multi-task setting, and setting in which training data is from one resource (e.g. App Store) and the model is tested on data from another platform (e.g. Twitter).

II. Research Questions

RQ1: *How accurate and efficient are the PTMs in the classification of app reviews compared to the existing tools?*

In this research question, we explore how the current PTMs perform compared to the existing tools, including their required time for training and prediction. The existing tools are based on curated rules, feature engineered machine learning algorithms, and deep learning models. The results can give us insights about the applicability of using PTMs in practice.

RQ2: *How does the performance of the PTMs change when they are pre-trained on app-review dataset, instead of a generic dataset (e.g. Wiki-documents, book corpus)?*

The current PTMs are trained using general text scraped from web. In some studies, domain specific models are trained e.g. LEGAL-BERT [19] and medical BERT [14] and have shown to improve the performance compared to the non-domain-specific models. In RQ2, we intend to explore the performance of a domain-specific pre-trained model when trained on app reviews. The results will help the researchers focus on building domain specific models (e.g. building app review models in languages other than English) versus using the general PTMs.

RQ3: *How do the PTMs perform in the following settings?*
(a) Binary vs multi-class setting,
(b) Zero-shot classification,
(c) Multi-task setting (i.e. different app-review analysis tasks),
(d) Classification of user-reviews collected from different resources (i.e., Twitter, App Store).

The answers to this research question will help us understand the applicability of PTMs and their performance in different settings (a), when labeled dataset is not available (b), the task changes (c), or different resources are used (d). In part (d) we explore the transferability of the models as the distributions of data on various platforms is different [22]. We expect that PTMs achieve reasonable results for these settings.

III. Research Materials

In this section, we briefly discuss about the considered datasets, prior approaches, and the pre-trained models.

A. Datasets

We conduct our study on six publicly available datasets.

1) ***Dataset 1*** *(D_1):* This dataset is procured by Gu and Kim [23] and contains reviews for 17 popular Android apps from Google Play in 16 most popular categories, such as games and social. The authors have manually labeled 2,000 reviews for each app (34,000 in total) in five classes according to their predefined rules: Aspect Evaluation (5,937), Praise (8,112), Feature Request (2,323), Bug Report (2,338), and Others (15,290).

2) ***Dataset 2*** *(D_2):* This dataset contains 6,406 app reviews from Google Play and 10,364 tweets (sampled from 5 million tweets written in English) which are labeled manually into three classes: Problem Report, Inquiry, and Irrelevant [20]. They included Problem Report (1,437), Inquiry (1,100), and Irrelevant (3,869) records from Google Play. The number of

records in each of these categories from Twitter data is 2,933, 1,405 and 6,026 respectively.

3) ***Dataset 3*** *(D_3):* This dataset is provided by Lu and Liang. [2]. Researchers selected two popular Apps, one from Apple App Store (iBooks in the books category) and one from Google Play (WhatsApp in the communication category). They sampled the raw reviews collected from each platform and manually classified 2,000 review sentences for each app (4,000 in total). The reviews are classified into six categories: Usability (432), Reliability (587), Portability (119), Performance (121), Feature Request (558), and Others (2,183).

4) ***Dataset 4*** *(D_4):* This dataset was procured by Maalej and Nabil [8] and contains 2,000 manually labeled reviews from random apps selected from top apps in different categories (1,000 from Apple App Store and 1,000 from Google Play), where half of the apps are paid and half of them are free apps. The authors ensured that there are 200 reviews for each of the 1, 2, 3, 4, and 5 stars in each of the 1000-reviews datasets. In addition, they provide 2,400 more reviews which are manually labeled. These reviews are for selected 3 random Android apps and 3 iOS apps from the top 100 apps (400 reviews for each app). This dataset contains 4,400 labeled reviews in total with four categories: Bug Report (378), Feature Request (299), User Experience (737), and Rating (2721).

5) ***Dataset 5*** *(D_5):* This dataset is published by Guo et al. [24] and contains 1,500 app reviews, which are manually labeled as User Action (428), App Problem (399), and Neither (673). The reviews are selected from over 5.8 million records about 151 apps from Apple App Store.

6) ***Dataset 6*** *(D_6):* This dataset was procured by Guzman et al. [25]; it contains reviews of 3 apps from Apple App Store and 4 apps from Google Play. The apps are popular and from diverse categories. They sampled 260 reviews per app and five annotators labeled the records, summing to 1,820 reviews that are manually labeled. This dataset includes seven categories, namely: Bug Report (990), Feature Strength (644), Feature Shortcoming (1281), User request (404), Praise (1703), Complaint (277), and Usage Scenario (593)[1].

Reasons behind choosing these Datasets (D_1 to D_6): We considered these datasets for their diverse characteristics from three perspectives, other than having different categories. *First*, the considered datasets were constructed from various repositories: Apple App Store, Google Play, and Twitter. *Second*, the size of the datasets vary and they have different number of records for each label. For example, *Dataset 2* is around four times larger than *Dataset 5*. Also, *Dataset 2* contains approximately 2,000 reviews per label, where *Dataset 1* has around 400 reviews per label. The size differences can provide insights about the ability of PTMs for different sizes (total training set and the available data for each label). *Third*, except *Dataset 1* and *Dataset 5* which are balanced dataset, the other four datasets are imbalanced. These differences ensure that we explore the capability of PTMs for different sizes, platforms, and in more realistic settings.

7) ***Merged Dataset*** *(D_7):* This dataset will be compiled by merging all the 6 datasets, D_1–D_6. This *Merged Dataset* has 55,933 app reviews with 16 labels. As some of the labels used in the datasets are the same or have similar definitions, we have grouped some of them. For grouping the labels, we have consulted the definitions of the classes from the studies published the datasets D_1–D_6. This dataset would be closer to practical applications, where a lot of irrelevant data exists and multiple classes of informative reviews are of interest to be extracted from this highly imbalanced dataset. The labels in the Merged Dataset, the grouped labels from D_1–D_6, and the number of app reviews per group are provided in Table I.

