Noname manuscript No.
(will be inserted by the editor)

Evaluating Software User Feedback Classifier Performance on Unseen Apps, Datasets, and Metadata

Peter Devine · Yun Sing Koh · Kelly Blincoe

Received: date / Accepted: date

Abstract Understanding users' needs is crucial to building and maintaining high quality software. Online software user feedback has been shown to contain large amounts of information useful to requirements engineering (RE). Previous studies have created machine learning classifiers for parsing this feedback for development insight. While these classifiers report generally good performance when evaluated on a test set, questions remain as to how well they extend to unseen data in various forms.

This study evaluates machine learning classifiers' performance on feedback for two common classification tasks (classifying bug reports and feature requests). Using seven datasets from prior research studies, we investigate the performance of classifiers when evaluated on feedback from different apps than those contained in the training set and when evaluated on completely different datasets (coming from different feedback channels and/or labelled by different researchers). We also measure the difference in performance of using channel-specific metadata as a feature in classification.

We find that using metadata as features in classifying bug reports and feature requests does not lead to a statistically significant improvement in the majority of datasets tested. We also demonstrate that classification performance is similar on feedback from unseen apps compared to seen apps in the majority of cases tested. However, the classifiers evaluated do not perform well on unseen datasets. We show that multi-dataset training or zero shot classification approaches can somewhat mitigate this performance decrease. We discuss the implications of these results on developing user feedback classification models to analyse and extract software requirements.

Peter Devine · Kelly Blincoe
Human Aspects of Software Engineering Lab, University of Auckland, New Zealand
E-mail: pdev438@aucklanduni.ac.nz; k.blincoe@auckland.ac.nz

Yun Sing Koh
School of Computer Science, University of Auckland

Keywords Software user feedback, User feedback classification, Unseen data domains, Machine learning, requirements engineering, Software quality

Conflict of interest

None of the authors listed have a declared conflict of interest related to this work.

1 Introduction

Software product quality is deeply tied to user satisfaction and the extent to which the product meets the users' needs[12]. To that end, Requirements Engineering (RE) is considered key to developing high-quality software which meets users' needs [4,26]. Recent research has found explicit online software user feedback (such as app reviews, Tweets, and forum posts) to be a rich source of information for understanding users' needs and the software requirements associated with those needs. For example, Pagano et al. showed that more than 30% of 1100 manually analysed reviews on mobile app stores contained requirements relevant data that can then be leveraged by developers to improve their product [27]. Similarly, feedback channels such as Twitter posts [15], forum posts [36], Reddit posts [1], Facebook posts [34], and Steam reviews [22], have also been shown to contain helpful insights to guide the development and maintenance of software.

Studies have proposed classification methods to automate the ingestion and analysis of this feedback to help identify software requirements [16,17, 18,28]. These methods are largely underpinned by machine learning models which require manually labelled example data to train on. Human annotators give labels such as "bug report" or "feature request" to each piece of feedback. These feedback-label pairs are then used as a training dataset to train a model to label feedback into one of these classes automatically.

The utility of these classifiers is multifaceted. Proposals have been made for using classifiers to help developers understanding their users' requirements by integrating user feedback into the development cycle. An example of this is MARA (Mobile App Review Analyzer), which classifies app store reviews into "feature request" and "bug report", and these reviews are then used to inform software design and maintenance [18]. Commercial solutions, such as MonkeyLearn[1], also exist. In addition to aiding software teams to identify requirements, these classifiers can also be used in research studies. The classifiers can be trained on labelled training data and applied to large unlabelled datasets. This can enable researchers to study the requirements relevant characteristics of a large set of feedback, for example as was done by Nayebi et al. analysing user feedback on Twitter [25].

[1] https://monkeylearn.com/

Despite classifiers' widespread use throughout the literature, questions remain as to how effective they are at classifying user feedback for a given software project if they have not been trained on a representative sample of manually labelled feedback about that project. Tizard et al. demonstrated that the popular ARDOC (App Reviews Development Oriented Classifier) user feedback classifier, which has been trained on user feedback from app store reviews, does not perform well when applied to forum post feedback [36]. Many of these classification techniques also rely on using feedback channel metadata (e.g., app review rating) as input features for classifiers - metadata which may not be available when applying such a classifier to another channel. While much user feedback can be similar across the RE literature (e.g. bug reports can be found across app reviews [18,28], Tweets [25,13], forum posts [36], and Reddit posts [19]), it can also be given for different reasons and by different people depending on the feedback channel [35]. It is therefore unknown how different these feedback sources are, making it unclear as to how a classifier trained on one channel would perform on another. Applying classifiers to feedback from unseen apps (i.e. apps of which none of its feedback was included in the classifier's training set) has also not been explicitly explored within the literature, which makes it unclear as to how they would perform on this unseen app feedback. If these classifiers are not able to correctly classify feedback from new apps and new channels to a reasonably satisfactory degree, then they only have limited practical use in supporting the software development cycle.

We investigate the robustness of user feedback classifiers over separate apps, domains, and features. To do this, we focus on three aspects of user feedback classifier training: training and testing both with and without metadata-based features, training and testing on separate apps, and training and testing on separate datasets.

Firstly, feedback metadata such as review ratings and app categories have also been used in many studies as a feature in classification. However, the effect of including metadata in classification on performance across multiple datasets is not fully known. Indeed, few studies examine the effect of each type of metadata to isolate the effect of metadata on overall performance. Therefore, we train classifiers both using and not using metadata as features to determine the change in performance with their inclusion. Understanding the relative importance of metadata on feedback classification informs how applicable they are to different sources which may have different metadata available.

Secondly, many reported classification performance statistics are based on training and evaluating on user feedback from the same app, which leaves the expected performance of these classifiers on unseen apps unclear. Therefore, this study examines the difference in classification performance between models trained and tested on feedback from separate apps, and trained and tested on the same apps. This is done to investigate how well models can classify feedback from unseen apps.

Finally, many public datasets of user feedback exist from prior studies. These datasets contain feedback from various feedback channels (e.g. app store

reviews, tweets, forum posts) and are labelled using various label sets. However, these datasets often contain labels that are similar as labels in the other datasets (e.g. “bug”, “bug report”, and “error” labels from three separate datasets). One labeller’s definition of, for example, a bug report may differ from another’s. Therefore, we evaluate whether a classifier trained on the labels of one dataset transfer to labelling another dataset. This will provide an understanding on the ability of user feedback classification models to generalise to new domains or slightly different labelling schemas.

These aims resulted in the following three research questions:

- **RQ1:** How does training and testing with *metadata* affect classification F1 scores?
- **RQ2:** How does training and testing on feedback from separate *apps* affect classification F1 scores?
- **RQ3:** How does training and testing on feedback from separate *datasets* affect classification F1 scores?

We answer these questions by evaluating the classification performance of state-of-the-art text classifiers under different data configurations using seven datasets from the literature. We find that using metadata as a feature in a classifier tends not to improve classification of bug reports and feature requests in the majority of cases. We also find little difference in classification performance between feedback from seen and unseen apps, but a large drop in performance on unseen datasets.

This paper first outlines the previous work related to user feedback classification in Section 2. The datasets used in this work are then detailed in Section 3 and the method used to train the classifiers is explained in Section 4. The results of the evaluation on these datasets is described in Section 5, and the implications of these results are described in Section 6. Finally, threats to validity are considered in Section 7.

The replication package for this work can be found online[2].

2 Related work

This section first details the effect that Requirements Engineering (RE) has on software quality, before exploring the applications of machine learning in RE within the literature. Finally, it gives examples of previous studies regarding the evaluation of different machine learning techniques in RE.

