Hierarchical Bayesian Multi-kernel Learning for Integrated Classification and Summarization of App Reviews

Moayad Alshangiti
University of Jeddah, Saudi Arabia
Rochester Institute of Tech., USA
mshangiti@uj.edu.sa

Weishi Shi
Rochester Institute of Technology,
Rochester, USA
ws7586@rit.edu

Eduardo Lima
Rochester Institute of Technology,
Rochester, USA
eduardo.lima@rit.edu

Xumin Liu
Rochester Institute of Technology,
Rochester, USA
xumin.liu@rit.edu

Qi Yu
Rochester Institute of Technology,
Rochester, USA
qi.yu@rit.edu

ABSTRACT

App stores enable users to share their experiences directly with the developers in the form of app reviews. Recent studies have shown that the feedback received from users is a valuable source of information for requirements extraction, which encourages app developers to leverage the reviews for app update and maintenance purposes. Follow-up studies proposed automated techniques to help developers filter the large volume of daily and noisy reviews and/or summarize their content. However, all previous studies approached the app reviews classification and summarization as separate tasks, which complicated the process and introduced unnecessary overhead. Moreover, none of those approaches explored the potential of utilizing the hierarchical relationships that exist between the labels of app reviews for the purpose of building a more accurate model. In this work, we propose Hierarchical Multi-Kernel Relevance Vector Machines (HMK-RVM), a Bayesian multi-kernel technique that integrates app review classification and summarization using a unified model. Moreover, it can provide insights into the learned patterns and underlying data for easier model interpretation. We evaluated our proposed approach on two real-world datasets and showed that in addition to the gained insights, the model produces equal or better results than the state of the art.

CCS CONCEPTS

• **Software and its engineering → Requirements analysis**;

KEYWORDS

Bayesian Modeling, Multi-Kernel Learning, Relevant Vector Machines, App Reviews, User Requirements

ACM Reference Format:
Moayad Alshangiti, Weishi Shi, Eduardo Lima, Xumin Liu, and Qi Yu. 2022. Hierarchical Bayesian Multi-kernel Learning for Integrated Classification and Summarization of App Reviews. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549174

1 INTRODUCTION

User opinions on mobile apps are highly valued by app developers due to the competitive nature of the market [4], where developers attempt to attract the highest possible user base and maintain their satisfaction level. Thus, developers would like to analyze the feedback received from users in the form of app reviews to understand the users' requirements, preferences, and complaints [20, 31]. However, the large volume of app reviews received on a daily basis has made a manual analysis of reviews too time-consuming. As such, it became favorable to have automated approaches that can facilitate quicker and easier access to the feedback found in app reviews.

Existing efforts on this task fall into two directions. In the first one, a classification model is constructed to assign reviews into a predefined list of labels considered to be useful for app developers (*e.g.,* bug reports and feature requests) as a way to automate the process [11, 16, 20, 24, 25, 33, 40, 41]. However, assigning such general labels is inadequate to extract useful requirements as one can easily find thousands of reviews that fall under one of those labels and significant manual work is still needed to find the actual requirements. Thus, the second direction aims to summarize or group together user reviews with similar topics for easier requirement extraction [9, 13]. Visualization techniques have been used to highlight the most frequent terms used in those reviews and it is left to the developers to infer the requested feature(s). Similarly, clustering has been leveraged to group reviews that cover the same set of topics but the developers still have to analyze each cluster to identify the requirements embedded in the review content. In a more end-to-end research, both the classification and summarization tasks were attempted [15, 32, 34, 43]. We align our work with this direction. However, unlike previous work where the classification and summarization tasks were handled separately, we propose to merge the two tasks together in a single learning process.