TABLE I
DESCRIPTION FOR MERGED DATASET (D7)

Labels in Merged Dataset	**Grouped Labels [Label Name: Dataset Initial (# of Reviews)]**	**# Total App Reviews**
Performance	Performance: D3 (121)	121
Portability	Portability: D3 (119)	119
Usability	Usability: D3 (432)	432
Reliability	Reliability: D3 (587)	587
Usage Scenario	Usage Scenario: D6 (593)	593
Feature Strength	Feature Strength: D6 (644)	644
User Experience	User Experience: D4 (737)	737
Feature Shortcoming	Feature Shortcoming: D6 (1,281)	1,281
Inquiry	Inquiry: D2 (1,100), User Action: D5 (428)	1,528
Problem	Problem Report: D2 (1,437), App Problem: D5 (399), Complaint: D6 (277)	2,113
Rating	Rating: D4 (2,721)	2,721
Bug Report	Bug Report: D1 (2,338), Bug Report: D4 (378), Bug Report: D6 (990)	3,706
Feature Request	Feature Request: D1 (2,323), Feature Request: D3 (558), Feature Request: D4 (299), User request: D6 (404)	3,584
Aspect Evaluation	Aspect Evaluation: D1 (5,937)	5,937
Praise	Praise: D1 (8,112), Praise: D6 (1703)	9,815
Irrelevant	Others: D1 (15,290), Irrelevant: D2 (3,869), Others: D3 (2,183), Neither: D5 (673)	22,015
Total		**55,933**

B. Prior Approaches

In this section, we briefly discuss the four widely-used approaches for app review analysis and the reasons of choosing them in our experiments. We have conducted literature review and reviewed analysis of app review surveys to select these prior approaches.

1) ***AR-Miner****:* Chen et al. proposed App Review Miner (AR-Miner) [7] which extracts valuable information from user reviews. Provided a collection of user reviews, AR-Miner first

[1]Note that adding numbers in all categories will exceed the total number, because some reviews belong to multiple groups. We will follow the steps in [25] to calculate the evaluation metrics for this dataset.

applies a pre-trained classifier that separates non-informative reviews. Then, AR-Miner applies Latent Dirichlet Allocation (LDA) over the informative reviews to chunk them into different groups for prioritizing them by an efficient ranking model proposed by the authors. AR-Miner has been widely used to mine app-issues from user-reviews [26]–[31]. Therefore, it is considered as a prior approach in our experiments.

Reason behind choosing AR-Miner: AR-Miner is the first framework to accommodate application developers in mining informative topics from a large volume of app reviews. This was the first effective attempt to leverage an unsupervised Topic Modeling technique to extract edifying issues from app reviews. Developers embraced AR-Miner for its ability to filter out non-informative reviews and display the key user feedbacks in an intuitive, concise manner. The effectiveness of AR-Miner was validated by conducting a comprehensive set of experiments on user reviews of four Android apps. We have included their dataset in our study. Researchers compared the AR-Miner results against real app developers' decisions in different studies [26]–[31]. They analyzed the advantages of AR-Miner over manual inspection and other techniques used in a traditional channel [32]–[35]. Based on the empirical results, AR-Miner has proven to be effective and efficient in extracting informative reviews.

2) ***SUR-Miner****:* Gu et al. [23] proposed Software User Review Miner (SUR-Miner), which is a framework that summarizes users' sentiments, opinions, and emotions toward different aspects of the application. SUR-Miner parses aspect-opinion pairs from reviews by considering their structures. For this purpose, it uses pre-defined sentence templates/patterns[2]. Then, for each review, it combines the sentiment of the sentences with its aspect-opinion pairs. These are used in the final step to summarize the software aspects. SUR-Miner generates reliable summaries and achieved high F1-score for aspect-opinion identification, sentiment analysis, and app review classification compared to the previous works [36], [37]. It is adopted in many studies [38]–[43] and therefore we choose it as a prior approach.

Reason behind choosing SUR-Miner: SUR-Miner is the first framework that fully took advantage of user reviews' monotonic structure and semantics; it also defined sentence patterns to extract aspect opinion pairs from app-review. Researchers carefully selected five distinct text features, specifically tailored for app reviews, and leveraged these features to train a supervised machine learning classifier. Based on the empirical evaluation, we choose this Machine Learning approach as the terminal classes categorized by the SUR-Miner model were shown to be significantly more accurate and precise than other methods, demonstrating its effectiveness [36], [37].

3) ***Ensemble Methods****:* Guzman et al. systematically evaluated different sets of ensemble methods and identified one for classifying user reviews [25]. They selected machine learning techniques that were used for text classification, and compared the performance of each algorithm individually, for app review classification. They also studied the performance of these models when their results are combined using ensemble methods. Ensemble methods are well known machine learning techniques as they can enhance the prediction performance of single classifiers, maintaining their strengths and reduce their vulnerabilities. In this work, Logistic Regression, Naive Bayes, Support Vector Machines, and Neural Network Classifier were grouped to vote for the final prediction. This ensemble method outperformed the individual algorithms with statistical significance. It was tested on a large dataset of seven diverse apps, where it shows that in all cases, this ensemble method either outperformed or matched its best baseline. Also, this method influenced further research in the field of app review analysis [3], [44]–[48] and therefore is used in our study.

Reason behind choosing Ensemble Methods: Different studies have attempted to identify the best Machine Learning approach for app issue classification without spending too much time perfecting custom-engineered features for different issues [2], [43], [49]–[52]. Guzman et al. [25] first leveraged different machine learning algorithms and combined their predictions to circumvent the problem of various machine learning models being used for classification of issues from app reviews. Their study showed that recall improved significantly after combining the predictions of Logistic Regression and Neural Networks. Software developers and software evolution experts widely adopted this approach to analyze user reviews and prioritize their tasks [53]–[56].

4) ***Deep Learning Model leveraging Non-contextual Word-embedding****:* Stanik et al. used a Deep Convolutional Neural Network (CNN) [20] for classifying app reviews. Their model contains an embedding layer and its weights are initialized with a word embedding model, e.g. word2vec or FastText. This CNN architecture outperforms the shallow Neural Networks classifying app reviews in the determined groups, with a small margin. This method uses non-contextual word embedding within a deep neural network and performed a transfer of knowledge representation. This approach is one of the first studies leveraging transfer learning and deep learning and is used in other studies for issue classification task [21], [57], [58], and therefore we choose it as a prior approach in our study.

