App Review Analysis via Active Learning

Reducing Supervision Effort without Compromising Classification Accuracy

Venkatesh T. Dhinakaran†, Raseshwari Pulle†, Nirav Ajmeri‡, Pradeep K. Murukannaiah†
†*Department of Software Engineering, Rochester Institute of Technology*
Rochester, NY 14623, USA
‡*Department of Computer Science, North Carolina State University*
Raleigh, NC 27695, USA
vt7300@rit.edu, rp3737@rit.edu, najmeri@ncsu.edu, pkmvse@rit.edu

***Abstract*—Automated app review analysis is an important avenue for extracting a variety of requirements-related information. Typically, a first step toward performing such analysis is preparing a training dataset, where developers (experts) identify a set of reviews and, manually, annotate them according to a given task. Having sufficiently large training data is important for both achieving a high prediction accuracy and avoiding overfitting. Given millions of reviews, preparing a training set is laborious.**

We propose to incorporate *active learning*, a machine learning paradigm, in order to reduce the human effort involved in app review analysis. Our app review classification framework exploits three active learning strategies based on uncertainty sampling. We apply these strategies to an existing dataset of 4,400 app reviews for classifying app reviews as features, bugs, rating, and user experience. We find that active learning, compared to a training dataset chosen randomly, yields a significantly higher prediction accuracy under multiple scenarios.

***Keywords*-App review analysis; active learning; mobile apps; Crowd RE; social requirements;**

I. Introduction

Smart devices including smartphones, tablets, and a variety of wearables are ubiquitous; so is developing software applications or *apps* for these smart devices. According to the latest estimates, there are over 2.5 billion smartphone users, and over 12 million app developers who develop apps for these devices [9]. These developers have made available over five million smart device apps on digital distribution stores such as Google Play, Apple App Store and Amazon Appstore, garnering over 200 billion application downloads [39].

The app users describe their interaction and experience with an app as *app reviews* on application distribution stores. App reviews are a rich source of information for app developers. Specifically, app reviews contain a wealth of information related to requirements, e.g., bug reports [24], change requests [8], privacy requirements [40], nonfunctional requirements [1], [21], and even the features within an app that users like [11] and the rationales for their likes and dislikes [20]. It is crucial for developers to recognize and act on such information in a timely manner [28]. Otherwise, given the amount of choices, users could simply move to an alternative app [43].

According to a recent empirical study [33], apps on digital distribution platforms receive on an average 22 reviews every day and, depending on an app's popularity, this number could go as high as a few thousands per day. Of these reviews, only about a third are helpful to the analysts and developers [6]. With the sheer amount of the review data that is being generated every day, it is extremely labor-intensive for developers to manually vet all reviews to identify the ones that are useful for their purposes. Accordingly, it is not surprising that app developers, typically, do not respond to app reviews although responding may have positive effects such as users increasing ratings as a result of a developer response [27].

The need for automated approaches to app review analysis has been well recognized in the literature. Broadly, there are two categories of automated app review analysis techniques: (1) *supervised* approaches such as *classification*, e.g., [4], [8], [24]; and (2) *unsupervised* approaches such as *clustering* and *topic modeling*, e.g., [6], [14]. A key difference between these two types of approaches is that supervised approaches need "labelled" data instances for training whereas unsupervised approaches can be trained with unlabelled data instances.

Labeling data requires human effort (or supervision). Hence, one may argue in favor of using an unsupervised technique, such as the one proposed by Villarroel et al. [43], instead of using a supervised approach such as the one proposed by Maalej et al. [24]. However, unsupervised techniques are not effective when the number of classes are unknown [32] and may need a very large dataset to be effective. Further, unsupervised approaches may perform well for simple tasks such as categorizing reviews as informative versus not informative reviews [6], but may fail to make finer distinctions such as feature versus bug, or privacy versus security requirements. In contrast, a classifier (supervised) learns from user-provided labels and predicts classes for unlabeled instances. However, to obtain a satisfactory accuracy, it may still require a large number of labels, incurring significant human effort.

The review labelling task, typically performed by developers or subject-matter experts (SMEs), is extremely labor-intensive considering the amount of noise in reviews, the informal language often used, and the ambiguities inherent to natural language. Further, it is necessary to obtain labels from multiple developers for each review to create a reliable training dataset. Requiring significant manual effort, especially from experts, weakens the argument for automated app review analysis.

CrowdRE [10] is a promising avenue for engaging the *crowd* (general public) in human-intensive RE tasks such as classifying reviews and extracting requirements [3]. CrowdRE

could substantially reduce labeling workload on developers, but it may not be effective for all review labeling tasks. For example, the crowd's lack of domain knowledge limits their ability to distinguish bug reports from feature requests; such miscategorization severely impacts the accuracy of the resulting models [15]. Further, it would be desirable to engage the crowd (and human intelligence, in general), for creative [30] as opposed to mundane tasks such as labeling.

Contributions: We seek to effectively utilize the available human resources in the process of automated app review analysis. This is an important challenge considering that app review analysis may incur significant time, effort, and money. To this end, we summarize our contribution as follows.

Goal: Reducing the human effort required for app review analysis without compromising the accuracy of analysis.

Method: We propose to exploit active learning, a well-known machine learning paradigm [37], [38], for app review classification. We describe a generic active learning framework (pipeline) that seeks to minimize human effort required for training a review classifier by intelligently selecting unlabelled reviews for labelling via uncertainty sampling.

Evaluation: We conduct extensive experiments on an existing dataset [24] consisting of 4,400 labelled reviews (consisting of four classes), comparing active learning and baseline classifiers, considering (1) both binary and multiclass problems; (2) three uncertainty sampling strategies; (3) different training set sizes and classification techniques.

Organization: Section II details our active learning approach to classify app reviews. Section III describes our experiments. Section IV discusses results. Section V describes the related works and Section VI concludes the paper.

II. Method

In this section, we motivate the need for active learning for app review analysis via an example scenario, demonstrating (intuitively) that choosing the right training set can potentially enhance the review classification accuracy. Then, we describe the three active learning strategies our approach explores.

A. Motivating Example

Consider a cloud storage app, CloudDrive which helps users store and share files on the cloud (similar to Dropbox and Google Drive). Suppose that the developers of the app want to analyze its reviews to identify the features requested by the app's users. Since CloudDrive has more than a million reviews, the developers would like to employ an automated classifier to identify reviews corresponding to feature requests.

The first step toward realizing an app review classifier is to build a training set consisting reviews labelled as a feature request (✓) or as a non-feature request (✗). The CloudDrive's developers intend to crowdsource these labels. However, being a startup, CloudDrive has a limited budget for acquiring the labels. To simplify the example, imagine that CloudDrive's target is to acquire labels for five reviews (in practice, one would acquire labels for a larger number of reviews).

TABLE I
Sample reviews of the CloudDrive app. The shaded reviews are labelled as feature request or not, and the unshaded reviews are unlabelled

ID	Review	Feature
r_1	Very convenient app where I can store all of my photos+ videos with easy access	✗
r_2	Plz **add** scan with jpg format	✓
r_3	The app is ok but their plans are too expensive	✗
r_4	Please **introduce** something like Dropbox lite! 55Mb app size is way too much!	✓
r_5	It does ***not support*** multi-selection that I need to download one by one on web version	?
r_6	Please **add** dark themes	?
r_7	***No support*** for OCR on PDFs? I'm looking to move...	?
r_8	**Introducing** an option to automatically save photos from Micro SD card will be great!	?
...	...	...