In this work, we present a novel approach to facilitate the extraction of user requirements from app reviews in which both the classification and summarization are achieved simultaneously using a unified model. In particular, we propose Hierarchical Multi-kernel Relevant Vector Machines (HMK-RVM), in which three main goals are accomplished. First, we exploit the hierarchical relationships between the labels during the learning process to build a more accurate classifier. Second, we adopt a Bayesian multi-kernel learning approach that encapsulates a rich feature space into separate kernels, which achieved improved model interpretability and understanding of predictions. Third, in addition to a competitive classification accuracy, the proposed approach can identify a small set of most representative reviews as part of its learning process that can effectively summarize the content of all available reviews. We extensively evaluate our approach on two real-world datasets and show that it can produce equal or better classification accuracy than the state of the art while identifying the most informative reviews to greatly facilitate requirements extraction. We summarize our contributions as follows:

- We leverage the hierarchical relationships that exist in the app review labels as part of the learning process, which is a new perspective on the classification task that was not considered before. We further demonstrate that using the hierarchical relationships between the labels can lead to a more accurate model.
- We propose a multi-kernel Bayesian approach that integrates classification and summarization under a unified framework. We show that our proposed approach can offer two additional benefits beyond accuracy: (i) an insight into the learned task and the underlying data for better model interpretability, and (ii) an insight into the most representative reviews that best summarize the users feedback, for easier requirements extraction.
- We evaluate our proposed approach (HMK-RVM) on two real-world datasets and show that it can provide better classification results while providing significantly better summarization results than the state of the art.

The remainder of this paper is organized as follows. We present a summary of related work in Section 2. We discuss our proposed approach in Section 3. We evaluate our approach and present our results in Section 4. We then discuss the significance of the results and the potential threats to validity in Section 5 and Section 6. Finally, we conclude the paper in Section 7.

2 RELATED WORK

In this section, we summarize existing studies related to app reviews classification and/or summarization. We divide existing works into two major categories: (i) classification only and (ii) classification followed by summarization. For the former, a key limitation is that they do not address requirement extraction. Hence, developers need to manually analyze all informative reviews (which is usually around 35%-40% of all reviews) that leads to significant overhead. As for the latter, they usually rely on a complex pipeline that requires the implementation, tuning, and maintenance of two different ML models, one for classification and one for extraction through clustering/visualization or other relevant strategies.

2.1 Summarizing User Reviews

There are several existing works with a focus on summarizing or visualizing the overall topics found in user reviews. In [19] an approach to summarize the most discussed aspects of a product and the corresponding user opinions (*i.e.,* positive or negative) is presented. In [9], topic modeling is exploited to discover the topics found in the reviews along with representative sentences. In [11], DBSCAN clustering is used to group together similar reviews. In [44], the authors proposed an information retrieval framework that processes the reviews and put them in a knowledge database. The framework returns the most relevant reviews that discuss the provided topics given the developer-selected keywords. In [13, 15, 34], different visualization tools/techniques are presented. For example, in [34], an HTML tool was presented that visualizes the content of reviews by showing terms formatted in a word cloud. In [13, 45] they focused on providing an interface that summarizes and tracks the change in reviews under specific topics between different versions to highlight abnormal changes (*e.g.,* version 2 has significantly higher bug reports than all other versions).

2.2 Classifying User Reviews

As for app review classification, [11] is among the first attempts to classify app reviews to be either *informative* or *non-informative*. The authors used a bag-of-words (BoW) representation, similar to other studies [24, 25, 43]. In [43], the authors leveraged N-grams in the BoW representation to account for context that requires two or three words (*e.g., not laggy*). If we process those words separately, we will not understand the actual intention. In [24], the tense of the verb was incorporated into the feature space. The authors argued that verbs in the past are usually associated with users reporting bugs, whereas, verbs in the future are usually correlated with hope and requests for additions (*i.e.,* feature requests). In [33], the authors claimed that most reviews follow a specific linguistic patterns and identifying those patterns can help to improve classification performance. Thus, they created 246 linguistic patterns that describe the general form in which a review would be in to fall under a specific label (*e.g., [someone] should add [something]*). In [15], the BoW representation is replaced with a representation generated from parsing sentences as parsing trees and then traversing the tree to construct the representation. The authors claimed this approach can take word semantics into consideration. In [40, 41], the authors suggested to classify on the sentence level instead of the review level to allow for multi-label classification. It is also worth mentioning that some studies investigated connections beyond the classification of app reviews. For example, in [32], the authors investigated the possibility of linking user feedback to the source code components. Different from the studies above, we argue that there is still room to improve the automation of requirements extraction from app reviews. Therefore, we extend this line of work by integrating summarization and classification tasks while exploring unique characteristics of the problem, such as the hierarchical relationships between the labels. By doing so, we uncover new ways to further improve the automation of such tasks.