2.1 Software quality and requirements engineering

The benefits gained from RE have been shown to be integral to software quality. Many models exist within the literature that include RE to improve software quality [4,5]. The practice of RE has also been shown to be beneficial,

[2] https://doi.org/10.5281/zenodo.5733504

with Damian and Chisan demonstrating that productivity, quality, and risk management were all improved when effective RE was done within a commercial software project [8]. Similarly, Radliński showed that multiple RE factors had a positive, statistically significant effect on software quality factors within a literature dataset of thousands of software projects [29]. A survey of developers also showed that teams that used RE approaches were much more likely to say that their product's capabilities fit their customers' needs well and that end users found their products easy to use than those which did not [20]. Other RE-adjacent concepts such as requirements traceability have also been shown to positively impact software quality [30].

2.2 Machine learning in requirements engineering

Machine learning has become a common tool used within the requirements engineering literature for supporting the creation of requirements. Approaches like the one proposed by Cleland-Huang et al. [7] have been made to integrate automated text classifiers into the requirements engineering process. The MARA model, developed by Iacob et al. [18] focuses on developing requirements from the ingestion of online user feedback using such a text classifier. These classifiers have become prevalent throughout the literature, with a systematic literature review by Lim et al. showing that 38 out of 63 studies which did user feedback analysis based on manually labelled data used machine learning to classify this feedback [21]. One of the potential reasons for this popularity is the reported high classification performance of some of these machine learning models.

Work from Maalej et al. [23], Panichella et al. [28], and Stanik et al. [33] all report bug report classification F1 scores of app reviews higher than 0.75, with Maalej et al. reporting as high as 0.9. However, Stanik et al. also report a bug report classification score of 0.59 in Tweets, with Nayebi et al. also reporting lower feature request classification F1 scores of 0.67 [25]. These diverse values highlight that the expected classification performance can vary dramatically depending on data source, classification method, and evaluation method. This underscores the need to rigorously compare and standardise techniques for both training and evaluating classification models.

2.3 Comparisons of classification techniques

There are several studies within the literature that compare techniques used for classifying user feedback. Work by Aurajo et al. evaluated the performance of four classical machine learning classifiers on classifying user feedback from one dataset using both term frequency derived features and features from deep pre-trained language models, showing that deep pre-trained language models generate superior text embedding features compared to frequency-based features for classification [2]. Similarly, Henao et al. demonstrated the increase in

performance in user feedback classification when using pre-trained language models over both classical models as well as other deep models [17]. Hadi and Fard proposed a study where the classification accuracy of pre-trained language models is compared against that of previously constructed classifiers from the literature as well as exploring the effect of self-supervised pre-training, binary classification, multi-class classification, and zero-shot settings on classification performance [16]. Dhinakaran et al. showed that models trained on training data that was chosen randomly were found to consistently underperform more sophisticated training data selection techniques, such as active learning [10]. Di Sorbo et al. investigated the correlation between app review rating and feedback type classifier prediction, finding that predictions of "problem discovery" from the ARDOC classifier were negatively correlated with the app rating, whereas predictions of "feature request" were uncorrelated [11]. As can be seen, there has been extensive work evaluating which text-based features and machine learning models are best to use when classifying user feedback. Some work has also been done to improve the data-efficiency of training a classifier. What remains unclear is how different training and evaluation methods affect the evaluation result of these classifiers (particularly on out-of-domain data), and how features apart from text affect classification performance.

This study adds to the literature by exploring the effect of several machine learning techniques to highlight where and when user feedback classifiers can and cannot be used in the real world. Firstly, the use of metadata for classification across multiple domains is evaluated to determine its effect on user feedback performance. Secondly, evaluating on seen and unseen app reviews is evaluated, in order to determine how well user feedback classifiers perform in classifying feedback for an unseen app. Finally, classifiers trained and tested on separate datasets are evaluated so as to determine how well classifiers can be applied to similar data.

3 Datasets

To measure different training and evaluation techniques, seven unique datasets from six studies were used in our evaluation. The datasets studied vary in size, feedback label set, and feedback channel, coming from app reviews, Twitter, and forum posts. This variance among datasets was chosen partly to evaluate how well these classifiers perform across different domains and different labellers. The selection of these datasets was done by first studying a broad collection of previous work on user feedback, with any that linked to a publicly available user feedback dataset then being further considered. Then, only those datasets that had manually labelled user feedback (i.e. not classifier labelled) which contained classes analogous to either "bug report" or "feature request" were used within our investigation.

From these seven datasets, it was found that all seven shared a "bug report" or similar class, and six shared a "feature request" or similar class. Therefore,

ID	Domain	Size	No. apps	Bug label (%)	Feature label (%)	Meta-data	Bug F1	Feature F1
A	Reviews	1,565	27	Error (30.2%)	None	None	NR	NA
B	Reviews	4,385	7	Bug report (22.6%)	User request (9.2%)	Rating	0.81	0.51
C	Reviews	1,438	48	Bug (9.5%)	Feature (12.4%)	Rating	0.88	0.85
D	Reviews	2,986	705	Bug (25.5%)	Feature (11.1%)	Rating, App category	0.9	0.72
E	Reviews	707	14	Bug (69.3%)	Feature (22.9%)	None	NR	NR
F	Forums	2,652	2	Apparent bug (15.3%)	Feature request (4.4%)	Post position, Topic	0.725 to 0.728	0.83
G	Twitter	3,907	10	Bug (27.1%)	Feature (24.2%)	None	0.78	0.66

Table 1 Details of the 6 datasets used in our evaluation. (NR - not reported)
"No. apps" is the number of unique apps contained within the dataset. "Bug label (%)" is the name of the bug report label in the dataset and the percentage of that dataset which has this label. "Feature label (%)" is the name of the feature request label in the dataset and the percentage of that dataset which has this label.

comparison of classification across datasets was done on a binary basis for these two classes. This section describes each dataset, with Table 1 summarizing and comparing the broad statistics of each dataset.

The datasets included in this analysis are taken from replication packages linked in:

- **Dataset A** from Ciurumelea et al. [6][3]
- **Dataset B** from Guzman et al. [14]
- **Dataset C** from Maalej et al. [23]
- **Dataset D** from Scalabrino et al. [32][4]
- **Dataset E** from Scalabrino et al. [32][4]
- **Dataset F** from Tizard et al. [36]
- **Dataset G** from Williams et al. [37]

Each dataset consists of publicly available user feedback which has been scraped from the internet, before being manually labelled by the researchers of their respective studies. The smallest of these datasets has 707 pieces of feedback, while the largest has 4,385. The datasets span three distinct user

[3] This dataset contains feedback that is labelled as "Error". While the classification of this class of feedback is not reported on in the paper, we use this class as our bug report class.

[4] The replication package contains two datasets referring to research questions 1 and 3 from this study, of which the latter is a pre-filtered set of feedback (filtered to contain only requirements relevant feedback) used to measure clustering performance, rather than classification. Therefore, while no classification metrics are reported for this RQ3 dataset (Dataset E), we still use it for training and testing models.

RQ	Training configuration	Trained on	Tested on
RQ1	Single mixed	Dataset α mixed train and validation sets	Dataset α mixed test set
	Single separated	Dataset α separated train and validation sets	Dataset α separated test set
RQ2	Single out-of-dataset	Dataset α mixed train and validation sets	Dataset δ mixed test set
	Leave one out	Dataset α, β, γ mixed train and validation sets	Dataset δ mixed test set
RQ3	Single mixed (text and metadata)	Dataset α mixed train and validation sets (text and metadata)	Dataset α mixed test set (text and metadata)

Table 2 Example permutations of datasets for RQ1, RQ2, and RQ3 for example datasets α, β, γ, and δ.

feedback domains: app store reviews, forum posts, and tweets. The maximum number of distinct apps within a dataset was 705, while the smallest was two.

4 Method

To answer our research questions, we created training, validation and test sets applicable to each experiment, before using the training and validation sets to train state of the art text classifiers. Finally, we evaluated these models on test sets to get performance scores for each experiment.