Reason behind choosing this Deep Learning Approach: This study is a recent work and the purpose of [20] was to understand and evaluate the extent to which deep learning models could be used to categorize user feedback into different predefined categories. Based on this study, researchers determined that the domain experts' knowledge incorporation with traditional machine learning model could achieve comparable results to those of the deep learning approach due to the substantial performance improvements provided by using simple yet powerful features in the traditional machine learning techniques. Nevertheless, the developers and researchers widely adopted this deep learning approach as the improvement of pre-trained word embeddings bring considerable performance gain to the classifier in other domains [59]–[61].

[2]https://guxd.github.io/srminer/appendix.html

It is worth noting that the number of studies for app review classification is numerous. We choose these four prior approaches as they are well cited and widely used in many other studies. More importantly, they use different machine learning and deep learning approaches which can be representative of multiple techniques used in other studies. The best of these four approaches in terms of the performance metrics defined in Section IV-B will be chosen as the baseline.

We have already considered two more tools that we eliminated from our study: AR-Doc and CLAP. AR-doc uses a parser based on a NLP Classifier, which uses a set of 500 sentence-structures manually evaluated for four types of classes: Info. Giving, Info. Seeking, Feature Request, and Problem Discovery. To use AR-Doc, we will need to build different sets of sentence structures for each dataset and their respective labels, which is infeasible. In addition, we could not find any publicly available implementation of CLAP. We contacted the authors of CLAP but did not receive a response; and therefore unable to use CLAP.

C. Pre-Trained Models (PTM)

In this section, we briefly discuss our choice of the four PTMs. All these models have Transformer deep learning architecture which is based on attention mechanism [62]. As Transformer is currently the main architecture for PTMs [15], we choose the following Transformer PTMs: BERT, XLNet, RoBERTa, and ALBERT. The same PTMs are used for sentiment classification in software engineering [12].

*1) **BERT**:* Devlin et al. [13] designed BERT to learn contextual word representations from unlabeled texts. Contextual word embeddings designate a word's representation based on its context by capturing applications of words across different contexts. BERT employed a bidirectional encoder to learn the words' contextual representations by optimizing for Masked Language Model (MLM) and Next Sentence Prediction (NSP) tasks. For MLM, 15% of each sequence's words are replaced with a token ([MASK]) beforehand for BERT to attempt at predicting the masked words, based on the context provided by the non-masked words. For NSP, the model takes sentence-pairs as input for learning to predict pair-match is correct or wrong. During training, 50% of the inputs are true consequent pairs, while the other 50% are randomized non-consequent sentence-pairs. Devlin et al. trained two versions: small-sized $\text{BERT}_{\text{BASE}}$ and big-sized $\text{BERT}_{\text{LARGE}}$. $\text{BERT}_{\text{BASE}}$ is a smaller model with 12 layers and 110 million parameters. $\text{BERT}_{\text{LARGE}}$ on the other hand has 24 layers and 340 million parameters. $\text{BERT}_{\text{LARGE}}$ is more computationally expensive and consumes more memory compared to $\text{BERT}_{\text{BASE}}$. Thus, in this work, we will use $\text{BERT}_{\text{BASE}}$.

Reason behind choosing BERT: BERT advanced the state-of-the-art for 11 NLP tasks. It achieved an absolute improvement of 7.6% over the previous best score on General Language Understanding Evaluation (GLUE) benchmark[3]; it also achieved 93.2% accuracy on SQuAD 1.1[4], outperforming human performance.

*2) **XLNet**:* XLNet [62] uses Auto-Regressive (AR) language modeling and Auto-Encoding (AE). The model is "generalized" because it captures the bi-directional context using a mechanism called Permutation Language Modeling (PLM). XLNet integrates auto-regressive models and bi-directional context modeling, yet overcoming the disadvantages of BERT. PLM is the idea of capturing bidirectional context on all permutations of terms present in an input sequence. XLNet discards the one-directional linear modeling to maximize the log-likelihood over all permutations of the sequence-terms. Each position is expected to learn utilizing contextual information from the entire sequence, thereby capturing the bidirectional context. No [MASK] is needed, and input data need not be corrupted.

BERT neglects the dependency between masked positions. Consider the sentence "New Delhi is a city." Then, input to BERT to be "[MASK] [MASK] is a city," and the objective of BERT would be predicting "New" given "is a city" and predicting "Delhi" given "is a city". In this objective, there is no dependency between learning "New" and "Delhi." So, BERT can result in a prediction like "New Gotham is a city." If we assume that the current permutation is [is, a, city, New, Delhi], BERT would predict the tokens 4 and 5 independent of each other. Whereas, XLNet, predicts in the order of the sequence. i.e., first predicts token 4 and then predicts token 5: it computes the likelihood of "New" given "is a city" plus the likelihood of "Delhi" given "New, is a city".

Reason behind choosing XLNet: XLnet outperformed BERT on many NLP tasks; for 8 different tasks XLNet beat BERT by a substantial margin. This model achieved best results for 18 NLP tasks, including sentiment classification natural language inference.

*3) **RoBERTa (Robustly optimized BERT approach)**:* RoBERTa outperformed all the state-of-the-art benchmarks upon release [15]. Liu et al. modified BERT's pretraining steps that yield substantially better performance on all the classification tasks. RoBERTa increased the amount of mini-batch sizes, data, and training time to train the model. RoBERTa is also trained on dataset that includes longer sequences than before. The masking pattern in RoBERTa was also modified to be generated spontaneously.

Reason behind choosing RoBERTa: RoBERTa outperforms BERT on nine different NLP tasks on the GLUE benchmark; it also equals or exceeds XLNet's model in four out of nine individual tasks.

*4) **ALBERT (A Lite BERT)**:* ALBERT [63] applies three parameter reduction techniques: Factorized embedding parameterization, Cross-layer parameter sharing, and Inter-sentence coherence loss. In the first one, researchers separated the hidden layers' size from the input embeddings' size (previously of same sizes). They projected one-hot vectors to embedding

[3]https://gluebenchmark.com/

[4]Stanford Question Answering Dataset: https://rajpurkar.github.io/SQuAD-explorer/explore/1.1/dev/

and the hidden space with lower dimensions; it increased the hidden layer-size without significantly increasing the vocabulary embeddings' parameter size. Second, all the parameters across all layers are shared. The big-scale ALBERT model has substantially fewer parameters than $\text{BERT}_{\text{LARGE}}$. Finally, the NSP task is swapped with Sentence-Order Prediction (SOP) loss which help ALBERT perform better.