As shown in Table I, suppose that the developers have somehow acquired labels for four reviews (r_1–r_4) so far. The question, then, is which review to acquire the fifth label for. Suppose that there are four candidates (r_5–r_8) to choose the fifth review from (in practice, the number of candidates is much larger—every unlabelled review is a potential candidate).

A simple approach is to choose the next review to label randomly from the candidate pool. Suppose that the randomly chosen review is r_6. Next, the developers train a review classifier on the training set (consisting of five labelled reviews) and predict whether each of the remaining three reviews (r_5, r_7, and r_8) is a feature request or not. Assume that each of these remaining three reviews is a feature request (ground truth). Further, given this training set, consider that the classifier somehow learns that reviews consisting of tokens "add" and "introduce" are feature requests and others are not. Then, the classifier would predict r_8 as a feature request, and r_5 and r_7 as not, yielding a very low (33%) prediction accuracy.

Instead of choosing r_6 as the fifth training instance, what if we somehow chose r_5? From this training set, the classifier may have learned that the tokens "add," "introduce," and "not support" are indicative of a review being a feature request. This classifier would, then, predict each of the remaining reviews (r_6–r_8) as a feature request, yielding 100% accuracy.

Thus, a key question is: how do we decide to choose r_5 instead of r_6 before acquiring the label (and training the classifier with five instances)? The problem with adding r_6 as the fifth training instance is that it provides no new information to the classifier in the training process. That is, the classifier can learn from the original four training instances that the token "add" indicates that a review is potentially a feature request; and, including r_6 (which, too, consists of "add"), does not provide any new information. In contrast, including r_5 provides an opportunity for the classifier to learn that the token "not support" may also be indicative of a review being a feature request. Active learning (Section II-C) exploits this intuition to choose the right training set.

Note that the scenario above is a simplifying example. In practice, (1) one may acquire labels for hundreds, if not

thousands, of reviews; (2) the number of potential candidates to choose the next review to label from may be tens of thousands; (3) there may be more than two classes (feature, bug, and rating); and (4) a classifier's decision boundary may be much more complex than deciding whether an instance belong to a class based on certain keywords. Yet, the idea that a classifier can learn better from some training sets than others also applies to more complex scenario as we describe next.

B. An Active Review Classification Framework

Figure 1 shows an overview of our active (learning) review classification framework. In a nutshell, the framework takes a large pool of unlabelled app reviews as input and outputs categorized reviews. In this process, the framework employs:

An active learner that (iteratively) chooses unlabelled reviews for labelling;

A set of review oracles (human labellers), who assign each review selected by the active learner to one (or more, depending on the classifier) predefined categories; and

A review classifier that, given a set of labelled reviews, automatically learns a model capable to predicting the categories (labels) for new (unlabelled) reviews.

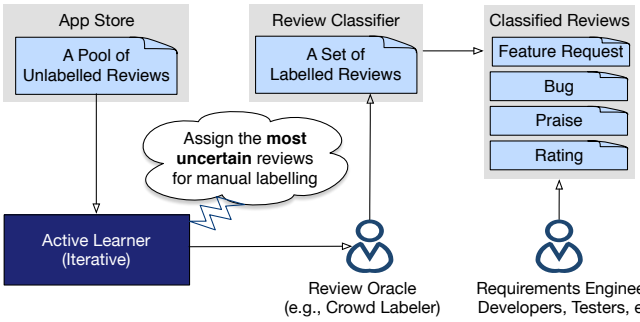

Fig. 1. An active learning pipeline for classifying app reviews

Other works have explored techniques for classifying reviews and for exploiting review oracles (described in Section V). The crux of our work is the active learner.

Figure 2 shows the steps involved in active learning a review classifier from a large pool of unlabelled reviews. We employ one of the most common active learning frameworks known as *uncertainty sampling* [38]. The process of active learning via uncertainty sampling involves the following steps.

Initialization: Randomly sample a small set of unlabelled reviews and acquire labels for those from the review oracles. Note that the initial sample needs to be large enough such that it is possible to train a classifier from it (how good that classifier is does not matter at this stage).

Training: Train a classifier from the training set of labelled reviews. Any classifier can be used in this stage. However, it is required to ascertain an *uncertainty* score for each prediction the classifier makes. Most probabilistic classifiers (such as naive Bayes and logistic regression) meet this requirement, where the probability of an instance belonging to a class is indicative of the uncertainty (e.g., for a binary classification problem, the closer the probability of a prediction is to 0.5, the higher the uncertainty of the classifier's prediction). Similarly, for margin-based classifiers such as Support Vector Machines (SVMs), the uncertainty can be measured as a function of the distance of an instance from the decision boundary (the separating hyperplane for SVM) [42]—the closer an instance is to the boundary, the higher the corresponding uncertainty.

Prediction: For each remaining unlabelled review, predict the class label and ascertain the uncertainty of prediction.

Choose the most uncertain predictions: Order the reviews by their respective uncertainty of prediction, and choose the reviews corresponding to the k most uncertain predictions. The choice of k is a tradeoff between classification accuracy and training efficiency—ideally, k should be low, but choosing a very low value (e.g., $k = 1$) slows the training process without considerably enhancing the classification accuracy.

Repeat: The reviews chosen in the step above are the ones for which the classifier is most unsure of the labels. In essence, acquiring labels for these reviews can potentially be most informative to the classifier. Thus, we acquire the labels for these reviews from the review oracles, add these labelled reviews to the training set in the second step above, and repeat the process. The process stops when (1) the desired classification accuracy is reached, (2) the labelling budget is exhausted, or (3) there are no more unlabelled reviews.

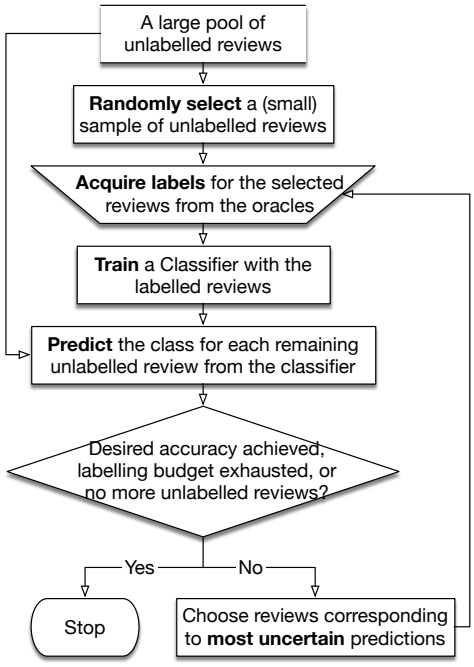

Fig. 2. A flowchart depicting the steps employed by the active learner

C. Uncertainty Sampling Strategies

A key step in the active learning process above is ascertaining the uncertainty of predictions. We experiment (Section III) with well-known strategies [37], [38] for doing so. We briefly describe these strategies below to be self contained.