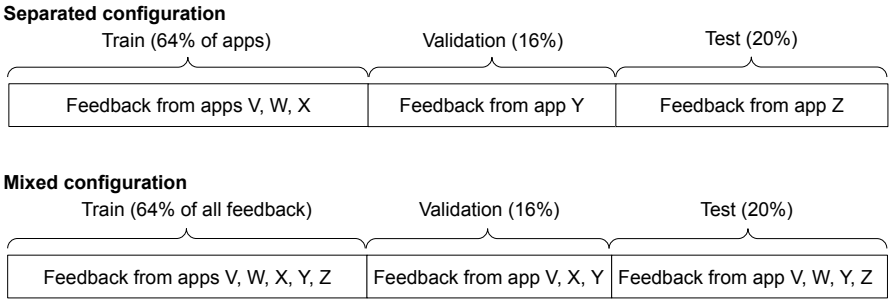

Fig. 1 Diagram visualising how the train, validation, and test splits are created for both the mixed and separated datasets of RQ2.

4.1 Data handling

Within this study, the performance of classifiers on feedback from apps that it had not been trained on is analysed, and as such, the information as to which app a piece of feedback came from is needed for each dataset. For this reason

each dataset was cleaned so that any rows which did not have an app identifier (i.e. are null) were dropped. This only affected dataset C, where some rows did not contain app identification data, and accounted for 1,821 out of 3,259 (55.9%) rows within that dataset, leaving 1,438 pieces of feedback from that dataset to use in our experiments.

Two different configurations of generating train, validation and test data splits were then used for each dataset, and these were named "mixed" and "separated". The "mixed" data splits were created by randomly splitting all feedback across the dataset into a 64:16:20 (train : validation : test) split. These ratios were chosen to be an initial 80:20 (train+validation : test) split before splitting the train + validation set with a further 80:20 split. This random splitting of data into train, validation, and test sets is current standard practice throughout the user feedback classification literature.

The "separated" data splits were generating by randomly sampling apps within a dataset and assigning them to either the train, validation, or test split. All feedback for a given app was put into the same data split. Again, we aimed for a 64:16:20 (train : validation : test) ratio while ensuring each of these splits contained data from different apps. By sampling at the app level instead of the feedback level, we are not able to guarantee that the amount of feedback in each split strictly adheres to the 64:16:20 ratio, but we mitigate this potential discrepancy by using cross validation in all our experiments. Cross validation in our experiments splits the data into 5 random variations of the train, validation, and test partitions of the overall dataset, and the mean of the metrics across each variation are then taken as our evaluation metrics. Multiple samples of this partition reduce the likelihood that one skewed partition impacts the final evaluation metrics. "Separated" data splits were not created for Dataset F because it included only two apps, and so contained too few distinct apps to split into train, validation, and test sets. A visual depiction of how these two configurations were created can be seen in Fig. 1.

The "mixed" and "separated" configurations were achieved by using Sci Kit Learn's ShuffleSplit and GroupShuffleSplit respectively. Cross validation was used to generate 5 distinct data folds for each dataset, and reported metrics in our results are the mean over these 5 folds.

4.2 Model

4.2.1 Training machine learning models

We trained machine learning models based on state-of-the-art pre-trained language models, which have been shown to achieve higher classification performance than other models [17]. These models require text input to be tokenized before they can be trained.

Tokenization

To train and evaluate deep pre-trained language model based classifiers, we first tokenized all feedback text. Each piece of feedback was broken down into

a series of token IDs, which each corresponding to either a word or a part of a word from a learned vocabulary. This was done to efficiently encode text into a set of one-hot-vectors while maintaining the model's ability to handle previously unseen words. Tokenization was done using Huggingface's Tokenizers library in Python [5] using the "bert-base-cased" version of the "BertTokenizer" tokenizer, which has a vocabulary size of 28,996 unique tokens. This results in the creation of input IDs, an attention mask, and token type IDs for every piece of feedback. Each piece of feedback is also prepended by a [CLS] token and appended by a [SEP] token to denote the start and end of a piece of text. These values are then fed into the model for training or inference. We supply metadata tags to the model in the form of special tokens. All possible metadata tokens (i.e. those contained in train, validation, and test splits) are passed to the tokenizer when it is initialized such that it does not tokenize these features and that all metadata tags are valid input IDs when running training and inference.

Model training

The pre-trained language model used in our experiments was the "distilbert-base-cased" version of the "BertForSequenceClassification" model from Huggingface's Transformers [6] library in Python. This model variant was chosen due to it's relative high performance on general natural language tasks compared to larger language models [31] and because a smaller model allowed for more reasonable training times for the high number of models created within the constraints of this study.

The BertForSequenceClassification model generates a binary class probability for each piece of feedback. This is done by inputting the token IDs derived from the feedback as described in section 4.2.1 into a pre-trained transformer based language model, which outputs a fixed length vector at each token position. The vector values at the first token position (i.e. the [CLS] token position) were then passed to one linear neural network layer of two nodes which represent "true" and "false" for the class that we were training on. The output of these logits was finally passed to a softmax layer, which normalises the input logits such that they sum to 1 to generate probabilities for the "true" and "false" binary class. The equation for this calculation can be found in Equation 1.

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{1}$$

Where $Softmax(x_i)$ is the probability for class i (either true or false), $\exp(xi)$ is the exponential of the logit output of the linear layer for class i, and $\sum_j \exp(x_j)$ is the sum of the exponentials of the logits of both true and false.

These modules of language model, linear layer, and softmax layer form the model that we use as our classifier in all of our experiments. At training time, the output probabilities of a given piece of feedback (whether it is or is

[5] https://huggingface.co/docs/tokenizers
[6] https://huggingface.co/transformers

not a bug report, feature request etc.) is then compared to the true label of that piece of feedback. The difference between the probabilities and the true label is then back propagated through the model, updating the weights of the language model and linear layer such that it trains to our training data. At evaluation time, the model's output probabilities of each piece of feedback in the validation and testing set are converted into labels by choosing the most probable class (true or false) based on the softmax output. For example, if a bug classifier outputs that a piece of feedback is 60% likely a bug report (and so 40% that it is not a bug report), then we consider that the model has predicted that this feedback is a bug report, and then we compare this label to the true label using evaluation metrics such as F1 score (described in Section 4.3).

Each model is trained for 500 steps with a batch size of 128 (128 pieces of feedback training the model at each step). Every time all feedback has been used to train the model (i.e. each epoch) the model is evaluated on the validation set. The weights of the model at the epoch with the highest associated F1 score on the validation set were loaded after training and saved for use in evaluating on the test set. The choice of training for 500 steps was made as it was observed that both the smallest and largest datasets had safely peaked in validation set F1 score by that point.

A Trainer object from Huggingface's Transformers was used to train the model, into which we set a training batch size of 128. All other hyperparameters were left as default for the Trainer (initial learning rate = 5e-05, weight decay = 0, adam beta 1 = 0.9, adam beta 2 = 0.999, adam epsilon = 1e-08) as there was little observed difference in performance when these were changed.

4.2.2 Zero shot classifier

A zero shot classifier model is a text classifier that does not require any training data before being used. In our work, we use the "bart-large-mnli" model developed by Facebook as our zero shot model due to its performance and popularity on the HuggingFace model portal[7]. This model was not explicitly trained to classify user feedback, but has been designed to classify text without pre-training on class-labelled training data (hence "zero-shot") by leveraging the entailment prediction abilities of natural language inference models, as proposed by Yin et al. [38]. Therefore, this classifier relies on the textual content of the label as well as the text content of the feedback, and so requires model labels to categorise text into. For the text input for the zero shot classifier, we use the phrase "bug report" for classifying bug reports, and "feature request" for classifying feature requests. This classifier outputs a classification probability score between 0 and 1, rather than a simple label. We set our probability threshold at 0.5, and so predict a piece of feedback as a bug report or feature request if the zero shot classifier outputs a greater than 50% probability for that label.