2.3 Other Related Studies on App Reviews

There is a number of studies that focus on analyzing app stores [18], types of feedback in user reviews [10, 12, 17, 21], and the interaction between these two [27]. We differ from those studies in that we are not analyzing the feedback itself. Instead, we focus on app review classification and summarization to largely automate the extraction of user requirements.

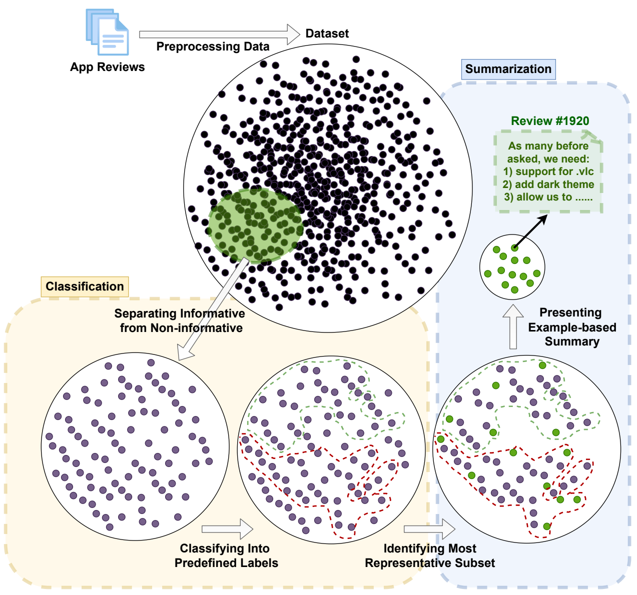

App Reviews
Preprocessing Data
Dataset
Summarization
Review #1920
As many before asked, we need:
1) support for .vlc
2) add dark theme
3) allow us to ......
Classification
Separating Informative from Non-informative
Presenting Example-based Summary
Classifying Into Predefined Labels
Identifying Most Representative Subset

Figure 1: Overview of the proposed approach for requirements extraction from app reviews

3 METHODOLOGY

Overview. In this section, we present the proposed HMK-RVM model as seen in Figure 1. Given a set of app reviews, the proposed approach first classifies each review into the predefined labels. This would help developers separate informative reviews from non-informative ones. Additionally, the predefined labels (*e.g.,* feature request, bug report, *etc.*) can further guide the developers to extract the relevant requirements. Along with the classification process, the model also identifies a set of the most representative reviews that summarize the entire collection. As a result, developers can only focus on these representative reviews for requirements extraction as they are expected to capture most of the discussed requirements.

For the rest of this section, we first discuss the difference between a flat and a hierarchical approach and justify how the latter fits better with app review classification. We then present our proposed multi-kernel RVM, which leverages Bayesian sparse learning and multiple kernels to integrate classification and summarization.

3.1 Hierarchical User Review Classification

For common classification tasks with a set of balanced classes, standard multi-class models can be straightforwardly applied. However, for problems with highly imbalanced classes (*e.g.,* those that involve rare classes), standard techniques may suffer from a poor performance due to lack of attention given to the minority class. Meanwhile, it has been shown that leveraging existing hierarchical relationship between classes can improve the performance of the classifier [22, 29]. In traditional flat classification, the hierarchical relationship between classes is ignored. For example, a binary flat classifier would attempt to distinguish app reviews with *feature requests* from all other classes. This ignores the fact that reviews with *feature requests* and/or *bug reports* are all considered as *informative* reviews, *i.e.,* they share a common parent class. Taking this information into consideration when training the classifier can help us build a better classifier that attempts to first distinguish the *informative* reviews from the *non-informative* reviews and then further classify those *informative* reviews into their appropriate class. In this way, classes at both levels tend to be more balanced.