Reason behind choosing ALBERT: ALBERT uses parameter reduction techniques which resulted in having 18 times less parameters than BERT. Its' training time is 1.7 faster and has negligible inferior performance than the original $\text{BERT}_{\text{LARGE}}$ model. The much larger ALBERT architecture, which contains fewer parameters than $\text{BERT}_{\text{LARGE}}$, achieved higher results, F1 score of 92.2% and 89.4% on the SQuAD 2.0 and the GLUE benchmarks, respectively; it also achieved 89.4% accuracy on the ReAding Comprehension from Examinations benchmark (RACE benchmark).

IV. Execution Plan

A. Experimental Setup

Following [20], we will use k-fold cross validation on each dataset D_i separately, where D_i represents one of the seven datasets in our study. We use k-fold cross validation as a more rigorous approach than splitting the dataset into train and test set once [12]. Also, this method is used previously for app review classification in software engineering domain [64]. In the k-fold cross validation, we split a dataset D into k equal size disjoint parts/folds $D^{(1)}, ..., D^{(k)}$. We then build k classifiers $c^{(i)}$, each time using $k-1$ splits as training and one part $D^{(i)}$ as test set. This way, each split of the data can be used as test set once. As a result, we will have k different test set performances. As the datasets in the study are imbalanced, we will use an alternate option of cross validation called stratified k-fold cross validation [65], [66]. In this method, the only difference is that the distribution of the examples from each class in the original dataset is preserved in each fold $D^{(i)}$. The stratified k-fold cross validation is commonly used in machine learning practices as it reduces the experimental variance. Therefore, when comparing different methods, it is easier to identify the best method [65]. We consider $k = 5$ as this value has been shown empirically to have test error rate estimates that do not suffer from high bias or high variance [67]. The value of $k = 5$ is also used in previous app review classification studies [20]. For the evaluation, we use metrics explained in section IV-B. It is worth mentioning that we apply the stratified 5-fold cross validation on each of the datasets D_1–D_7 separately. For example, we split D_1 into 5 folds ${D_1}^{(1)}, ..., {D_1}^{(5)}$ and build 5 classifiers for D_1. We then compute the evaluation metrics for D_1. We continue this process for each of the datasets D_2–D_7 and report the performance on each of them separately.

Note that the number of labels in the datasets $D_1 - D_6$ ranges between [3,7] and all the considered prior approaches were designed for multi-class classification. Therefore, we can retrain and evaluate them using the stratified 5-fold cross validation. Among the prior approaches, only AR-Miner requires some adaptation. AR-Miner has three phases: classification of non-informative reviews, topic modeling of informative reviews, and ranking of the groups according to their relevance to developers' application. As all datasets D_1–D_7 are labeled, we will directly utilize the second step to group all the reviews (from both training and test set) in a dataset into the corresponding number of labels available. Following the literature [7], we will count the training set reviews in a resulting group and use the label with the majority count to annotate the group. This annotation indicates the class that the reviews in the group belong to. Then, we will examine the group's test set reviews to compute the evaluation metrics.

For evaluating the PTMs, for each dataset, we will use the training set to fine tune the PTMs and then evaluate them on the test set. We will obtain all the four PTMs that are pre-trained on general domain corpora from Hugging Face library[5]. To build a classifier, a feed-forward dense layer and Soft-max activation function will be added on top of each model. We will follow [12] and [13] to set the hyper-parameter values. In our work, we will set the batch size to 16 and Adam learning rate to $2e-5$. The models will be trained for 4 epochs and *AdamW* optimizer will be used for all models.

We plan to execute all the experiments on a Linux machine with Intel 2.21 GHz CPU and 16G memory. For training PTMs from scratch, we plan to use 2 $\times$ NVIDIA Tesla V100 32GB to enhance the parallelization performance.

B. Evaluation Metrics

In this section, we briefly describe the evaluation metrics. For all the classifications, we will use three metrics: Precision (P), Recall (R), F1-score (F1), and their micro and macro average values. To answer the time efficiency of the models, we will examine the training time for the prior approaches and the fine tuning time for PTMs. We will also report the prediction times and changes in time for all models.

Precision (P): Precision can be calculated by dividing the number of records that their labels are correctly predicted by total number of predicted observations in that class: $P = \frac{TP}{TP+FP}$. Here, TP refers to the number of records that their label is correctly predicted, and FP refers to the the number of records falsely predicted to belong to this class. In multi-class classification, for each group *A*, all the observations that belong to other labels and are falsely predicted as group *A* are added to compute the FP.

Recall (R): For each group *A*, Recall can be calculated by dividing the number of accurately predicted observations in *A* by the number of all observations available in the corresponding class: $R = \frac{TP}{TP+FN}$ Here, FN is the number of observations in class *A* which are falsely predicted as other labels.

F1-Score (F1): F1 Score is the weighted average of Precision and Recall:

$$\text{F1} = \frac{2 \cdot (P \cdot R)}{P+R} \tag{1}$$

[5]https://huggingface.co/

As we have multi-class classification and the datasets are imbalanced, we use micro-average and macro-average metrics as follows. Here, the micro-average calculates the contribution of all records in all classes (therefore the contribution of the class with the predominant number of records is taken into account), whereas the macro-average is the average of the values for each class (therefore, each class contributes equally in the final value). The micro- and macro-averaged precision (P) are computed as:

$$P_{micro} = \frac{\sum_{j=1}^{m} TP_j}{\sum_{j=1}^{m} TP_j + \sum_{j=1}^{m} FP_j} \quad (2)$$

$$P_{macro} = \frac{\sum_{j=1}^{m} P_j}{m} \quad (3)$$

TP_j and FP_j are the number of true positive and false positive predictions for the j-th class, respectively. P_j is the precision for class j and m is the number of classes. Similarly, the micro- and macro-average of Recall (R) and F1-score will be calculated and denoted as R_{micro}, R_{macro}, $F1_{micro}$, and $F1_{macro}$.