As shown in Figure 2, each iteration of the active learning process starts with a training set T of labelled reviews, and a set R of unlabelled reviews. Let C be the set of all labels (classes). Suppose that we train a probabilistic classifier from T and let Θ be the parameters of the classifier. Then, for an $r \in R$ and a $c \in C$, $P_\theta(c|r)$ indicates the probability that r belongs to class c as predicted by the classifier. Our objective is to choose one unlabelled review (let $k = 1$), $r^* \in R$, to add to the training set in the next iteration of the process. We describe three uncertainty sampling strategies for selecting r^*.

a) Least Confident Prediction (LC): Perhaps, the simplest strategy to choose the most uncertain review is to choose the review for which the classifier is the least confident about predicting a class [7]. A probabilistic classifier, typically, predicts the class of an unlabelled instance to be the class corresponding to the highest predicted probability. Let $\hat{c} = \text{argmax}_{c \in C} P_\theta(c|r)$ be the predicted class. Then, we can choose r^*_{LC} as follows:

$$r^*_{LC} = \underset{r \in R}{\text{argmax}} \Big[1 - P_\theta(\hat{c}|r)\Big]$$

For example, considering that $C = \{c_1, c_2, c_3\}$, $R = \{r_1, r_2, r_3\}$, and the prediction probabilities as shown in Table II, the LC strategy chooses $r^*_{LC} = r_1$.

b) Smallest Margin (M): Although potentially effective, a problem with the LC strategy is that it only considers probability of one class (that corresponding to the highest predicted probability). However, just the highest probability may not be indicative of the uncertainty of the prediction. An alternative strategy is to employ *margin*—the difference of the two highest probabilities—in the computation of uncertainty [36]. Let $\hat{c_a}$ and $\hat{c_b}$ be the classes corresponding to two highest predicted probabilities, respectively. Then, we can choose:

$$r^*_M = \underset{r \in R}{\text{argmin}} \Big[P_\theta(\hat{c_a}|r) - P_\theta(\hat{c_b}|r)\Big],$$

For example, as Table II shows, although LC chooses r_1, M chooses r_2 since the margin for r_2 is smallest of the three.

c) Highest Entropy (H): The last strategy we consider is based on the classic notion of *entropy*. Although the margin strategy considers top two probabilities, it does not consider the entire distribution of probabilities across the classes. This is particularly important for multiclass classification problems where the number of classes is greater than three. The entropy-based strategy computes the entropy over class prediction probabilities for each unlabelled instance and chooses the instance with the highest entropy. That is:

$$r^*_H = \underset{r \in R}{\text{argmax}} \Big[- \sum_{c \in C} P_\theta(c|r) \ \text{log} P_\theta(c|r)\Big]$$

For example, in Table II, the entropy-based strategy (H) chooses r_3 over r_1 and r_2.

As shown in Table II, for the same set of predictions, each of three strategies may choose a different review instance for labelling next. However, it is important to note that the three strategies are equivalent for a binary classification problem.

TABLE II
EXAMPLES DEMONSTRATING THE CHOICE OF r^* FOR EACH OF THE THREE UNCERTAINTY SAMPLING STRATEGIES

r	$P_\theta(c_1\|r)$	$P_\theta(c_2\|r)$	$P_\theta(c_3\|r)$	$1 - LC$	M	H	r^*
r_1	0.3	0.2	0.5	0.5	0.2	0.447	r^*_{LC}
r_2	0.05	0.55	0.4	0.45	0.15	0.37	r^*_M
r_3	0.24	0.25	0.51	0.49	0.26	0.448	r^*_H

That is, in a binary classification problem, each of three strategies chooses the same review instance for labelling next.

III. EXPERIMENTS

Evaluating our active learning pipeline is challenging, considering that there are multiple sources of variation, including: (1) the type of classification task (binary versus multiclass), (2) the size of the training set, (3) the classification technique, and (4) the active learning strategy. We systematically vary these parameters, and, for each case, evaluate the benefits of incorporating active learning for app review classification.

A. Dataset

For our evaluation, we require a dataset that (1) consists of labelled reviews (to serve as training set as well as ground truth for evaluation), (2) consists of more than two labels (to experiment with multiclass active learning and the corresponding strategies), and (3) is sufficiently large (so that we can experiment with training sets of varying sizes). Accordingly, we employ the dataset provided by Maalej et al. [24], [25]. Table III summarizes this dataset, showing the classes in it, the class distribution, and an example review for each class.

TABLE III
THE CLASS DISTRIBUTION AND EXAMPLE REVIEWS FROM THE APP REVIEW DATASET WE EMPLOY

Class	Size	Example review
Feature request	299	This app is awesome and makes recording everything so easy, the only thing I can request is to make it compatible with iPads!
Bug report	378	I liked very much the upgrade to pdfs (divisions and search) However, they aren't displaying anymore. Fix it and it will be perfect.
User experience	737	This is a great little app; especially for those with hectic schedules, it keeps you in like for visual people like me.
Rating	2,721	Very nice app.

B. Experimental Setup

1) Binary and Multiclass: We train four binary classifiers (feature request, bug, user experience, and rating) and a multiclass classifier. We create a dataset for each type of classifier. For each binary classifier, we choose reviews from the corresponding class as positive set and a sample of reviews from the remaining three classes as negative set. For the multiclass classifier, we select reviews from all four classes.

2) Incremental Training: Given a classifier's dataset, we divide it into three sets as shown in Figure 3. We chose the sizes of initial training (20%) and final test (30%) sets, based on experimentation, such that there were sufficient reviews from each class for initial training and final test, respectively.

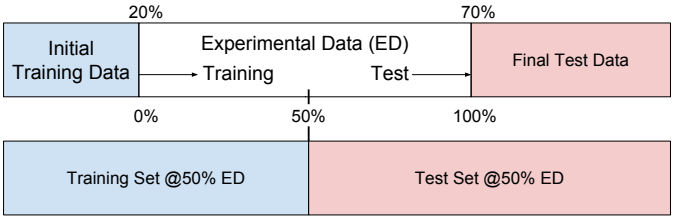

Fig. 3. Experimental setup for incremental training

We employ an incremental training setup common for evaluating active learning. That is, each of our experiments involves multiple training and testing iterations. In the first iteration, the training set consists of initial training data and the test set consists of the experimental data (ED) and the final test data. In the subsequent iterations, we select instances from the experimental data, add them to the training set and remove them from the test set. We stop this process when the size of the test set reaches the size of the final test data.

3) Features: We use the bag-of-words approach to identify the features from the reviews in the training set. After removing stop words and lemmatizing the tokens, we consider each remaining token (unigram) as a feature. Note that we did not perform feature selection since our objective is to measure the effectiveness of active learning. Feature selection is an additional step in the pipeline which can influence the effectiveness of both active and baseline learning methods.

4) Classifiers: We experiment with three classification algorithms—naive Bayes, logistic regression, and SVM. We employ the Scikit-learn [35] implementation of each. For most experiments (unless noted otherwise), we found similar patterns of results for each classifier; thus, we only report results for the naive Bayes classifier.

5) Baseline and Active Learning: We train two binary classification variants: baseline (BL) and active learning (AL). For multiclass, we train a baseline (BL) variant, and a variant for each active learning strategy (AL_{LC}, AL_{M}, and AL_{H}).