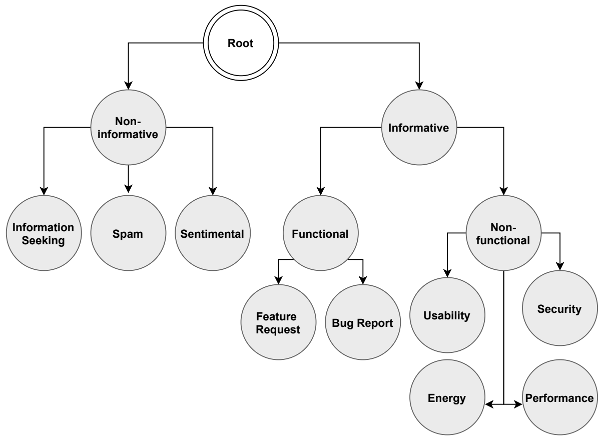


Figure 2: The hierarchical structure in app review classes

We observe that all the previous works have approached the problem as a flat classification problem. In [11], a binary classifier that determines whether an app review is *informative* or *non-informative* was used, introducing the first two types of classes. A follow up work [25] further studied the app reviews and introduced a new set of labels *rating*, *bug reports*, *feature requests*, and *user experience*. A more recent study used feedback from the industry to further break down *user experience* into reviews reporting *security* concerns, *energy* concerns, and so on. However, no existing work has attempted to leverage the hierarchical relationship between those classes as part of the learning process. Based on the analysis of previous work, it is clear that the classes of app reviews can be organized into a fairly complex hierarchy as shown in Figure 2. It has been reported in multiple studies [31] that the *informative* subset of app reviews represent at most 30%-35% of the whole corpus. If we further break down the *informative* subset into multiple classes, we can observe that some classes can be as rare as 5%-10%. As such, using traditional flat classification will create classifiers dominated by the negative (*i.e.,* non-informative) class, leading to a poor classification performance. This limitation can be addressed when a hierarchical classification approach is used.

3.2 Multi-Kernel Relevance Vector Machines

Notations. Let $X = \{\mathbf{x}_1, \mathbf{x}_2, ..\mathbf{x}_N\}$ denote a set of N training instances, where $\mathbf{x}_i \in \mathbb{R}^D$. We limit the introduction to Relevance Vector Machines (RVM) [42] to binary classification problem for simplicity where each data instance $\mathbf{x}_i$ is assigned with a label $t_i \in \{0, 1\}$. Later, the binary classification solution can be directly generalized to multi-class problem with the one-vs-the-rest formulation. The RVM is a Bayesian model in which the label follows the

Bernoulli distribution $t_i \sim \text{Bernoulli}(\sigma)$:

$$p(t_i = 1) = y_i = \sigma\left(\sum_{m=1}^{M} \phi_m(\mathbf{x}_i)w_m\right) = \sigma(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_i)) \tag{1}$$

where $\phi(\mathbf{x}_i)$ is a vector of M basis functions that projects the feature space from $\mathbb{R}^D$ to $\mathbb{R}^M$: $\phi(\mathbf{x}_i) = [\phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), ..\phi_M(\mathbf{x}_i)]$. Typical basis functions include polynomial, Gaussian, and sigmoidal [6]. In RVM, the basis functions are specified with a kernel function $k(\cdot,\cdot)$: $\phi_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i)$. We denote $K \in \mathbb{R}^{N\times N}$ as the gram matrix whose i-th row is given by $[k(\mathbf{x}_1, \mathbf{x}_i), k(\mathbf{x}_2, \mathbf{x}_i), ..k(\mathbf{x}_N, \mathbf{x}_i)]^\top$. The kernel view of (1) is given by:

$$p(t_i = 1) = y_i = \sigma\left(\sum_{n=1}^{N} w_n \mathrm{k}(\mathbf{x}, \mathbf{x}_n)\right) \tag{2}$$

where $\mathbf{w}$ are model parameters that follow a Gaussian distribution $p(\mathbf{w}; \boldsymbol{\alpha}) \sim \mathcal{N}(\mathbf{0}, A^{-1})$, with A being a diagonal matrix $A = \text{diag}(\alpha_1, ..., \alpha_N)$. The goal of RVM is to learn the posterior distribution $p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha})$ as well as to estimate the hyper-parameter $\boldsymbol{\alpha}$. Here, we omit the dependency on X, which is implied.

The posterior distribution can be derived via the Bayes' rule:

$$\ln p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) \propto \ln p(\mathbf{t}|\mathbf{w}) + \ln p(\mathbf{w}|\boldsymbol{\alpha}) \tag{3}$$

By applying Laplace approximation, the posterior distribution also follows a Gaussian distribution $\mathcal{N}(\mathbf{w}^*, \Sigma)$, whose mean and covariance are given by $\mathbf{w}^* = A^{-1}K^\top(\mathbf{t} - \mathbf{y})$ and $\Sigma = (K^\top BK + A)^{-1}$, respectively, where $B = \text{diag}(\mathbf{y} \odot (1 - \mathbf{y}))$.