As we are using stratified 5-fold cross validation, for each dataset D, we will report averages of these metrics obtained from each of the k classifiers:

$$F1^{avg}{}_{micro} = 1/k \sum_{n=1}^{k} F1_{micro}{}^{(i)} \quad (4)$$

$$F1^{avg}{}_{macro} = 1/k \sum_{n=1}^{k} F1_{macro}{}^{(i)} \quad (5)$$

The $F1_{micro}{}^{(i)}$ and $F1_{macro}{}^{(i)}$ refer to the test performance of classifier $c^{(i)}$ on held out test set $D^{(i)}$. Similarly, the $P^{avg}{}_{micro}$, $P^{avg}{}_{macro}$, $R^{avg}{}_{micro}$, and $R^{avg}{}_{macro}$ will be computed.

Following previous works [12], *if a model has higher values for both $F1^{avg}{}_{micro}$ and $F1^{avg}{}_{macro}$, we consider it to be better than other models.*

Time: To compare the time efficiency, we will report the training time for other approaches and fine-tuning time for PTMs, and the prediction time of all the approaches. Prediction time is the time the model requires to process the test data and predict labels. The time will be measured in seconds and will be reported for each dataset. *A model is considered to be more time efficient than another if the prediction time is less;* as training a model or fine-tuning a PTM is a one-time process and need not be repeated.

Time Change in Percentage: We will employ all four prior approaches on all datasets D_1–D_7 and choose their best as our *baseline* approach. In addition to the Time mentioned above, we will also report the increase and decrease in percentage for the studied PTMs with respect to the considered baseline for the measured Time. The reason of reporting this change in percentage is that time-duration depends on hardware configuration, and percentage change provides a better understanding about the change in time for the readers.

C. Study Design

The following steps will be taken to answer each research question.

1) RQ1 - Accuracy and Efficiency of PTMs: All prior approaches and PTMs will be trained and tested on all of the datasets separately, as explained in section IV-A. We only consider the reviews (not tweets) from *Dataset 2* for RQ1. For each dataset and approach, all the evaluation metrics described in section IV-B will be reported. We will highlight the best model for each dataset with highest $F1^{avg}{}_{micro}$ and $F1^{avg}{}_{macro}$. The lowest training/fine-tuning and prediction times will also be highlighted for each dataset, along with the time change in percentages. We denote the best performing PTM among others as `PTM-X` to pursue experiments in other research questions. If no PTM achieves the best results in terms of both micro and macro F1-score, the PTM that achieves the highest $F1^{avg}{}_{micro}$-score (denoted as `PTM-XM1`) and the PTM with the highest $F1^{avg}{}_{macro}$-score (denoted as `PTM-XM2`) will be used in RQ2.

2) RQ2 - Domain-Specific PTMs vs. General PTMs: In RQ2, we are interested to evaluate the performance of PTMs when they are pre-trained on domain-specific corpus rather than non-domain-specific corpora. In RQ2, we will use `PTM-X` (from RQ1) and all the six datasets. We can either pre-train `PTM-X` using just the domain-specific app reviews or combine them with the general domain documents to pre-train `PTM-X` from scratch. In the first approach, `PTM-X` with the same architecture will be trained from scratch on app review sentences. But pre-training `PTM-X` on a small number of domain-specific documents has its disadvantages; the model may become less adaptable and overfit on our dataset. Therefore, we will use another approach [14], where we will pre-train the `PTM-X` simultaneously with both general and domain-specific documents. Following the literature [14], we will double the frequency of pre-training domain-specific documents during the optimization on the Masked Language Modeling (MLM) task. We will pre-train three different `PTM-Xs`, where we will respectively integrate 2.8 million, 5.6 million, and 10 million app review sentences that we have collected from Google Play, with the general domain documents from Wiki-texts. Our collected dataset has already 5.6 million reviews with over 10 million sentences for more than 400 apps from different categories. The dataset includes app_name, app_category, review, rating, reply_text, and date. This will give us three versions of a domain-specific pre-trained model which we refer to as *Customized PTM-X* (`CPTM-X`). We denote the three sizes of the `CPTM-X` as $\texttt{CPTM-X}_{\texttt{base}}$, $\texttt{CPTM-X}_{\texttt{medium}}$, and $\texttt{CPTM-X}_{\texttt{large}}$, respective to the number of the app review sentences integrated for pre-training. Finally, we will fine-tune all the `CPTM-Xs` on the six datasets according to the steps and evaluation metrics explained in RQ1. The best `CPTM-X` will be used in RQ3.

3) RQ3 - Extent of PTMs' Capacity: We are considering four different settings to determine the PTMs' capacity regarding app review analysis.

(i) *Binary vs multi-class setting:* Previous studies have shown that the pre-trained Transformer based models deviate from their original performance when the classification task involves multiple classes instead of binary classes [68], [69].

Studies on app review classification also show that binary classification yields better results [49], [70]. We are, therefore, interested in evaluating the performance of the PTMs in both binary and multi-class settings, especially, investigating the performance of CPTM-X. We will investigate the accuracy and time efficiency of PTMs for binary classification compared to the original multi-class settings. We will convert all the six datasets into binary-class datasets by randomly choosing a class (i.e. label) for each dataset and switching all the other labels in that dataset into "Others". Then, we will use these modified datasets to regenerate the PTM results for the binary classification task on each dataset. The four PTMs and CPTM-X will be used to record the evaluation metrics. All other steps are similar to RQ1.

(ii) *Zero-shot classification setting:* The zero shot learning (ZSL) that we will investigate in our study is the recent approach in which we evaluate the models on fully-unseen-labels. In this setting, we assume that the system is never trained on any labeled data for the task we are interested in, which is different from some settings in which labels are partially seen by the data. The setting we choose is more realistic and can provide insights about potential benefits of PTMs in practice, where the developers might want to use the same model to derive new unseen aspects of the dataset. Using PTMs for this kind of zero shot classification requires a different training.

