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 the 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 misclassification 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 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 crowdsourcing 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).

As shown in Table I, suppose that the developers have somehow acquired labels for four reviews (r₁–r₄) so far. The question, then, is which review to acquire the fifth label for. Suppose that there are four candidates (r₅–r₈) 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 r₆. 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 (r₅, r₇, and r₈) 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 r₈ as a feature request, and r₅ and r₇ as not, yielding a very low (33%) prediction accuracy.

Instead of choosing r₆ as the fifth training instance, what if we somehow chose r₅? 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 (r₆–r₈) as a feature request, yielding 100% accuracy.

Thus, a key question is: how do we decide to choose r₅ instead of r₆ before acquiring the label (and training the classifier with five instances)? The problem with adding r₆ 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 r₆ (which, too, consists of “add”), does not provide any new information. In contrast, including r₅ 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

<table>
<thead>
<tr>
<th>ID</th>
<th>Review</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₁</td>
<td>Very convenient app where I can store all of my photos+ videos with easy access</td>
<td>✓</td>
</tr>
<tr>
<td>r₂</td>
<td>Please add scan with jpg format</td>
<td>✓</td>
</tr>
<tr>
<td>r₃</td>
<td>The app is ok but their plans are too expensive</td>
<td>✓</td>
</tr>
<tr>
<td>r₄</td>
<td>Please introduce something like Dropbox lite! 55 Mb</td>
<td>✓</td>
</tr>
<tr>
<td>r₅</td>
<td>It does not support multi-selection that I need to download one by one on web version</td>
<td>?</td>
</tr>
<tr>
<td>r₆</td>
<td>Please add dark themes</td>
<td>?</td>
</tr>
<tr>
<td>r₇</td>
<td>No support for OCR on PDFs? I’m looking to move...</td>
<td>?</td>
</tr>
<tr>
<td>r₈</td>
<td>Introducing an option to automatically save photos from Micro SD card will be great!</td>
<td>?</td>
</tr>
</tbody>
</table>

TABLE I

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

As shown in Table I, the developers have somehow acquired labels for four reviews (r₁–r₄) so far. The question, then, is which review to acquire the fifth label for. Suppose that there are four candidates (r₅–r₈) 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 r₆. 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 (r₅, r₇, and r₈) 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 r₈ as a feature request, and r₅ and r₇ as not, yielding a very low (33%) prediction accuracy.

Instead of choosing r₆ as the fifth training instance, what if we somehow chose r₅? 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 (r₆–r₈) as a feature request, yielding 100% accuracy.

Thus, a key question is: how do we decide to choose r₅ instead of r₆ before acquiring the label (and training the classifier with five instances)? The problem with adding r₆ 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 r₆ (which, too, consists of “add”), does not provide any new information. In contrast, including r₅ 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
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 labelers), 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](image)

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](image)

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 $\Theta$ 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:

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

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 $\hat{c}_1$ and $\hat{c}_2$ be the classes corresponding to two highest predicted probabilities, respectively. Then, we can choose:

$$r^*_{M} = \text{argmin}_{r \in R} \left[ P_\theta(\hat{c}_1|r) - P_\theta(\hat{c}_2|r) \right].$$

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 classical 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:

$$r^*_{H} = \text{argmax}_{r \in R} \left[ - \sum_{c \in C} P_\theta(c|r) \log P_\theta(c|r) \right].$$

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

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

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

#### 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](image)

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\(_{L,C}\), 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, \(F_\beta\), 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
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 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

<table>
<thead>
<tr>
<th>Class</th>
<th>Precision (%)</th>
<th>AL (%)</th>
<th>Recall (%)</th>
<th>F1 (%)</th>
<th>AL (%)</th>
<th>F1 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug Report</td>
<td>0.67</td>
<td>14</td>
<td>0.92</td>
<td>68</td>
<td>0.77</td>
<td>18</td>
</tr>
<tr>
<td>Feature Request</td>
<td>0.72</td>
<td>12</td>
<td>0.90</td>
<td>56</td>
<td>0.80</td>
<td>25</td>
</tr>
<tr>
<td>User Experience</td>
<td>0.67</td>
<td>9</td>
<td>0.86</td>
<td>54</td>
<td>0.75</td>
<td>14</td>
</tr>
<tr>
<td>Rating</td>
<td>0.51</td>
<td>68</td>
<td>0.50</td>
<td>5</td>
<td>0.50</td>
<td>5</td>
</tr>
</tbody>
</table>

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.

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.

<table>
<thead>
<tr>
<th>Class</th>
<th>Precision (%)</th>
<th>AL (%)</th>
<th>Recall (%)</th>
<th>F1 (%)</th>
<th>AL (%)</th>
<th>F1 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug Report</td>
<td>0.67</td>
<td>14</td>
<td>0.92</td>
<td>68</td>
<td>0.77</td>
<td>18</td>
</tr>
<tr>
<td>Feature Request</td>
<td>0.72</td>
<td>12</td>
<td>0.90</td>
<td>56</td>
<td>0.80</td>
<td>25</td>
</tr>
<tr>
<td>User Experience</td>
<td>0.67</td>
<td>9</td>
<td>0.86</td>
<td>54</td>
<td>0.75</td>
<td>14</td>
</tr>
<tr>
<td>Rating</td>
<td>0.51</td>
<td>68</td>
<td>0.50</td>
<td>5</td>
<td>0.50</td>
<td>5</td>
</tr>
</tbody>
</table>

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.

### 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.
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 generalizability 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...
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 Maaalej [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.

Jacob 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.