The hyper-parameter $\boldsymbol{\alpha}$ can be derived using type II maximization. To do that, we first compute the model evidence

$$p(\mathbf{t}|\boldsymbol{\alpha}) = \int p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})\mathrm{d}\mathbf{w} \simeq \int p(\mathbf{t}|\mathbf{w}^*)p(\mathbf{w}^*|\boldsymbol{\alpha})\mathrm{d}\mathbf{w} \tag{4}$$

where we used Taylor expansion on the integrant at $\mathbf{w}^*$ to remove the integral. Then the optimal value of $\boldsymbol{\alpha}$ is obtained by solving $\frac{\partial p(\mathbf{t}|\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} = 0$:

$$\alpha_i^* = \frac{1 - \alpha_i \Sigma_{ii}}{(w_i^*)^2} \tag{5}$$

Training of RVM is achieved through an iterative process of updating $\mathbf{w}^*, \Sigma$, and $\boldsymbol{\alpha}_i^*$ until convergence. In the prediction phase, the predictive distribution of a test data point $\mathbf{x}'$is given by $p(t'|\mathbf{x}', \mathbf{w}^*) = \text{Bernoulli}(\sigma(\mathbf{w}^{*\top}\mathbf{x}'))$. The prior distribution adopted by RVM is commonly referred as auto relevance detection (ARD). It makes the model prefer simpler explanations than complex explanations so that over-fitting can be automatically addressed. Specifically, during the training process, a certain number of $\boldsymbol{\alpha}$'s components will be driven to infinity, making their corresponding training data instances independent from the prediction and the remaining few important training data instances are called *relevance vectors*.

We make a general extension to RVM to handle the input with multiple modalities (*e.g.,* different representations). Suppose the input X is now given in three different representations X_I, X_{II}, and X_{III}. Then, we construct a overall gram matrix as the linear combination of the gram matrix for each representation.

$$K = \theta_1 K(X_I, X_I) + \theta_2 K(X_{II}, X_{II}) + \theta_3 K(X_{III}, X_{III}) \tag{6}$$

Replacing the gram matrix in standard RVM with (6), we have the RVM for multi-modality data input. The hyper-parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)^\top$ can be solved by type II maximization similar to solving $\boldsymbol{\alpha}$. However, the objective function is not convex with respect to $\boldsymbol{\theta}$ and may cause the optimization either to trap into the local optima or commit to slow convergence. To address this, we adopt a gradient-free method, simplex [5], to directly search the optimal $\boldsymbol{\theta}$ in the hyper-parameter space.

3.3 Why Extend RVM?

A fundamental reason for using and extending RVM for our problem is its ability to identify the most representative points that can summarize the underlying dataset. This aligns well with the task of finding the best subset of reviews that can be used for requirement extraction. During model training, it ensures both the *sparsity* and the *quality* of the selected data points. The former, *i.e.,* sparsity, allows us to identify a small subset of reviews to summarize the entire dataset in a compact way. As a result, the developer can safely ignore a large portion of the reviews to significantly reduce the manual analysis effort when performing requirement extraction. The latter, *i.e.,* quality, further helps to identify the most informative reviews that can ensure the accuracy and quality of the extracted requirements. At the end of this training process, a set of points are selected which the model uses for classification. We propose to use those points for summarization as well. Consequently, we would achieve both aspects using a single learning model.

3.4 Constructing the Kernels

The number of kernels used with the approach and their types can be selected based on the available data and given task. For the purpose of app reviews classification, we constructed four kernels to build a comprehensive kernel space.