For ZSL, we will explore the method proposed by Yin et al. [16] in which classification is considered as a natural language inference (NLI) task. This approach determines the compatibility of two distinct sequences by embeddeing both sequences and labels into the same space. In NLI, a pair of sentences are considered: "premise" and "hypothesis"; and the task is to predict whether the "hypothesis" is an entailment of "premise" or not (contradiction). We follow the steps in [16] to prepare the datasets and models and set up our study for ZSL. We will use the reviews as "premise" and candidate labels as "hypothesis". So, the NLI model can predict whether the hypothesis is an entailment of the premise. This prediction will be compared to the actual labels for the reviews to calculate the evaluation metrics. We will fine tune PTM-X on MultiNLI dataset[6] and test it on the six datasets to determine how well PTMs can classify issues from app-reviews without being trained on them. In addition, we will use RoBERTa-large-nli[7], which is a readily available fine tuned model for zero-shot classification.

(iii) *Multi-task setting:* Research has confirmed that when a new classification task (e.g. sentiment classification) is introduced, a new model should be trained and the models for another classification task (e.g. app issue classifiers) cannot be applied for this new task. In addition, the same model or classification technique can have different performance on various analysis tasks. For example, Hemmatian and Sohrabi found Naive Bayes (a probabilistic classifier) to be performing better than Decision Tree (a non-probabilistic classifier) for sentiment classification tasks [71]; on the other hand, Maalej et al. reported that using Bag-of-Word technique and Decision Tree outperformed Naive Bayes for app issue classification task [49]. Therefore, we are interested in evaluating the performance of PTMs in both app issue classification and sentiment analysis task settings. In sentiment analysis, the task is to classify the sentiment of a given review into positive, neutral, or negative polarities [12], which is different than the app issue classification. In this setting, we aim to evaluate the PTM-X and CPTM-X on the new task of sentiment classification for app reviews. We will use another dataset[8] that consists of 37,185 app reviews with three sentiment polarity labels: *positive* (13,758), *neutral* (9,993), and *negative* (13,434). The experimental settings and evaluation metrics we will use are similar to RQ1. The results will determine how these models perform for a different classification task in the same domain (app reviews).

(iv) *Multiple resources:* The distribution of the data from multiple resources varies and models built for App Stores are not suitable to classify user feedbacks from another resource such as Twitter [42]. In this setting, we will study the accuracy and time efficiency of the PTMs for classification of data collected from multiple resources. For this experiment, *Dataset 2* will be used as it contains App Store reviews and Twitter data with same labels. First, we will fine-tune the PTM-X and CPTM-X on App Store reviews of *Dataset 2* and report the evaluation metrics (training steps are similar to RQ1) when tested on Twitter data. In addition, we will fine-tune the PTM-X and CPTM-X on Twitter part of *Dataset 2* and report the results when tested on App Store reviews. This way, we will gain insights about the PTMs' capacity to classify app reviews collected from different resources for which the PTMs were not fine-tuned on.

V. Threats to validity

A. Internal Validity

A possible internal validity can be related to the obtained results. To mitigate this threat, we use stratified k-fold cross-validation to avoid the bias that might be introduced to the results by the test set. In addition, to mitigate threats to the validity of the results, hyper-parameter values will be kept the same for all PTMs and Deep learning models in all the steps of experiments. We will also keep the values of alpha and beta for LDA as reported in AR-Miner. We run all the experiments on a single machine, and we report the machine-configuration to enforce the reproducibility of the results. Furthermore, we consider the same metrics to compare the PTMs with prior approaches. Along with the generated raw duration, we will also provide the change of time in percentage as compared to the baseline. Our adopted stricter micro and macro-metrics diminish the ambiguity while providing comparisons among the considered approaches' associated results.

[6]https://cims.nyu.edu/ sbowman/multinli/

[7]https://huggingface.co/joeddav/xlm-roberta-large-xnli

[8]https://www.kaggle.com/lava18/google-play-store-apps

B. Construct Validity

The selection of the prior approaches and PTMs can pose a validity threat to our study. We identified the four most common approaches as priors by examining the highly practiced methods, tools, and techniques employed by researchers and application developers. The considered approaches include different Machine Learning algorithms, Ensemble methods, and Deep Learning Approaches for app issue classification tasks and are selected after conducting literature review. The PTMs were adopted by following previous study [12] that conducted an empirical study on PTMs performance for sentiment analysis in Software Engineering. Additionally, we pre-trained transformer-based models by incorporating app-review-related datasets with the previously used generic dataset to better understand the potential of domain specific PTMs (`CPTM-X`) for app issue classification. Another threat to the study can be related to dataset D_7. This dataset is a merge of other datasets and has labels borrowed from them. Although we consulted the definitions of the labels from their publishers, there might be a chance that the samples in one group of a dataset is closely related to the samples from another label of another dataset, thus, affecting the results. We will investigate the results of D_7 closely to mitigate potential threats.

C. External Validity

In this study, we empirically study the ability of the PTMs in issue classification of app reviews. The app review classification mainly focuses on extracting useful information in the context of software engineering; which can be used by app developers for requirements engineering, release planning, and other tasks as mentioned in the paper. Our study is therefore limited to app review classification for software engineers and we do not study PTMs for other purposes such as finding issues from business perspectives [72], product reviews [73], and applications such as intention mining [74]. We also do not study summarization of relevant issues from app reviews. However, we conduct our study on six different datasets. In addition, by merging them, we run experiments on D_7 which has multiple classes covering different aspects of apps useful for app developers. In the future, we plan to curate datasets labelled with security and energy consumption (related to batter issues) issues and conduct the same experiments on these datasets to extend the external validity of the results. Another threat in the generalizability of the results lies in different tasks we considered. We only use sentiment classification for the alternate task in the multitask setting. Although PTMs *might* be useful for other tasks, we do not study them in this work. To ensure the generalizability of our study in the specified scope, we have incorporated datasets that include app reviews from diverse app categories and from two platforms i.e., Google Play and Apple App Store. The datasets have various sizes and different labels. In addition, we experiment on issue classification from another platform, i.e. Twitter, which has been shown to require different models than App Stores due to the differences of the platforms, noise data, and user feedback.

REFERENCES

[1] J. Dabrowski, E. Letier, A. Perini, and A. Susi, "App review analysis for software engineering: A systematic literature review," University College London, Tech. Rep., dec 2020.