The experimental setup is identical for each variant. Each variant employs the same initial training set. However, in the subsequent iterations, their training (and test) sets differ. Whereas the baseline strategy randomly increments the training set, the active learning variants increment the training set according to their uncertainty sampling strategy.

C. Metrics

We evaluate classification performance via the standard metrics of precision, recall, and F_1 scores. For multiclass, we compute per-class and aggregate measures (macro-precision, macro-recall, and macro-F_1 scores measure the mean values of per-class precision, recall, and F_1 scores, respectively).

Precision and recall may not be equally important for all usecases. For instance, in the usecase of automatically assigning bugs from app reviews to developers, it may be more important to precisely classify whether or not a review is a bug report. On the other hand, if the usecase is to find creative feature requests from app reviews, a high recall may be preferable since we want to go through as many feature requests as possible to identify the creative ones.

Our work explores app review analysis as a generic technique, not for a specific purpose. Accordingly, we report F_1 scores (weighing precision and recall equally). Depending on the goal of app review analysis, one may weigh precision and recall differently. In such cases, a weighted version of F score, F_β, can be used to evaluate the classifier [2].

D. Statistical Tests

Our experimental setup has two sources of randomness. First, the initial training set is randomly selected. Second, for the baseline classifier, the training set is selected randomly at each iteration. Thus, the results of our experiment can vary from one run to another. To make sure that the differences in results we observe are not purely by chance, we run our experiment 30 times and compare the samples via Wilcoxon's ranksum test, which is non-parametric and does not make any assumptions about the underlying distributions [16].

IV. Results and Discussion

A. Binary Classification

Figure 4 compares the evaluation metrics for the baseline and the active learning classifiers for the binary classification task. Since we repeated our experiments 30 times, we show the mean values measured from the 30 runs of the experiments in this and other similar plots. Figure 5 compares the distribution of precision, recall, and F_1 scores for baseline and active learning classifiers when the training size is maximum, i.e., initial training data plus 100% experimental data (ED).

We make several key observations from Figures 4 and 5.

1) F_1 Scores: As evident from Figure 4 (third row), the F_1 scores for active learning are consistently higher than those for the baselines across training set sizes and for each binary classifier. Further, as Figure 5 (third column) shows, the differences between the baseline and active learning classifiers are statistically significant. Accordingly, we conclude that:

> A binary app review classifier trained on an actively learned training set yields a higher F_1 score than a baseline classifier trained from passively (randomly) chosen training set.

2) Precision: From Figures 4 and 5, we observe that, the precision values for active learning are significantly better for Bug Report, Feature Request, and User Experience classifiers. However, the difference is not significant for the Rating classifier. We attribute this result to the way in which active learning picks training instances in our setting. That is, as active learning picks new training instances in each iteration, the resulting positive and negative classes in the training set becomes increasingly more representative of the positive and

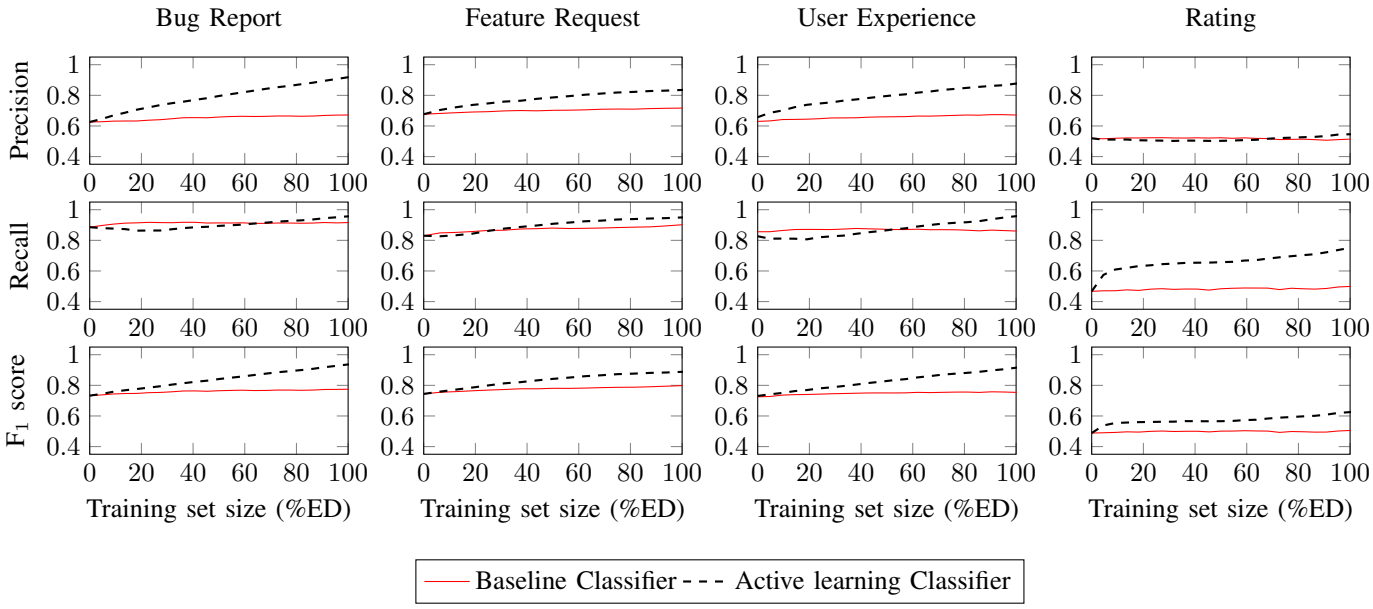

Fig. 4. Comparing the active learning and baseline classifiers in the binary classification task

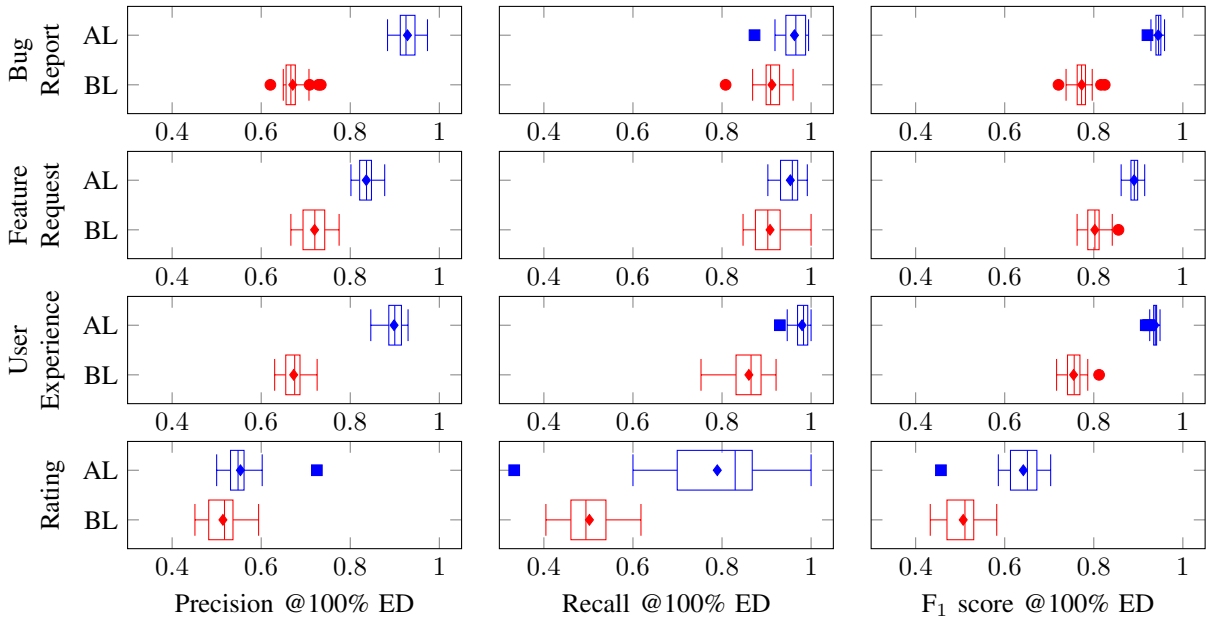

Fig. 5. Comparing the Precision, Recall, and F_1 scores for baseline (BL) and active learning (AL) classifiers for maximum training set (100% ED)

negative instances in the overall dataset, respectively. As a result, when tested, the classifier yields fewer false positives, and thus, higher precision, in each successive iteration.