[7] https://huggingface.co/models?pipeline_tag=zero-shot-classification

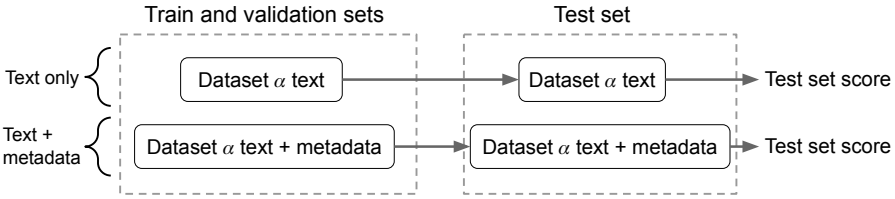

Fig. 2 Diagram visualising how the training and testing is done using text, and text with metadata for datasets in RQ1.

4.3 Evaluation metrics

Each dataset was evaluated using the F1 metric. The equation for this metric can be found in equations 2, 3, and 4, with "No. True Positives" being the number of bug reports or feature requests that were correctly classified, "No. False Positives" is the number of pieces of feedback that were classified as a bug report or feature request, but actually were not, and "No. False Negatives" is the number of pieces of feedback that were classified as not being a bug report or feature request, but actually were.

$$precision = \frac{No.\,True\ Positives}{No.\,True\ Positives + No.\,False\ Positive} \tag{2}$$

$$recall = \frac{No.\,True\ Positives}{No.\,True\ Positives + No.\,False\ Negative} \tag{3}$$

$$F1 = 2.\ \frac{precision\ .\ recall}{precision + recall} \tag{4}$$

This metric provides a good measure of how well a class is being correctly labelled due to it balancing recall and precision, and is less sensitive to class imbalances in the data compared to the accuracy metric.

Statistical significance between the performances of different classifier types were determined by using an independent two-sample t-test on the F1 metrics all folds of cross validation for one given test dataset.

4.4 Training and evaluation

4.4.1 Metadata (RQ1)

To determine the difference in performance of classifiers that use both metadata and text against those which use only text, we train two different models for every fold of every dataset, one which just receives feedback text as input, and one which also receives feedback metadata. As in RQ2, the "Single dataset - mixed" splits of data were used for these evaluations. A visual representation of this training and evaluation can be found in Fig. 2

Metadata was added as a feature to the model by prepended feedback text with metadata tags before being passed to the model, in order to allow the classifier to process the metadata information with the feedback textual information within the deep layers of the language model. Continuous metadata such as the number of followers of a tweet author was binned into 5 quintiles such that individual binned categories could be learned by the model, rather than requiring a new metadata token for each unique continuous value, which would make generalisation almost impossible. Metadata tags were then generated using the format as specified in Equation 5, such that an app review that has an associated rating of 3 stars would be prepended with the tag "[METADATA_rating_3]" or a forum post written by the thread original author would have the tag "[METADATA_is_original_thread_author_TRUE]".

```
[METADATA_ + metadata column name + _ + metadata value + ]   (5)
```

Each metadata tag is added to the text tokenizer as a special token so as to prevent it from being broken up upon tokenization. We trained models using all metadata available to us from their datasets. This includes metadata that was used as features when making classifiers in the original studies associated with these datasets. App review metadata included app rating (a simple rating of an app on a 5 point scale) and the category of the app. Forum post metadata consisted of the position of the comment within the forum post thread (with 0 being the original post in the thread, 1 being the first reply, etc.), a boolean of whether the comment author is the author of the thread's original post, the topic of the forum, and the level of the user (i.e. the user's experience level on the forum). Twitter metadata contained the number of total favorites the Tweet author has, the number of followers of the Tweet author, the number of people the author follows, the number of media tweets the author has made, whether the Tweet is a reply to someone else, and whether the user is verified by Twitter (usually reserved for public figures). Full details of all metadata used for each dataset can be found in Table 3. After training on a given train and validation set, each model was evaluated on the respective test set. **"Text only"** denotes the evaluation results of the model which was trained and tested using only text features. **"Text and metadata"** denotes the evaluation results of the model which was trained and tested using both text and metadata features.

4.4.2 Unseen Apps (RQ2)

In order to determine the difference in performance between evaluating on feedback from unseen apps compared to seen apps, we trained models on the cross validation folds of each dataset's training and validation sets for both "separated" and "mixed" configurations, for evaluation on their respective test sets. In our results, we denote these models as **"Single dataset - separated"** and **"Single dataset - mixed"**.

Dataset	Metadata
A	App rating (numeric)
B	App rating (numeric)
C	App rating (numeric)
D	App rating (numeric), App category (categorical)
E	App rating (numeric)
F	Post position within forum thread (numeric), Is commenter original thread author (boolean), Forum topic (categorical), User level (categorical)
G	No. of favorites (numeric), No. of followers (numeric), No. of friends (numeric), No. of statuses (numeric), No of media tweets (numeric) Is tweet a reply (boolean) Is user verified (boolean)

Table 3 List of metadata used for datasets in RQ3

4.4.3 Unseen datasets (RQ3)

To find the classification ability of classifiers trained on one dataset before being applied to another, we used the models trained on "Single dataset - mixed" splits from RQ2 as the literature standard is to evaluate classifiers on mixed-app dataset splits. These were then evaluated on each dataset except the one that it was trained on. In our results, **"Train A-F"** denotes the models which have been trained on one of these datasets and is then evaluated on all others. In addition, we also trained a "leave-one-out" (denoted **"LOO"**) model for each data split, where all datasets except one were used to train a model, and then evaluated on the excluded dataset. A visual representation of how this training was done can be found in Fig. 3.

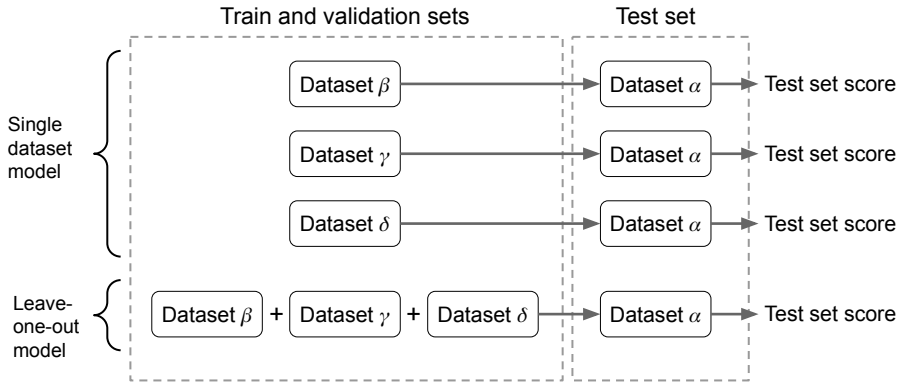

Fig. 3 Diagram visualising how the training and testing is done on different datasets in RQ2.

Dataset	Bug classification			Feature request classification		
	Text only	Text and Metadata	T-test stat.	Text only	Text and Metadata	T-test stat.
A	0.764	0.817	-2.059	-	-	-
B	0.725	0.761	-3.270 *	0.468	0.475	-0.168
C	0.357	0.432	-1.481	0.544	0.535	0.171
D	0.856	0.857	-0.155	0.653	0.669	-0.698
E	0.875	0.878	-0.172	0.642	0.687	-0.791
F	0.455	0.522	-1.985	0.270	0.463	-5.102 **
G	0.704	0.722	-1.235	0.597	0.605	-0.367

Table 4 F1 score results for classifying bug reports and feature requests in RQ3 both using and not using metadata based features. Student's independent t-test scores are also given with statistically significant differences shaded and highlighted with asterisks (* $p<0.05$, ** $p<0.01$, *** $p<0.001$).

Due to the fact that a majority of datasets in our evaluation contain app reviews (Datasets A-E), we also trained an "app review only" leave-one-out classifier which is the same as the above described leave-one-out classifier, but is not trained on Dataset F or G. This was done to evaluate the transfer abilities of a classifier between datasets but on feedback from the same feedback channel.