[2] M. Lu and P. Liang, "Automatic classification of non-functional requirements from augmented app user reviews," in *21st EASE*, ser. EASE'17. New York, NY, USA: ACM, 2017, p. 344–353.

[3] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *2017 IEEE 24th SANER)*. IEEE, 2017, pp. 91–102.

[4] A. Al-Hawari, H. Najadat, and R. Shatnawi, "Classification of application reviews into software maintenance tasks using data mining techniques," *Software Quality Journal*, pp. 1–37, 2020.

[5] Y. Zhou, Y. Su, T. Chen, Z. Huang, H. C. Gall, and S. Panichella, "User review-based change file localization for mobile applications," *IEEE Transactions on Software Engineering*, 2020.

[6] G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, and H. C. Gall, "Exploring the integration of user feedback in automated testing of android applications," in *2018 IEEE 25th SANER*. IEEE, 2018, pp. 72–83.

[7] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "Ar-miner: Mining informative reviews for developers from mobile app marketplace," in *36th ICSE*, ser. ICSE 2014. NY, USA: ACM, 2014, p. 767–778.

[8] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *2015 IEEE 23rd International RE*, 2015, pp. 116–125.

[9] A. R. Besmer, J. Watson, and M. S. Banks, "Investigating user perceptions of mobile app privacy: An analysis of user-submitted app reviews," *IJISP*, vol. 14, no. 4, pp. 74–91, 2020.

[10] R. Deocadez, R. Harrison, and D. Rodriguez, "Preliminary study on applying semi-supervised learning to app store analysis," in *Proceedings of the 21st ICEASE*, 2017, pp. 320–323.

[11] V. T. Dhinakaran, R. Pulle, N. Ajmeri, and P. K. Murukannaiah, "App review analysis via active learning: reducing supervision effort without compromising classification accuracy," in *2018 IEEE 26th IRE*. IEEE, 2018, pp. 170–181.

[12] T. Zhang, B. Xu, F. Thung, S. A. Haryono, D. Lo, and L. Jiang, "Sentiment analysis for software engineering: How far can pre-trained transformer models go?" in *2020 IEEE ICSME*, 2020, pp. 70–80.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[14] S. Wada, T. Takeda, S. Manabe, S. Konishi, J. Kamohara, and Y. Matsumura, "A pre-training technique to localize medical bert and enhance biobert," *arXiv preprint arXiv:2005.07202*, 2020.

[15] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

[16] W. Yin, J. Hay, and D. Roth, "Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach," 2019.

[17] E. Biswas, M. E. Karabulut, L. Pollock, and K. Vijay-Shanker, "Achieving reliable sentiment analysis in the software engineering domain using bert," in *2020 IEEE ICSME*. IEEE, 2020, pp. 162–173.

[18] I. Beltagy, K. Lo, and A. Cohan, "SciBERT: A pretrained language model for scientific text," in *Proceedings of the 2019 EMNLP-IJCNLP*. Hong Kong, China: ACM, Nov. 2019, pp. 3615–3620.

[19] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, N. Aletras, and I. Androutsopoulos, "LEGAL-BERT: The muppets straight out of law school," in *Findings of the ACL: EMNLP 2020*. Association for Computational Linguistics, Nov. 2020, pp. 2898–2904.

[20] C. Stanik, M. Haering, and W. Maalej, "Classifying multilingual user feedback using traditional machine learning and deep learning," in *2019 IEEE 27th REW*, 2019, pp. 220–226.

[21] N. Aslam, W. Y. Ramay, K. Xia, and N. Sarwar, "Convolutional neural network based classification of app reviews," *IEEE Access*, vol. 8, pp. 185 619–185 628, 2020.

[22] S. Ruder and B. Plank, "Learning to select data for transfer learning with bayesian optimization," in *Proceedings of the 2017 Conference on Empirical Methods in NLP*. ACL, sep 2017, pp. 372–382.

[23] X. Gu and S. Kim, "What parts of your apps are loved by users? (t)," in *2015 30th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*. Lincoln, Nabraska: IEEE/ACM, 2015, pp. 760–770.

[24] H. Guo and M. P. Singh, "Caspar: extracting and synthesizing user stories of problems from app reviews," in *2020 IEEE/ACM 42nd ICSE*. IEEE, 2020, pp. 628–640.

[25] E. Guzman, M. El-Haliby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution (n)," in *2015 30th IEEE/ACM International Conference on ASE*, 2015, pp. 771–776.

[26] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," in *2015 IEEE ICSME*. IEEE, 2015, pp. 281–290.

[27] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *2016 IEEE/ACM 38th ICSE*. IEEE, 2016, pp. 14–24.

[28] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 2016 24th ACM SIGSOFT*, 2016, pp. 499–510.

[29] G. Bavota, M. Linares-Vasquez, C. E. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "The impact of api change-and fault-proneness on the user ratings of android apps," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 384–407, 2014.

[30] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User reviews matter! tracking crowd sourced reviews to support evolution of successful apps," in *2015 IEEE ICSME*. IEEE, 2015, pp. 291–300.

[31] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk, "Automatically discovering, reporting and reproducing android application crashes," in *IEEE ICST*. IEEE, 2016, pp. 33–44.

[32] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "App store analysis: Mining app stores for relationships between customer, business and technical characteristics," *RN*, vol. 14, no. 10, p. 24, 2014.

[33] T. Johann, C. Stanik, W. Maalej *et al.*, "Safe: A simple approach for feature extraction from app descriptions and app reviews," in *2017 IEEE 25th International RE*. IEEE, 2017, pp. 21–30.

[34] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing bug reports for android applications," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 673–686.

[35] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang, "Feature lifecycles as they spread, migrate, remain, and die in app stores," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. IEEE, 2015, pp. 76–85.

[36] K. Yatani, M. Novati, A. Trusty, and K. N. Truong, "Review spotlight: a user interface for summarizing user-generated reviews using adjective-noun word pairs," in *Proceedings of the SIGCHI*, 2011, pp. 1541–1550.

[37] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *2014 IEEE 22nd International RE*. IEEE, 2014, pp. 153–162.

[38] Y. Zhao, X. Xu, and M. Wang, "Predicting overall customer satisfaction: Big data evidence from hotel online textual reviews," *International Journal of Hospitality Management*, vol. 76, pp. 111–121, 2019.