In contrast, we cannot claim the same pattern for the baseline classifier. Since the baseline classifier adds new training instances randomly, the distributions of reviews in the classes may not be representative of the of entire dataset even when we increase the training set, explaining our observation that precision curves are almost flat for the baseline classifiers.

For the Rating classifier, we note that the Rating class is much larger than other classes (Table III). Further, the class is also "noisy" or less-structured in that there are a variety of ways to merely express a rating. In such cases, we conjecture that active learning, similar to baseline, is unable to pick representative reviews for the training set.

A binary AL classifier yields a higher precision than a BL classifier when the classes are sufficiently well structured. For app review analysis, the Bug Report, Feature Request, and User Experience classes are sufficiently well structured.

3) Recall: For all but Rating classifier, the recall values for both baseline and active learning are high, in general.

Further, in all cases, the recall for active learning increases as the training set size increases. Although the recall for active learning is slightly lower than the baseline initially, it picks up and outperforms baseline for larger training sets.

We attribute this observation to the distribution of reviews between positive and negative classes in the dataset. Specifically, in these binary classification tasks, the positive classes contain similar reviews whereas the negative classes are a lot more diverse. For example, for the Bug Report classifier, the positive class contains reviews about Bug Reports whereas the negative class contains all other types of reviews. As a result, the classifiers (AL or BL) tend to classify more reviews as positive class, in essence, yielding high recall values. Further, as active learning makes the positive class in the training set more representative of the same in the test set, it is able to increase the recall, but the baseline fails to do so.

A binary AL classifier yields a higher recall than a BL classifier when the training set is sufficiently large.

To further demonstrate the value of active learning, Table IV shows the evaluation metrics for the baseline classifier when the training set size is maximum (100% ED), and the training set size active learning requires to consistently outperform baseline on each of the evaluation metrics.

TABLE IV
TRAINING SET SIZE (IN %ED) THE ACTIVE LEARNING CLASSIFIER TAKES TO CONSISTENTLY OUTPERFORM THE BASELINE CLASSIFIER TRAINED WITH MAXIMUM (100% ED) TRAINING SET

Class	Precision	AL (%ED)	Recall	AL (%ED)	F_1	AL (%ED)
Bug Report	0.67	14	0.92	68	0.77	18
Feature Request	0.72	12	0.90	56	0.80	25
User Experience	0.67	9	0.86	54	0.75	14
Rating	0.51	68	0.50	5	0.50	5

In all binary app review classification tasks, active learning outperforms the baseline on all evaluation metrics with only a fraction of the training dataset the baseline employs.

B. Multiclass Classification

Figure 6 shows the per-class and macro-averaged evaluation metrics for the multiclass classifiers. It compares the metrics for baseline classifier and the classifier for each active learning strategy. Figure 7 compares the distribution of the macro metrics when the training set is maximum (100% ED).

1) Macro-Averaged Metrics: As evident from Figure 6, the classifiers trained via active learning strategies outperform the baseline classifier, on each macro-averaged evaluation metric, consistently across different training set sizes, demonstrating that active learning is valuable for multiclass classification.

An actively trained multiclass app reviewer classifier yields a better overall (macro) precision, recall, and F_1 score than a passively trained multiclass app reviewer classifier.

The multiclass classifiers (baseline or active learned) yield a lower accuracy compared to their binary counterparts. This is not surprising—going from two to four classes, learning the class (decision) boundaries is much more complex.

2) Per-Class Metrics: We observe from Figure 6 that the per-class metrics for active learning strategies are better than those for baseline in all cases with two main exceptions. First, for the Rating class, the precision values of the active learning strategies and baseline are almost same. Second, for Feature Request class, the recall values for the active learning strategies seem to be slightly worse than the baseline.

We borrow intuitions from the binary classification results to explain these exceptions. First, as we mentioned, the Rating class is larger and less-structured compared to the other classes. Thus, both baseline and active learning fail to choose a training set representative of the corresponding class in the overall dataset. Second, we note that Feature Request is the minority class among the four classes (Table III). As we observed in the binary case, the recall was lower for active learning when the training set size was small. We conjecture that the recall for the Feature Request class in the multiclass case is lower for a similar reason, and that it will go up when more training instances are added to that class.

An actively trained multiclass app reviewer classifier yields a better per-class precision, recall, and F_1 score than a passively trained multiclass app reviewer classifier when the corresponding class is sufficiently large and well structured.

C. Uncertainty Sampling Strategies

Based on the comparisons in Figures 6 and 7, we note that, there were no significant differences in the performances of the three uncertainty sampling strategies we employed. However, this result was specific to the naive Bayes classifier.

Table V shows a snapshot of our results for additional classifiers. For the logistic regression and support vector machines (SVM) classifiers, we observe that AL strategies based on least confident prediction (AL_{LC}) and margin (AL_M) outperform the strategy based on entropy (AL_H). However, these results were not consistent across all settings we tried.

For the active multiclass classification, there was no clear winner among the the uncertainty sampling techniques across (macro and per-class) metrics, classification techniques, and training set sizes. However, for logistic regression and SVM, AL_{LC} and AL_M outperform AL_H on macro metrics.

Our finding above is similar to Settles and Craven's finding [38] that there was no clear winner among different active learning strategies (they compare active learning strategies for different sequence labelling tasks on multiple corpora).

D. Summary

We highlighted our main findings above. Overall, they suggest that active learning classifiers outperform baseline classifiers under several app review classification settings. However, smaller training set sizes and the extent of noise in the classes may degrade the active learning performance.