The first is a meta kernel, which captures simple meta information about the review, *e.g.,* the rating and the number of words. The second is a kernel that utilizes the textual content of the review by capturing the important recurring terms, *e.g., add*, *crash*, *etc.* To construct this kernel, we applied a standard natural language processing methods, such as stop-word removal and word stemming, on the textual content of the title and body of an app review to generate such a representation using the *term frequency-inverse document frequency* (TF-IDF) approach [26]. However, one potential disadvantage with this kernel is that the textual nature of app reviews is quite noisy, which can lead to a large and sparse dictionary. For that purpose, we constructed a third a kernel that would provide us with a less sparse representation by attempting to capture the broad topics within the reviews, in contrast to relying on the exact terms. To construct this kernel, we used the topic modeling technique, Latent Dirichlet Allocation (LDA) [7]. LDA been widely used in many previous studies [2, 3, 38] to summarize the topics of a large document corpus. The intuition behind LDA is that it leverages the textual content of a set of documents to group together the frequently co-occurring words into an approximation of a real-world concept, *i.e.,* a topic.

Even though those kernels would provide a comprehensive representation, they lack one important aspect, and that is the semantics of the used words. In [25], they found that classifying the reviews coming from the iOS app store was significantly more accurate than those coming from the Android store. They attribute this difference to the language and vocabulary difference from those two app stores. They claim that the iOS store reviews were less noisy (*e.g.,* having less typos) and used a much more homogeneous vocabulary of terms. This observation highlights the effect of the noise found in user reviews on the classification task and the impact it has on the learning process. In [44], this observation was studied further as the authors also highlighted and described the observation that app reviews suffer from a high percentage of typos, acronyms, and abbreviations. They performed a preliminary analysis of 300,000 reviews and compared their textual content against an English dictionary of 150,000 common words, and found that a large portion of the used words in app reviews do not match any words in the English dictionary, mainly due to abbreviations and typos. Having such high noise and unique language (*e.g., wait* is written as *w8*) creates an issue for traditional data mining techniques that relies on stemming and dictionary creation as both *wait* and *w8* will still exist as two unique different words. They hypothesized that this observation might be due to the fact that reviews are written using mobile devices which lack a physical keyboard, hence, it is more likely to have typos, acronyms, and abbreviations. To overcome this issue, the authors in [44] manually created a custom dictionary that attempts to replace the most frequent out-of-dictionary words with their dictionary-equivalent (*e.g.,* replacing *exelent* with *excellent*). Moreover, in [15], a similar observation was made, and the authors manually constructed a collection of 60 different typos and contractions, and replaced them using regular expressions.

We have observed a similar pattern of noise with app reviews, where a large portion of words in the post-processing and stemming dictionary seem to represent the same word but written differently due to misspelled words (*e.g., fantastic* vs *fantastick*) or alternatively spelled words either for abbreviations purposes (*e.g., thanks* vs *thx*), or to represent a stronger emotion, (*e.g., loved* vs *loooved*). In Table 1, we show a few additional examples.

Table 1: Examples of mis- or alternatively spelled words

Word	Observed noise
amazing	amaazing, amaaazing, amassing, amazeng
thanks	thx, thanx, tx, tnx, 10x, thnx, ty
awesome	awasome, awesomeeee, awsome, owesome, asssome
love	lov, luv, lovve, looove, loveee
because	bc, b/c, cuz, coz, bcz, caus

While merging misspelled or alternatively spelled words would improve the textual representation and the model's performance, we argue that using a manually created custom dictionary would be too difficult to create and maintain over the time. To overcome this issue, we constructed a fourth kernel that leverages word embedding techniques to create a representation that captures the semantics of words. For example, the word2vec [28], the GloVe [36], or the FastText [8] model are all techniques that take into consideration word semantics and meanings. These techniques are built on the notion that words with similar semantic meanings will have the same set of words around them. For example, the words *love* and *like* are used in similar manners, *i.e., I love that app* and *I like that app*. As a result, they would be closely placed in the embedding space as they share a similar semantic meaning. To construct this kernel space, we need to either use a pre-trained model from a different domain (*e.g.,* Tweets), or train our own problem-specific model to generate the word embeddings for the app reviews. According to [23, 39], which studied this specific concern among other choices with word embedding models, it is recommended to use a model that is specific to your domain as it can better capture the relevant vocabulary and their unique usage. We believe this is especially true for app reviews and an important factor to overcome the issue of misspelled and alternatively spelled words. To the best of our knowledge, there's no publicly available word embedding model that was trained on app reviews. For that reason, we decided to create such a model and make it publicly available as part of our replication package. We trained a FastText [8] model on 1,673,672 app reviews collected from [35] and [14]. We chose FastText because it is most effective in settings where out-of-dictionary words are common, like ours. For more details on the training process and the selected parameters, kindly refer to the replication package[1].