For further context, a zero-shot text user feedback classification model (denoted **"Zero shot"**), as was proposed by Hadi and Fard [16], was also evaluated on each dataset to provide a performance benchmark.

A visual summary of how these three research questions were answered can be found in Table 2.

5 Results

This section first presents the results of training and testing using metadata (RQ1). It then presents the results from training and testing on mixed and separated apps within datasets (RQ2). Finally, it presents the results of training and testing on separate datasets (RQ3).

5.1 Using metadata features to classify (RQ1)

Table 4 details the mean F1 scores of RQ1 classifiers both including and excluding metadata features to classify feedback into bug reports and feature requests.

For bug reports, we find all datasets have higher F1 scores when metadata and text is used to classify compared to when only text is used. However, only one of these differences (dataset B) increases between models were statistically significant with a p-value of <0.05.

Similarly for feature requests, we find that for 5 out of the 6 datasets studied, models which use metadata and text achieve a higher F1 score than just

Dataset	Bug classification			Feature request classification		
	Single dataset separated	Single dataset mixed	T-test stat.	Single dataset separated	Single dataset mixed	T-test stat.
A	0.664	0.764	-2.137	-	-	-
B	0.680	0.725	-3.252 *	0.399	0.468	-2.087
C	0.339	0.357	-0.325	0.501	0.544	-0.639
D	0.868	0.856	0.716	0.659	0.653	0.341
E	0.683	0.875	-3.504 **	0.443	0.642	-2.580 *
G	0.688	0.704	-0.837	0.528	0.597	-3.227 *

Table 5 F1 score results for classifying bug reports and feature requests in RQ2 using both "separated" and "mixed" data splits. Student's independent t-test scores are also given with statistically significant differences shaded and highlighted with asterisks (* $p<0.05$, ** $p<0.01$, *** $p<0.001$).

using text. Again, only one (datasets F) of these differences were statistically significant to a p-value of 0.05.

Overall, we can see either a slight increase or no change in the performance of classifiers when metadata and text are used together compared to when text alone.

Answer to RQ1 - How does training and testing with *metadata* affect classification F1 score? Training on metadata results does not result in a statistically significant increase in classification F1 score on the majority of datasets tested.

5.2 Mixed vs. separated apps within data splits (RQ2)

Following from our results in RQ1, we do not use metadata in our classification experiments for RQ2 and RQ3. Table 5 details the mean F1 scores of classifiers from RQ2 for classifying bug reports and feature requests.

For the models which were trained on only one dataset, it can be seen that app-separated splits have a lower F1 score than mixed-app splits for 5 out of the 6 bug report datasets and 4 out of the 5 feature request datasets. For bug reports, two of these differences (B and E) were statistically significant to $p<0.05$, while for feature requests, two (E and G) were significant. Moreover, we see that only one of these differences is significant to $p<0.01$ over these 11 differences.

Answer to RQ2 - How does training and testing on feedback from separate apps affect classification F1 score? We find that in most cases, there is a small increase in classification F1 score when evaluating classifiers on feedback on the same apps that are contained in their training data. However, we find that only a minority (4/11) of datasets exhibited a statistically significant difference between being split along app lines compared to being split randomly. We therefore find that there is not necessarily always a jump in performance when evaluating on feedback from the same apps as was available

	A	B	C	D	E	F	G
Train A		0.413	0.315	0.430	0.595	0.140	0.301
Train B	0.628		0.311	0.716	0.690	0.267	0.425
Train C	0.540	0.551		0.585	0.668	0.173	0.408
Train D	0.330	0.465	0.278		0.541	0.158	0.391
Train E	0.467	0.364	0.185	0.399		0.203	0.439
Train F	0.191	0.177	0.108	0.282	0.161		0.212
Train G	0.645	0.581	0.324	0.624	0.817	0.317	
Train LOO	0.694	0.704	0.422	0.781	0.792	0.344	0.493
Zero shot	0.653	0.645	0.360	0.712	0.743	0.394	0.692
Single dataset - mixed	0.764	0.725	0.357	0.856	0.875	0.455	0.704

Table 6 F1 scores for classifying bug reports in RQ2. Details the performance of being trained on one dataset and tested on another, as well as the performance of the "leave-one-out" (LOO) model, the zero-shot model, and the model trained on the same dataset as it is tested on. Note that the "Single dataset - mixed" model has been trained on in-domain data (i.e. the training set associated with the test dataset).

	B	C	D	E	F	G
Train B		0.147	0.450	0.439	0.156	0.213
Train C	0.138		0.159	0.148	0.056	0.237
Train D	0.314	0.154		0.211	0.119	0.176
Train E	0.000	0.000	0.000		0.000	0.000
Train F	0.055	0.007	0.028	0.036		0.021
Train G	0.314	0.213	0.356	0.356	0.127	
Train LOO	0.406	0.281	0.527	0.445	0.148	0.274
Zero shot	0.385	0.296	0.365	0.479	0.153	0.522
Single dataset - mixed	0.468	0.544	0.653	0.642	0.270	0.597

Table 7 F1 scores for classifying feature requests in RQ2. Details the performance of being trained on one dataset and tested on another, as well as the performance of the "leave-one-out" (LOO) model, the zero-shot model, and the model trained on the same dataset as it is tested on. Note that the "Single dataset - mixed" model has been trained on in-domain data (i.e. the training set associated with the test dataset).

in training, and that in a majority of cases tested, such a jump is statistically insignificant.

5.3 Testing on separate datasets (RQ3)

Table 6 and Table 7 details the mean F1 scores of separate-dataset classifiers from RQ3 for classifying bug reports and feature requests, respectively.

As can be seen for both bug report and feature request classification, F1 score for any one given test dataset can vary wildly depending on the dataset of the training data. For bug report classifiers, we observe that the classifier trained only on dataset G training data (tweet user feedback) performs best on 4 of the 6 test datasets it is applied to. For feature request classifiers, the classifier trained only on dataset B (app review data) performs best on 4 of the 5 datasets it is applied to. The classifier trained in dataset F (forum data) performs worst on all test datasets it is applied to compared to other classifiers for bug reports. The classifier trained in dataset E (app reviews) performs

Label type	Classifier	A	B	C	D	E
Bug reports	**App LOO**	0.633	0.654	0.418	0.744	0.771
	LOO	**0.694**	**0.704**	**0.422**	**0.781**	**0.792**
Feature requests	**App LOO**	NA	0.378	0.206	0.495	0.422
	LOO	NA	**0.406**	**0.281**	**0.527**	**0.445**

Table 8 F1 scores of leave-one-out (LOO) and app review only leave-one-out (App LOO) classifiers across the five app review evaluation datasets

worst on all test datasets for feature requests. Overall, every classifier trained on one dataset and evaluated on a separate dataset achieves lower classification performance compared to models trained and tested on the same dataset.

In comparison to the models trained on only one (different) dataset, the leave-one-out classifier performs best on 6 of the 7 bug report datasets and 5 out of 6 feature report datasets that it is applied to. Compared to the model trained and tested on the same dataset ("Single dataset - mixed"), the leave-one-out classifier performs slightly worse on all but one dataset. An independent t-test between the leave-one-out and the "Single dataset - mixed" bug report classifier performances results in p-values of 0.122, 0.139, 0.194, 0.000, 0.027, 0.011, and 0.001 for Datasets A-G. The p-values for feature request classification performance are 0.225, 0.001, 0.013, 0.011, 0.002, 0.000 for datasets B-G. Therefore, we find that in a majority of cases, the difference in classification performance between the leave-one-out classifier and the in-domain trained model is statistically significant to $p<0.05$.

Table 8 shows the compared F1 scores of both the leave-one-out classifier and the app review only leave-one-out classifier. We can see that the app only classifier performs similarly, but slightly worse, than the full leave-one-out classifier across all datasets.