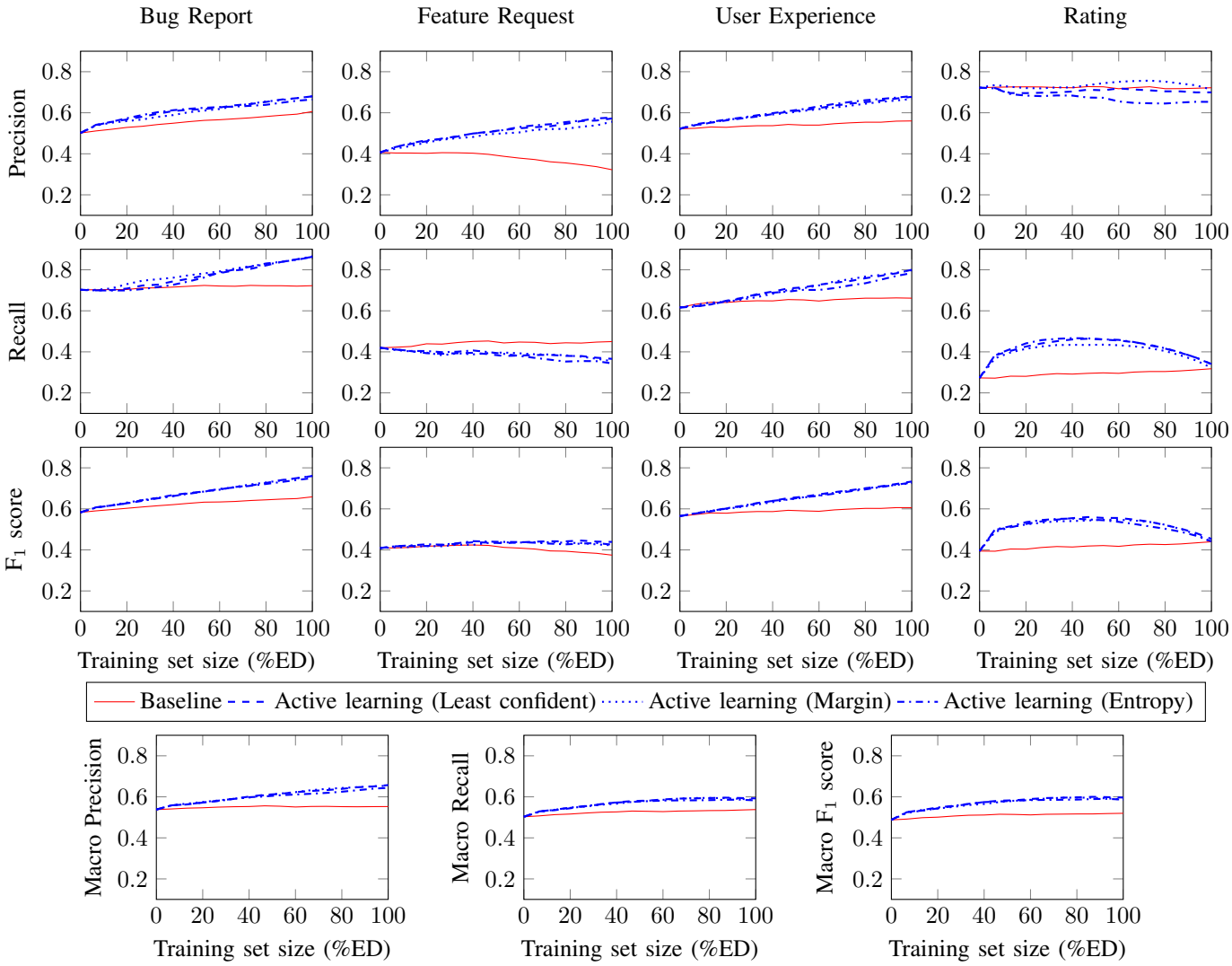

Fig. 6. Comparing per-class (above legend) and macro-averaged (below legend) metrics for baseline and active learning (three strategies) multiclass classifiers

TABLE V
THE MACRO-AVERAGED METRICS FOR DIFFERENT MULTICLASS CLASSIFIERS TRAINED AT 100% ED SIZE

Classifier	BL			AL_{LC}			AL_M			AL_H		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Naive Bayes	0.55	0.54	0.52	0.66	0.59	0.60	0.65	0.59	0.59	0.64	0.58	0.59
Logistic Regression	0.58	0.58	0.57	0.69	0.68	0.69	0.69	0.68	0.68	0.66	0.65	0.65
Support Vector Machines	0.55	0.55	0.54	0.68	0.67	0.67	0.70	0.69	0.69	0.64	0.63	0.63

In general, active learners are more effective than corresponding passive learners for most classification tasks. However, as Castro and Nowak [5] prove, the extent to which an active learner is more effective than a passive learner can vary significantly. The difference can be trivial or orders of magnitude based on noise conditions in the data. Thus, the benefits of active learning must be quantified specific to each domain. In that spirit, our work is the first to empirically evaluate active learning for app review classification.

E. Threats to Validity

We identify three threats to the validity of our results and describe our efforts toward mitigating those threats.

First, the accuracy of our results depend on the quality of the labelled ground truth data. To mitigate a *construct validity* threat of labeling bias, we use an existing dataset [24] labelled with the majority class for each review—bug report, feature request, experience, or rating. However, the genralizability of our findings is still a threat. Although we chose a dataset which is a random sample of over 1,300,000 reviews for about 1,200

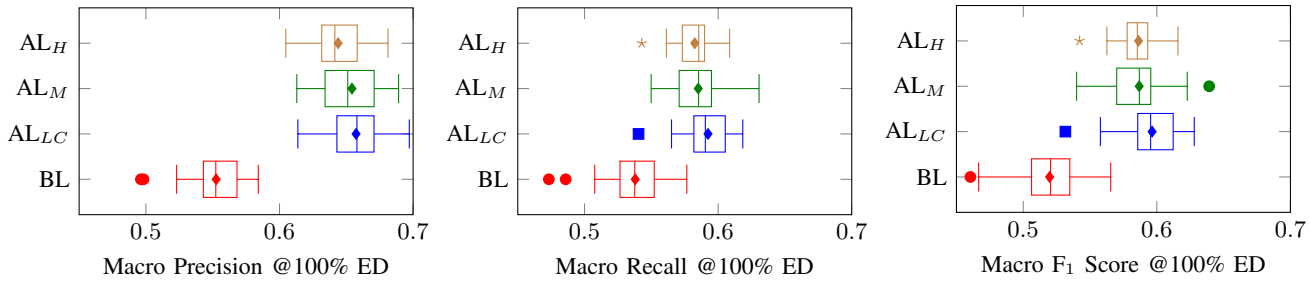

Fig. 7. Macro evaluation metrics for the baseline (BL), and least confident (LC), margin (M), and entropy (H) based active learning classifiers.

iOS and Android apps, we acknowledge that experiments with additional datasets are necessary to claim generalizability.

Second, the baseline approach in our evaluation randomly chooses an initial set to acquire labels. Because of randomness, the baseline approach can get lucky (or unlucky). To mitigate this bias, we run our experiments multiple times ($n = 30$), and report mean values of the metrics as well as their distributions in most cases. Further, we conduct rigorous statistical tests to indicate whether our findings are significant or not.

Finally, we notice that several reviews in the dataset we use could be classified under more than one class label, but the truth labels only contain the majority class. While our focus was to reduce human effort in labeling and not to improve classification accuracy, future works could consider building and using a dataset with multiple labels and employing multi-label classification techniques to improve classification accuracy.

F. Directions

1) Active Multilabel Classification: We manually analyzed several misclassified instances by both baseline and active learning classifiers. For the baseline, when an instance was misclassified, there were not any reviews similar to it in the training set but there were many such test instances. These are the cases where active learning is potentially beneficial.