[39] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *2017 IEEE/ACM 39th ICSE*. IEEE, 2017, pp. 106–117.

[40] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "Crowdsourcing user reviews to support the evolution of mobile apps," *Journal of Systems and Software*, vol. 137, pp. 143–162, 2018.

[41] A. Di Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. Canfora, "Surf: summarizer of user reviews feedback," in *2017 IEEE/ACM 39th ICSE-C*. IEEE, 2017, pp. 55–58.

[42] E. Guzman, R. Alkadhi, and N. Seyff, "An exploratory study of twitter messages about software applications," *Requirements Engineering*, vol. 22, no. 3, pp. 387–412, 2017.

[43] E. Bakiu and E. Guzman, "Which feature is unusable? detecting usability and user experience issues from user reviews," in *2017 IEEE 25th International REW*. IEEE, 2017, pp. 182–187.

[44] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE transactions on software engineering*, vol. 43, no. 9, pp. 817–847, 2016.

[45] E. Guzman, R. Alkadhi, and N. Seyff, "A needle in a haystack: What do twitter users say about software?" in *2016 IEEE 24th International Requirements Engineering Conference (RE)*. IEEE, 2016, pp. 96–105.

[46] E. Guzman, M. Ibrahim, and M. Glinz, "A little bird told me: Mining tweets for requirements and software evolution," in *2017 IEEE 25th International RE Conference*. IEEE, 2017, pp. 11–20.

[47] M. Ali, M. E. Joorabchi, and A. Mesbah, "Same app, different app stores: A comparative study," in *2017 IEEE/ACM 4th International Conference on MOBILESoft*. IEEE, 2017, pp. 79–90.

[48] S. Scalabrino, G. Bavota, B. Russo, M. Di Penta, and R. Oliveto, "Listening to the crowd for the release planning of mobile apps," *IEEE Transactions on Software Engineering*, vol. 45, no. 1, pp. 68–86, 2017.

[49] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requir. Eng.*, vol. 21, no. 3, p. 311–331, Sep. 2016.

[50] C. Wang, T. Wang, P. Liang, M. Daneva, and M. van Sinderen, "Augmenting app review with app changelogs: An approach for app review classification." in *SEKE*, 2019, pp. 398–512.

[51] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging AI Applications in Computer Engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[52] C. Wang, F. Zhang, P. Liang, M. Daneva, and M. van Sinderen, "Can app changelogs improve requirements classification from app reviews? an exploratory study," in *Proc. of the 12th ACM/IEEE Int. Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–4.

[53] M. Stade, F. Fotrousi, N. Seyff, and O. Albrecht, "Feedback gathering from an industrial point of view," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 71–79.

[54] A. AlSubaihin, F. Sarro, S. Black, L. Capra, and M. Harman, "App store effects on software engineering practices," *IEEE Transactions on Software Engineering*, 2019.

[55] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the content of github readme files," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1296–1327, 2019.

[56] J. Dabrowski, E. Letier, A. Perini, and A. Susi, "Finding and analyzing app reviews related to specific features: A research preview," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2019, pp. 183–189.

[57] R. K. Singh, M. K. Sachan, and R. Patel, "360 degree view of cross-domain opinion classification: a survey," *Artificial Intelligence Review*, pp. 1–122, 2020.

[58] C. Stanik, "Requirements intelligence: On the analysis of user feedback," Ph.D. dissertation, Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky, 2020.

[59] X. Yang, C. Macdonald, and I. Ounis, "Using word embeddings in twitter election classification," *Information Retrieval Journal*, vol. 21, no. 2, pp. 183–207, 2018.

[60] N. Reimers, B. Schiller, T. Beck, J. Daxenberger, C. Stab, and I. Gurevych, "Classification and clustering of arguments with contextualized word embeddings," *arXiv preprint arXiv:1906.09821*, 2019.

[61] Y. Ren, Y. Zhang, M. Zhang, and D. Ji, "Improving twitter sentiment classification using topic-enriched multi-prototype word embeddings," in *Proceedings of the AAAI Conf. on AI*, vol. 30, no. 1, 2016.

[62] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," 2020.

[63] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," 2020.

[64] N. Novielli, D. Girardi, and F. Lanubile, "A benchmark study on sentiment analysis for software engineering research," in *Proceedings of the 15th International Conference on MSR*, ser. MSR '18. New York, NY, USA: ACM, 2018, p. 364–375.

[65] G. Forman and M. Scholz, "Apples-to-apples in cross-validation studies: Pitfalls in classifier performance measurement," *SIGKDD Explor. Newsl.*, vol. 12, no. 1, p. 49–57, Nov. 2010.

[66] Y. M. Haibo He, *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press, 2013.

[67] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[68] A. Adhikari, A. Ram, R. Tang, and J. Lin, "Docbert: Bert for document classification," *arXiv preprint arXiv:1904.08398*, 2019.

[69] W.-C. Chang, H.-F. Yu, K. Zhong, Y. Yang, and I. Dhillon, "X-bert: extreme multi-label text classification with using bidirectional encoder

representations from transformers," *arXiv preprint arXiv:1905.02331*, 2019.

[70] N. Jha and A. Mahmoud, "Mining non-functional requirements from app store reviews," *Empirical Software Engineering*, pp. 1–37, 2019.

[71] F. Hemmatian and M. K. Sohrabi, "A survey on classification techniques for opinion mining and sentiment analysis," *Artificial Intelligence Review*, vol. 52, no. 3, pp. 1495–1545, 2019.

[72] A. K. Tang, "A systematic literature review and analysis on mobile apps in m-commerce: Implications for future research," *Electronic Commerce Research and Applications*, vol. 37, p. 100885, 2019.

[73] W. Zhao, Z. Guan, L. Chen, X. He, D. Cai, B. Wang, and Q. Wang, "Weakly-supervised deep embedding for product review sentiment analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 1, pp. 185–197, 2017.

[74] Q. Huang, X. Xia, D. Lo, and G. C. Murphy, "Automating intention mining," *IEEE Transactions on Software Engineering*, vol. 46, no. 10, pp. 1098–1119, 2018.

This figure "fig1.png" is available in "png" format from:

http://arxiv.org/ps/2104.05861v1