The zero-shot classifier performs better than any of the single-dataset out-of-dataset for 5 out of the 7 bug report datasets and 4 out of the 6 feature request datasets. The zero shot model exceeds the performance of the leave-one-out models for the test sets of Dataset F and G (forum posts and tweets) for bug reports, and 4 out of 6 datasets (C, E, F, and G) for feature requests. Therefore, zero shot models perform best relative to other models on datasets of distinct feedback channels. An independent t-test between the zero shot classifier and the "Single dataset - mixed" classifier performances results in p-values of 0.001, 0.000, 0.952, 0.000, 0.000, 0.098, and 0.141 for Datasets A-G, respectively. The p-values for feature request classification performance are 0.053, 0.003, 0.000, 0.020, 0.000, 0.003 for datasets B-G. As with the leave-one-out classifier, we find that in a majority of cases, the difference in classification performance between the zero shot classifier and the in-domain trained model is statistically significant to $p<0.05$.

Table 9 shows the recall and precision alongside the F1 scores for the zero shot classification of both bug reports and feature requests. While we have chosen 0.5 as our probability threshold between true and false labels in the zero shot classification evaluation, we find that recall is much higher than precision for feature requests in all datasets, indicating that a higher threshold

Label type	Metric	A	B	C	D	E	F	G
Feature request	**Recall**	NA	0.890	0.522	0.779	0.731	0.739	0.870
	Precision	NA	0.246	0.208	0.239	0.359	0.085	0.373
	F1	NA	0.385	0.296	0.365	0.479	0.153	0.522
Bug report	**Recall**	0.518	0.660	0.391	0.681	0.634	0.447	0.712
	Precision	0.884	0.630	0.336	0.748	0.898	0.354	0.673
	F1	0.653	0.645	0.360	0.712	0.743	0.394	0.692

Table 9 Precision, recall and F1 score of zero shot classification for both bug reports and feature requests

value may result in a higher F1 score. However, we do not find this trend to be as clear in bug report classification, with only four out of seven datasets' recall exceeding the precision.

Answer to RQ3 - How does training and testing on feedback from separate datasets affect classification F1 scores Training and testing a user feedback classifier on feedback from separate datasets results in overall lower performance than training and testing on the same dataset. However, this lower performance can be improved upon by models trained on multiple datasets or by zero-shot text classification models.

6 Discussion

This section discusses the results of this work and their implications. Firstly, the effect of training and testing on separate apps is discussed. Then the effect of training and testing on separate datasets is described. Finally, the effect of using metadata in user feedback classifiers is analysed.

6.1 RQ1 - Classifying with metadata

Our findings for RQ1 are that classification performance is modestly, but not significantly, improved when using metadata. This finding is in contrast to previous findings which reported the use of metadata on feedback classification, in which metadata was shown to have a positive impact on classification performance [23,36]. We theorise that this may be due to the fact that the state-of-the-art classifiers that we used contain millions of parameters [9], compared to very few parameters available in the classical machine learning models used in these earlier works. With this increased capacity, our model may be better able to infer metadata from the text itself (for example low review ratings would also be associated with more negative sentiment in the text), which means that having this information explicitly provided would not have much of an effect on the final prediction. We therefore recommend that the use of metadata as a feature should be reviewed within text-based software engineering machine learning tasks with the advent of new, very capacious language models such as BERT (Bidirectional Encoder Representations from Transformers). Without the use of what appears to be largely superfluous

metadata, these models are better able to be applied to different feedback and to new feedback channels, where metadata may differ.

We have shown that metadata does not affect performance significantly across a majority of datasets in the classification of bug reports and feature requests, but it is an open question as to how metadata would affect other classes of feedback. It is for future work to investigate a fuller picture as to which classes benefit most and least from use of metadata in their prediction.

6.2 RQ2 - Mixed vs. separated apps within data splits

Our results show an increase in F1 score when using the mixed configuration (random sampling across data splits allowing the same app to have feedback in both training and testing sets) over the separated configuration (different apps in the training and testing sets) in 9 of the 11 cases evaluated. We theorise that we obtain these results because training on feedback from one app allows the classifier to better model that app's feedback, making it better able to predict the class of similar feedback from that app. However, we only find a minority of these differences are statistically significant, meaning that we do not conclude that training and testing on the same app's feedback has a meaningful effect on classification performance in aggregate. We observe that the three datasets which do exhibit statistical differences (Datasets B, E, and G) all have a smaller number of apps per dataset compared to the datasets with no statistical differences (Datasets A, C, and D). We postulate that this could be due to the fact that a greater number of apps within the training data allows for a classifier to better generalise to new apps, and to not overfit to the apps it has been trained on. However, we do not find this trend to be robust, with only one dataset (Dataset E) displaying a statistically significant difference across both classification tasks studied (classifying bug reports and feature requests). Therefore, it is for future work to evaluate the effect of the number of apps included in the training data on classifier generalisability to new apps.

Overall, we found little difference between evaluating a classification model on feedback from unseen apps compared to evaluating on feedback from the same apps that it was trained on. This finding is made across both models classifying bug reports and feature requests. This result suggests that model evaluation as is currently carried out within the literature (i.e. not specifying that train, validation, and test splits must contain feedback from separate apps) can be seen to be a good predictor of performance of a classifier on unseen apps from within a dataset. This hints at potential real-world applicability of these models in that they could be used on feedback from unseen new apps (but crucially from the same channel and data-gathering process) without a significant expected drop in performance.

Another outcome of these experiments is that the classification F1 score can range from high (greater than 0.8) to low (less than 0.5). When a classifier has low absolute classification performance, their utility in finding requirements

is limited. This finding highlights the fact that automatic classification using current technology is not universally useful across all feedback datasets. The fact that many of these values are slightly lower than their literature quoted values could possibly be due to the fact that we decided against doing extensive hyperparameter tuning when training our classifiers. Reasoning and discussion of this is given in Section 7. Finally, the lower classification performance across most of the datasets for classifying feature requests compared to bug reports is a trend that can be broadly seen throughout the literature, and calls into question exactly why a bug report is so much easier to identify (from a machine learning perspective) compared to a feature request.

6.3 RQ3 - Testing on unseen datasets

In RQ3, we found that a model trained on one dataset and then applied to another dataset achieves worse performance than a model trained and tested on the same dataset. This is not a surprising finding, given that class balance and labelling methods vary slightly between datasets. However, it raises an important question: How informative are the predictions of these models when used in the real world? A dataset of user feedback for a given software project is not guaranteed to have a certain class balance, and a given researcher or developer is not guaranteed to consider a piece of feedback to contain a bug or feature request in the same way that the training data labellers did.

Our results with the leave-one-out models show better performance, in contrast. While the leave-one-out models perform worse than models trained and tested on the same data, they perform better and are more consistent compared to models trained on one dataset. The leave-one-out models also perform better compared to the zero shot classifier except for feedback from an unseen channel (tweets and forum posts) at training time. This indicates that while leave-one-out classifiers are useful, zero shot classifiers are more appropriate for classifying feedback from unseen feedback channels. Of particular note is the fact that in some cases, either the leave-one-out classifier or zero-shot classifier performs better than or not statistically significantly worse than the in-domain classifier for both bug report and feature request classification. This hints at the fact that in some cases, there may actually be no need to do in-domain training, and that zero-shot or out-of-domain pre-trained classifiers may suffice for classifying user feedback. Future work could assess exactly in which situations this would be most effective, and how this could be implemented.

It is also an open question as to how an ensemble of the leave-one-out and zero shot classifiers would perform in our evaluation. Such a classifier could potentially combine the strengths of both classifiers to be robust to feedback from unseen feedback channels while maintaining performance on app reviews. It remains for future work to determine the effectiveness of such an approach.

We also find that the zero-shot classification probability threshold may be too low for feature request classification. In this work, we decided to set our

probability threshold to 0.5 as a true "zero shot" classification configuration. We assume no prior knowledge of which threshold would be most appropriate for a given dataset, but we observe that for feature request classification, this threshold may be too low across all datasets, and thus may be generalisable to feature request classification more generally. It is for future work to explore which probability threshold is most effective at correctly classifying different types of user feedback.