We also analyzed scenarios in which active learning failed to make correct predictions and made an interesting observation. Often, both active learned and baseline classifiers made incorrect predictions when the ground truth was controversial. For example, the following review is labelled as a Feature Request in the training dataset: "This app is quite nice but ever since the last update it keeps auto deleting. I have re-installed it thrice including today. That is just wrong!" However, one may argue that this review could have been labelled as a Bug Report (and there are similar reviews labelled as Bug Reports). Such instances throw off the active learner (as well as the baseline).

A promising direction toward addressing the challenge above is multilabel classification. As McIlroy et al. [26] suggest a third of app reviews raise more than one type of issue, for instance, a review might contain both a feature request and a bug report. Specifically, researchers have proposed active learning techniques for multilabel classification problems [46], [22]. Their effectiveness for classification of app reviews is an interesting direction for future work.

2) Active and Semisupervised Learning: Although active learning optimizes the human effort spent for labelling, eventually it only learns from labelled instances, ignoring a much larger set of unlabelled reviews. Semisupervised learning [47], a complementary technique to active learning, exploits the unlabelled instances that a classifier is most confident in predicting, e.g., via *self-training*, in contrast to an active learner that solicits human labelling for least confident predictions. A future direction is to combine the two themes, where the active learner first picks reviews to be labelled by humans and the semisupervised learner is trained with both labelled and unlabelled reviews. We conjecture that such a pipeline would perform better than a solo active or semisupervised learning pipeline. However, a semisupervised learner assumes that there is an inherent structure in the unlabelled data that the learner can recognize. It remains to be seen how well that assumption holds for the noisy review data.

3) Active Learning in Context: We employed active learning in a passive learning scenario, where the app reviews have assumed to have been generated, but we are seeking labels for them. However, active learning can also be used in more "active" scenarios where feedback can be elicited in real time. To this end, Murukannaiah and Singh [32] describe an active learning strategy for eliciting *context* labels from smartphone users. An interesting extension to this work would be to elicit requirements from users in context via active learning.

V. Related Work

Several research works have focused on analyzing app reviews [6], [14], [20], [43], app descriptions [18], and discussions about apps on various platforms including Reddit and Twitter [12], [17], [19], to assist app developers and analysts better understand end-user needs. We review these works and others related to active learning in software engineering.

A. App Reviews Analysis

Guzman and Maalej [14] mine app features and sentiments associated with these features. Their approach is frequency based where non-common features can go undetected.

Johann et al. [18] propose a technique for gathering feature information from app pages and app reviews. They match features extracted from reviews and app description via binary text similarity function, results of which are not always accurate if the number of words in both candidates being compared

are not same. The deep manual approach adopted in their work can be simplified by incorporating active learning.

Researchers have also employed unsupervised techniques to analyze reviews. Chen et al. [6] propose AR-miner to filter noninformative reviews from informative reviews, and rank the reviews based on their significance. Binary class classification is a simpler problem where we need to learn one boundary to distinguish. Achieving high accuracy on a multiclass problem with unsupervised learning is challenging. Gu and Kim [11] develop SUR-Miner, a review summarization framework. It classifies reviews, identifies various aspects and opinions about those aspects in the review, and generate visual summaries.

Iacob and Harrison [17] develop MARA, a tool to extract feature requests from app reviews. MARA mines features via manually crafted keywords and linguistic rules, and employs LDA to identify topics associated with features. McIlroy et al. [26] deal with automatically assigning multilabels to user reviews for detecting anomalous apps, and experiment with various multilabeling approaches and classifiers.

Villarroel et al. [43] show how user reviews can be clustered and prioritized for release planning. Unlike our approach, the reviews in their work are classified as either bug reports or features using Random Forest machine learning algorithm on basis of predictor variables (i.e., ratings in this case). Palomba et al. [34] use clustering algorithms to group user reviews with similar user needs and feature suggestions. They classify reviews as either information giving, information seeking, feature request or problem discovery, and cluster preprocessed source code and user feedback. These clusters of feedback are linked to corresponding classes in source code which require modifications for accommodating user suggested features.

B. Requirements Analysis and Classification

Guzman, Alkadhi and Seyff [12] discuss the importance of analyzing reviews to improve an application. They perform a manual content analysis of tweets and identify tweet categories relevant to different stakeholders. They automate the manual process by employing SVM and decision trees.

Williams and Mahmoud [45] manually classify 4,000 tweets as bug reports and user requirements, and then employ SVM and naive Bayes to categorize useful tweets. Guzman, Ibrahim and Glinz [13] also mine tweets to identify requirements. Active learning could assist in labeling process of these works.

Other relevant RE research works include frameworks and techniques to analyze, classify or extract user requirements either manually or automatically. Munaiah et al. [29] build one-class classification technique for security requirements. Their approach relies on security and non-security requirements being labelled. Thomas et al. [40] develop an analytic framework and technique to identify privacy requirements from contextual factors such as actors, information and places, and to refine the identified requirements. Kanchev et al. [19] propose Canary, a query language to extract requirements including arguments supports and rebuts from online discussions. Canary depends on crowd annotated database. Crowd effort in Canary can be made effective by employing active learning.

C. Active Learning in Software Engineering

While active learning has not been employed in requirements engineering or for app review analysis, with a motivation to reduce the labeling effort, few works have employed active learning in software engineering to classify test cases and to identify defects. Lucia et al. [23] propose an active learning approach to classify true positives and false positives in anomaly reports. Wang et al. [44] propose LOAF, an active learning based technique to classify test reports that are useful and that are not. They compare LOAF with three baseline active learning strategies—margin sampling, least confidence, and informative and representative and found that LOAF yields better accuracy and efficiency than the baseline. However, these works are limited to binary classification.

Murukannaiah and Singh [32] develop Platys, an active learning based framework to learn a user's model of places. They empirically evaluate Platys via a developer study and find that their framework significantly reduces development effort. Murukannaiah and Singh's work elicits users places via active learning. The effectiveness of incorporating this approach in eliciting detailed requirements from users remains to be seen. A key challenge here is to not frustrate the users.

Most similar to our work is Thung, Li and Lo's [41] work on defect categorization. They develop an active and semisupervised multi-class classification method to classify defects into three defect families—control and data flow, structural, and non-code, and find that their approach performs significantly better than the baseline. Compared to Thung, Li and Lo's dataset that only contain 500 bug reports for three apps, our dataset draws a sample of 4,400 reviews from over 1,300,000 reviews covering nearly 1,200 apps. Whereas bug reports are likely to be better structured since they are written by testers and developers, app reviews written by end users are likely to be more unstructured and noisy, making the task of app review classification significantly harder.

VI. Conclusions

Two emerging themes in RE are exploiting Artificial Intelligence (e.g., machine learning and NLP for RE), and exploiting human intelligence (e.g., Crowd RE). These two themes are complimentary [31] in that AI techniques are often inexpensive, but may not be effective to the creative field of RE without human involvement. In contrast, Crowd RE techniques are expensive and may not be viable without some automation.

Active learning, as we employ in the paper, seeks to bridge these two themes by efficiently spending the human effort in an automated review classification task. Although automated approaches have been proposed to assist in searching for valuable app reviews, unsupervised techniques fail to achieve nuanced tasks, whereas supervised techniques incur human effort for labelling. We propose to exploit active learning as a middleground solution, which seeks to optimize the human effort incurred in a supervised approach by choosing the most informative training set. As our results indicate, active learning can significantly reduce the human effort required for training app review classifiers valuable to RE.

REFERENCES

[1] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? a study of classifying requirements," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, Ottawa, Sep. 2017, pp. 496–501.

[2] D. M. Berry, "Evaluation of tools for hairy requirements and software engineering tasks," in *Proceedings of 25th IEEE International Requirements Engineering Conference Workshops (REW)*, Lisbon, Sept 2017, pp. 284–291.

[3] T. D. Breaux and F. Schaub, "Scaling requirements extraction to the crowd: Experiments with privacy policies," in *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE)*, Karlskrona, Aug. 2014, pp. 163–172.

[4] L. V. G. Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, San Francisco, May 2013, pp. 582–591.

[5] R. M. Castro and R. D. Nowak, "Minimax bounds for active learning," *IEEE Transactions on Information Theory*, vol. 54, no. 5, pp. 2339–2353, May 2008.

[6] 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 *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, Hyderabad, May 2014, pp. 767–778.

[7] A. Culotta and A. McCallum, "Reducing labeling effort for structured prediction tasks," in *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, Pittsburgh, Jul. 2005, pp. 746–751.

[8] 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 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, Seattle, Nov 2016, pp. 499–510.

[9] EDC, "Mobile developer population reaches 12M worldwide, expected to top 14M by 2020," https://evansdata.com/press/viewRelease.php?pressID=244, Oct. 2016.

[10] E. C. Groen, N. Seyff, R. Ali, F. Dalpiaz, J. Doerr, E. Guzman, M. Hosseini, J. Marco, M. Oriol, A. Perini, and M. Stade, "The crowd in requirements engineering: The landscape and challenges," *IEEE Software*, vol. 34, no. 2, pp. 44–52, Mar. 2017.

[11] X. Gu and S. Kim, "What parts of your apps are loved by users?" in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Lincoln, NE, Nov. 2015, pp. 760–770.

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

[13] E. Guzman, M. Ibrahim, and M. Glinz, "A little bird told me: Mining tweets for requirements and software evolution," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, Lisbon, Sep. 2017, pp. 11–20.

[14] E. Guzman and W. Maalej, "How do users like this feature? A fine grained sentiment analysis of app reviews," in *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE)*, Karlskrona, Aug. 2014, pp. 153–162.

[15] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, San Francisco, May 2013, pp. 392–401.

[16] M. Hollander and D. A. Wolfe, *Nonparametric Statistical Methods*. Wiley, 1999.

[17] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, San Francisco, Oct. 2013, pp. 41–44.

[18] T. Johann, C. Stanik, A. M. A. B., and W. Maalej, "SAFE: A simple approach for feature extraction from app descriptions and app reviews," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, Lisbon, Sep. 2017, pp. 21–30.

[19] G. M. Kanchev, P. K. Murukannaiah, A. K. Chopra, and P. Sawyer, "Canary: Extracting requirements-related information from online discussions," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, Lisbon, Sep. 2017, pp. 31–40.

[20] Z. Kurtanovic and W. Maalej, "Mining user rationale from software reviews," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, Lisbon, Sep. 2017, pp. 61–70.

[21] Z. Kurtanovi and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, Ottawa, Sep. 2017, pp. 490–495.

[22] X. Li and Y. Guo, "Active learning with multi-label SVM classification." in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, Beijing, Aug. 2013, pp. 1479–1485.

[23] Lucia, D. Lo, L. Jiang, and A. Budi, "Active refinement of clone anomaly reports," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, Zurich, Jun. 2012, pp. 397–407.

[24] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, Sep. 2016.

[25] W. Maalej and H. Nabil, "On the automatic classification of app reviews: Project data," https://mast.informatik.uni-hamburg.de/app-review-analysis/, accessed: March 2018.

[26] S. Mcilroy, N. Ali, H. Khalid, and A. E. Hassan, "Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1067–1106, Jun. 2016.

[27] S. McIlroy, W. Shang, N. Ali, and A. E. Hassan, "Is it worth responding to reviews? studying the top free apps in google play," *IEEE Software*, vol. 34, no. 3, pp. 64–71, May 2017.

[28] I. Morales-Ramirez, D. Muante, F. Kifetew, A. Perini, A. Susi, and A. Siena, "Exploiting user feedback in tool-supported multi-criteria requirements prioritization," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, Lisbon, Sep. 2017, pp. 424–429.

[29] N. Munaiah, A. Meneely, and P. K. Murukannaiah, "A domain-independent model for identifying security requirements," in *Proceedings of the IEEE 25th International Requirements Engineering Conference (RE)*, Lisbon, Sep. 2017, pp. 506–511.

[30] P. K. Murukannaiah, N. Ajmeri, and M. P. Singh, "Acquiring creative requirements from the crowd: Understanding the influences of personality and creative potential in Crowd RE," in *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE)*, Beijing, Sep. 2016, pp. 176–185.

[31] ——, "Toward automating Crowd RE," in *Proceedings of the IEEE 25th International Requirements Engineering Conference (RE)*, Lisbon, Sep. 2017, pp. 512–515.

[32] P. K. Murukannaiah and M. P. Singh, "Platys: An active learning framework for place-aware application development and its evaluation," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 3, pp. 1–33, May 2015.

[33] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE)*, Rio de Janeiro, Jul. 2013, pp. 125–134.

[34] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. C. Gall, F. Ferrucci, and A. D. Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *Proceedings of the 39th International Conference on Software Engineering (ICSE)*, Buenos Aires, May 2017, pp. 106–117.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and douard Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct. 2011.

[36] T. Scheffer, C. Decomain, and S. Wrobel, "Active hidden Markov models for information extraction," in *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis (IDA)*, Cascais, Sep. 2001, pp. 309–318.

[37] B. Settles, *Active Learning*. Morgan & Claypool, 2012.

[38] B. Settles and M. Craven, "An analysis of active learning strategies for sequence labeling tasks," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Honolulu, Oct. 2008, pp. 1070–1079.

[39] Statista, "Number of mobile app downloads worldwide in 2017, 2018 and 2022 (in billions)," https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/, May 2018.

[40] K. Thomas, A. K. Bandara, B. A. Price, and B. Nuseibeh, "Distilling privacy requirements for mobile applications," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, Hyderabad, May 2014, pp. 871–882.

[41] F. Thung, X.-B. D. Le, and D. Lo, "Active semi-supervised defect categorization," in *Proceedings of the 23rd IEEE International Conference on Program Comprehension (ICPC)*, Florence, May 2015, pp. 60–70.

[42] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of Machine Learning Research*, pp. 45–66, Mar. 2002.

[43] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. D. Penta, "Release planning of mobile apps based on user reviews," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, Austin, May 2016, pp. 14–24.

[44] J. Wang, S. Wang, Q. Cui, and Q. Wang, "Local-based active classification of test report to assist crowdsourced testing," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Singapore, Aug. 2016, pp. 190–201.

[45] G. Williams and A. Mahmoud, "Mining twitter feeds for software user requirements," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, Lisbon, Sep. 2017, pp. 1–10.

[46] B. Yang, J.-T. Sun, T. Wang, and Z. Chen, "Effective multi-label active learning for text classification," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Paris, Jun. 2009, pp. 917–926.

[47] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.