Another result of this work is that the leave-one-out classifier trained only on app review data performed worse than the leave-one-out classifier trained on all other available datasets. This highlights the importance of more and potentially more varied data for generating classifiers that generalise across datasets.

These results suggest that user feedback analysis tools will achieve highest performance if before use they first require a sample of labelled user feedback from the developer who intends to use the tool (i.e. use in-domain training data). This could be done through an active learning approach, such as was explored by Magalhães et al. [24], in order to limit the amount of labelled data required. However, a tool that requires no further labelled data before being used can still achieve good performance if it is either: trained using a labelled feedback dataset from the same feedback channel; or a zero-shot text classifier if no such dataset exists.

With these results, we recommend that future creators of user feedback analysis tools train a classification model using as much labelled user feedback as possible, especially using data from the same channel as its intended use-case. If such a dataset does not exist and is prohibitively expensive to create, then we recommend using zero-shot classification models instead.

In order to aid future user feedback analysis tools, we make bug report and feature request classifiers available for use on the Huggingface channel. We aim to make these available with a link upon publication.

6.4 Implications for RE practitioners

These results help the developers of RE tools to better understand exactly when, where, and how they can use their automated classifiers for engineering better requirements, and ultimately enable RE practitioners to develop better software.

With this work, RE tool developers know that including metadata in a user feedback classifier is not essential for improved classification performance.

These results also show that practitioners can often use out-of-the-box tools (i.e. classifiers that do not need the user to provide their own training data before they can be used) without a massive loss in classification accuracy compared to a classifier trained on data the requirements engineer provided. This could potentially increase the agility of these classifiers, as they would not require a development cycle to prepare and train them, and thus could be used on feedback quickly.

Since these classifiers are not trained on their user's data, they will be less prone to overfitting that data. Thus, if the nature of the user feedback for a given product suddenly changes (i.e. the data distribution diverges to that of the training data), then these classifiers may be more robust to this change compared to traditionally trained classifiers. An ability to adapt to new situations would certainly make these tools more useful to requirements engineers, because times where feedback suddenly changes (e.g. a development team suddenly get swamped with angry Tweets because of a buggy update) is when these classifiers are most useful in the context of RE.

7 Threats to validity

One threat to the validity of this work is that the results of this study many not generalise to the classification of other feedback classes. This study only examined the performance of classification models on classifying feedback into the binary labels of "Bug report" or "No bug report" and "Feature request" or "No feature request". These two labels were chosen due to the fact that they were the only two consistent labels across multiple datasets. Being able to automatically detect bug reports and feature requests from users is one of the key promises of utilizing online user feedback for requirements engineering [18]. Furthermore, the abundance of these labels in various literature datasets highlights how useful these labels are considered to be. Therefore, focusing on the task of classifying bug reports and feature requests can still be seen to be valuable to those looking to engineer requirements using user feedback. It is for future work to replicate this research on other label types.

When using different datasets in our experiments for RQ3, we considered each individual dataset to be independent of each other dataset, and thus "unseen" to the classifier at prediction time. However, some datasets do share apps within their dataset (i.e. two datasets can contain feedback from the same app). The dataset which contains the most shared apps is Dataset B, where three of the seven apps (Picsart, Dropbox, and WhatsApp) are shared with other datasets: Dataset C (Picsart and WhatsApp), Dataset D (Picsart and Dropbox), and Dataset G (WhatsApp). These apps make up a minority of the feedback ($1857/4385 \approx 42.3\%$) contained within Dataset B. When considering the impact of this overlap for training the leave-one-out classifiers, the feedback from these three apps make up a small minority ($\approx 2.8\%$) of feedback within the combined datasets A, C, D, E, F and G, which was used for training. Since this is a minority of the classifier training dataset, coupled with the fact that we do not find compelling evidence that training and testing on feedback from the same app within datasets is useful across a majority of cases tested in RQ2, means that we do not expect this to have a large effect on the validity of our findings in RQ3. We also observe the leave-one-out classifier exhibiting comparatively the same classification accuracy differential compared to the single dataset, zero shot and in-domain classifiers that are observed in other datasets, indicating little classification improvement is made

by partially training on feedback from the same apps. However, we do consider that this could be a factor in the relatively good performance of the leave-one-out classifiers, and we leave it for future work to investigate the effect of shared apps on classification performance of inter-dataset classifiers more thoroughly.

Beyond the results in RQ2, we also consider the impact of comparing the results of classifiers trained on datasets with a varied number of both apps and feedback. While we do not observe a meaningful difference between results in any of the research questions related to dataset size, we do still consider that dataset size could have an effect on the generalisability of RE classifiers. Since all of our datasets have between hundreds and thousands of pieces of feedback, we cannot know if a dataset with several orders of magnitude more labelled feedback may perform better at classifying requirements on unseen datasets. While datasets of this size are not currently publicly available within the RE literature, it remains possible that a very large corpus of noisily labelled data could be created to train a classifier, at which point the generalisability of said classifier could be fundamentally different to those examined in our work. Therefore, while the experiments in this work do not find any effect of dataset size on the transfer performance of RE classifiers, it remains for future work to further investigate this further, particularly with respect to very large datasets.

As stated in section 4.1, we were not able to guarantee the 64:16:20 training : validation : testing data split ratio for the separated configuration compared to the mixed configuration due to the constraint of not having feedback from the same app in both splits of feedback. This could threaten the validity of our results in RQ2 due to the fact that the separated configuration data splits could be highly skewed compared to the mixed configuration. We mitigate this hazard in two ways - firstly, by removing datasets with too few apps to be able to effectively split into the 64:16:20 split (notably Dataset F, which has only 2 apps), and secondly by performing five fold cross validation so that potential discrepancies in dataset split size are averaged out over five runs.

Another potential threat to the validity of this work that we did not carry out any data balancing when creating our classifiers. Multiple studies within the literature, including those associated with datasets used in this work [23, 36], carried out data balancing before training their classifier. This is done to counteract the fact that user feedback may have classes of interest which are a small minority of overall feedback, and so a model is unable to learn the characteristics of this class if most of its training data is from other classes. However, studies on datasets outside of the domain of user feedback classification have shown that classifiers can perform well even when trained on highly unbalanced data [3]. Moreover, Henao et al. demonstrated that undersampling when training a deep language model has no major impact on the F1 score of the classifier [17]. It is for this reason that we decided against balancing our data, and it is for future work to fully explore the impact of data balancing on user feedback classification.

A final threat to validity considered was the lack of hyperparameter tuning done for any one model, which may have led to lower absolute classification performance. While optimising the hyperparameters for any one app or dataset

may have led to marginal performance gains, we found that in early experiments changing hyperparameters has little impact on overall classification performance. Our research questions also focused on the relative differences between machine learning treatments, rather than absolute values, and so we would expect that any performance improvements that would be introduced by hyperparameter tuning would not affect our overall conclusions. Furthermore, one of the aims of this work was to investigate how well models apply to unseen data domains. Tuning hyperparameters for the model's training domain may overfit it and disadvantage it when applied to out-of-domain data. It is for this reason we decided against extensive hyperparameter tuning.

8 Conclusion

The technical quality of software is meaningless if it does not meet the needs of its intended users. Requirements engineering (RE) offers a way to gather the requirements of users, and has been shown to improve software quality generally. This work builds on the RE literature in understanding and automatically processing online user feedback for use in developing and maintaining software. Previous work has shown that it is possible to create text classifiers that can automatically detect bug reports, feature requests, and other requirements relevant information in user feedback for use in the software development cycle. This work contextualises these past results, and informs the future improvement of these classifiers. This has led to three broad contributions.

Firstly, we demonstrate that classification of both bug reports and feature requests do not notably benefit from metadata (app ratings, forum post position, etc.) as features.

Secondly, we showed that there can be a small drop in classification performance when applying trained classifiers to feedback from unseen apps for some datasets. However, this trend was found to be statistically insignificant across the majority of datasets tested.

Finally, this paper demonstrated the classification performance of models which had not been trained on the dataset of given test set. We found that in the scenario where no data from a specific dataset is used to train a classifier, training a model on multiple other datasets achieves better performance than training on any one dataset alone. Moreover, we found that these multiple-dataset models are most applicable to datasets in which it contains feedback from channels which the model has been trained on (app reviews). We found that for other channels (tweets and forums), which did not have another dataset to represent it in the training data, zero-shot classification models performed better.

Overall, these three results can inform the creation of better user feedback analysis tools so that, ultimately, developers will better understand the needs of their users and create higher quality software.

We have made the replication package for this study available online[8].

[8] https://doi.org/10.5281/zenodo.5733504

9 Declarations

9.1 Funding and/or Conflicts of interests/Competing interests

Competing Interests:
One of the authors of this paper (Kelly Blincoe) is the editorial boards of the IEEE Transactions on Software Engineering, the Empirical Software Engineering Journal, and the Journal of Systems and Software.

The work was conducted in the course of a PhD study by the lead author (Peter Devine), which is funded by the Faculty of Engineering at The University of Auckland.

Consent:
Matters of consent are not applicable to this work due to the fact that no human participants were involved.

References

1. Ali Khan, J., Liu, L., Wen, L., Ali, R.: Conceptualising, extracting and analysing requirements arguments in users' forums: The crowdre-arg framework. Journal of Software: Evolution and Process **32**(12), e2309 (2020)
2. Araujo, A., Golo, M., Viana, B., Sanches, F., Romero, R., Marcacini, R.: From bag-of-words to pre-trained neural language models: Improving automatic classification of app reviews for requirements engineering. In: Anais do XVII Encontro Nacional de Inteligência Artificial e Computacional, pp. 378–389. SBC (2020)
3. Batista, G.E., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD explorations newsletter **6**(1), 20–29 (2004)
4. Berki, E., Georgiadou, E., Holcombe, M.: Requirements engineering and process modelling in software quality management—towards a generic process metamodel. Software Quality Journal **12**(3), 265–283 (2004)
5. Broy, M.: Requirements engineering as a key to holistic software quality. In: International Symposium on Computer and Information Sciences, pp. 24–34. Springer (2006)
6. Ciurumelea, A., Schaufelbühl, A., Panichella, S., Gall, H.C.: Analyzing reviews and code of mobile apps for better release planning. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 91–102. IEEE (2017)
7. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: Automated classification of non-functional requirements. Requirements engineering **12**(2), 103–120 (2007)
8. Damian, D., Chisan, J.: An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Transactions on Software Engineering **32**(7), 433–453 (2006)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: 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, Volume 1 (Long and Short Papers), pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (2019). DOI 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423
10. Dhinakaran, V.T., Pulle, R., Ajmeri, N., Murukannaiah, P.K.: App review analysis via active learning: reducing supervision effort without compromising classification accuracy. In: 2018 IEEE 26th International Requirements Engineering Conference (RE), pp. 170–181. IEEE (2018)

11. Di Sorbo, A., Grano, G., Aaron Visaggio, C., Panichella, S.: Investigating the criticality of user-reported issues through their relations with app rating. Journal of Software: Evolution and Process **33**(3), e2316 (2021)
12. Gillies, A.: Software quality: theory and management. Lulu. com (2011)
13. Guzman, E., Alkadhi, R., Seyff, N.: A needle in a haystack: What do twitter users say about software? In: 2016 IEEE 24th International Requirements Engineering Conference (RE), pp. 96–105. IEEE (2016)
14. Guzman, E., El-Haliby, M., Bruegge, B.: Ensemble methods for app review classification: An approach for software evolution (n). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 771–776. IEEE (2015)
15. Guzman, E., Ibrahim, M., Glinz, M.: A little bird told me: Mining tweets for requirements and software evolution. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp. 11–20. IEEE (2017)
16. Hadi, M.A., Fard, F.H.: Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews. arXiv preprint arXiv:2104.05861 (2021)
17. Henao, P.R., Fischbach, J., Spies, D., Frattini, J., Vogelsang, A.: Transfer learning for mining feature requests and bug reports from tweets and app store reviews. In: 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW), pp. 80–86. IEEE (2021)
18. Iacob, C., Harrison, R., Faily, S.: Online reviews as first class artifacts in mobile app development. In: International Conference on Mobile Computing, Applications, and Services, pp. 47–53. Springer (2013)
19. Iqbal, T., Khan, M., Taveter, K., Seyff, N.: Mining reddit as a new source for software requirements. In: 2021 IEEE 29th International Requirements Engineering Conference (RE), pp. 128–138. IEEE (2021)
20. Kassab, M., Neill, C., Laplante, P.: State of practice in requirements engineering: contemporary data. Innovations in Systems and Software Engineering **10**(4), 235–241 (2014)
21. Lim, S., Henriksson, A., Zdravkovic, J.: Data-driven requirements elicitation: A systematic literature review. SN Computer Science **2**(1), 1–35 (2021)
22. Lin, D., Bezemer, C.P., Zou, Y., Hassan, A.E.: An empirical study of game reviews on the steam platform. Empirical Software Engineering **24**(1), 170–207 (2019)
23. Maalej, W., Kurtanović, Z., Nabil, H., Stanik, C.: On the automatic classification of app reviews. Requirements Engineering **21**(3), 311–331 (2016)
24. Magalhães, C., Sardinha, A., Araújo, J.: Mare: an active learning approach for requirements classification. In: RE@Next! track of the 29th IEEE International Requirements Engineering Conference (2021)
25. Nayebi, M., Cho, H., Ruhe, G.: App store mining is not enough for app improvement. Empirical Software Engineering **23**(5), 2764–2794 (2018)
26. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, pp. 35–46 (2000)
27. Pagano, D., Maalej, W.: User feedback in the appstore: An empirical study. In: 2013 21st IEEE international requirements engineering conference (RE), pp. 125–134. IEEE (2013)
28. Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C.A., Canfora, G., Gall, H.C.: Ardoc: App reviews development oriented classifier. In: Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering, pp. 1023–1027 (2016)
29. Radliński, Ł.: Empirical analysis of the impact of requirements engineering on software quality. In: International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 232–238. Springer (2012)
30. Rempel, P., Mäder, P.: Preventing defects: The impact of requirements traceability completeness on software quality. IEEE Transactions on Software Engineering **43**(8), 777–797 (2016)
31. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108 (2019)

32. Scalabrino, S., Bavota, G., Russo, B., Di Penta, M., Oliveto, R.: Listening to the crowd for the release planning of mobile apps. IEEE Transactions on Software Engineering **45**(1), 68–86 (2017)
33. Stanik, C., Haering, M., Maalej, W.: Classifying multilingual user feedback using traditional machine learning and deep learning. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), pp. 220–226. IEEE (2019)
34. Sultan, M.A., Bethard, S., Sumner, T.: Back to basics for monolingual alignment: Exploiting word similarity and contextual evidence. Transactions of the Association for Computational Linguistics **2**, 219–230 (2014)
35. Tizard, J., Rietz, T., Liu, X., Blincoe, K.: Voice of the users: an extended study of software feedback engagement. Requirements Engineering pp. 1–23 (2021)
36. Tizard, J., Wang, H., Yohannes, L., Blincoe, K.: Can a conversation paint a picture? mining requirements in software forums. In: 2019 IEEE 27th International Requirements Engineering Conference (RE), pp. 17–27. IEEE (2019)
37. Williams, G., Mahmoud, A.: Mining twitter feeds for software user requirements. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp. 1–10. IEEE (2017)
38. Yin, W., Hay, J., Roth, D.: Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. arXiv preprint arXiv:1909.00161 (2019)