Finally, it is worth noting that our approach has the flexibility to use any number of kernels. We believe the suggested four kernels represent app reviews quite well and can capture both high-level concepts (*e.g.,* LDA) and low-level characteristics (*e.g.,* meta). As a result, the four kernels offer sufficient expressive power to construct a comprehensive feature space that can help the machine learning model achieve a good level of robustness and generalization.

4 EVALUATION AND RESULTS

In this section, we plan to evaluate the proposed model, specifically, by investigating the following research questions:

- RQ_1: Do we gain any app reviews prediction accuracy from hierarchical classification versus flat classification?
- RQ_2: How accurate is the *classification* of the proposed HMK-RVM approach compared to the state of the art?
- RQ_3: How accurate is the *summarization* of the proposed HMK-RVM approach compared to the state of the art?
- RQ_4: Beyond accuracy, what insights can we gain from using the proposed hierarchical multi-kernel RVM approach?

The source code and data used in the experiment section are available online for easy replication/validation purposes [1].

Table 2: Statistics of the used datasets

	Maalej	**Panichella**
Feature Request	252 (7%)	391 (13%)
Bug Report	370 (10%)	271 (9%)
User Experience	607 (16%)	334 (11%)
Total Info	1229 (33%)	880 (30%)
Total Non-Info	2455 (67%)	2024 (70%)
Total Reviews	3684	2904

[1]https://tinyurl.com/qup3h4l

4.1 Datasets

To address our questions, we will report results on two real-world datasets that were provided by previous research. The first is the *Maalej dataset* [24, 25], where reviews were randomly selected from both Apple and Google Play stores. The authors crawled over a million app reviews and followed a sampling strategy with the goal of picking a stratified and a representative sample (*e.g.,* equal number of free and paid apps, equal number of iOS and Android app, *etc.*). The second is the *Panichella dataset* [34, 40], where the authors favored an app specific sampling approach. The dataset contains reviews of 17 apps coming from Google Play, Microsoft, and Apple app stores. Unfortunately, the ground truth was not provided for this dataset so we asked two teams of graduate students to label the dataset separately according to a labelling guide that can be found with the replication package[1]. The guide follows closely the guidance of the original paper. Once the teams completed the labelling task, we compared the labels and addressed all disagreements. We found a good inter-rater agreement (kappa=0.68) between the annotators. The statistics of both datasets can be found in Table 2.

4.2 Experiment and Results

RQ$_1$: Do we gain any app reviews prediction accuracy from hierarchical classification versus flat classification?

Experimental Setup: To evaluate the model's accuracy gained from leveraging the hierarchical relationship embedded within the labels, we will use a simple feature space consisting of a bag-of-words representation using TF-IDF [37]. For this evaluation, we will not attempt to add any additional features such as meta-data features (*e.g.,* rating, review length, *etc.*) as we want to focus on the added benefit of hierarchical versus flat app reviews classification. Moreover, to make sure the results are not due to a specific classifier or to a specific dataset, we will evaluate on both the *Maalej* and *Panichella* datasets, and on four different classifiers: Logistic Regression (L_1 *Regularization*), Random Forest (*200 trees*), Support Vector Machines (*Linear Kernel)*, Relevant Vector Machines (*Linear Kernel)*, and Naive Bayes (*Multinomial*). As we are limited to the three mutual labels (bug report, feature request, and user experience) provided with those datasets, we will build the experiment around them. Finally, to make sure both the flat and hierarchical classifiers were exposed to the same set of reviews during training and testing, we used a single train/test split of 80/20 for both.

For flat classification, as shown in Figure 3(a), we are training three one-vs-the-rest binary classifiers, one classifier per label (*e.g.,* bug report or not). We prefer to use binary classifiers instead of a multi-class classifier as this setup allows for multi-label classification. This means an app review can be given a single or multiple labels. For example, an app review with multiple labels from the Panichella dataset is "*This is a great app for keeping track of weight ... there should be a way to turn off daily reminder ... also I notice it keeps changing the year I was born...*". However, using this setup, it is also possible for an app review not to be assigned any of the three classes. For that purpose, in Figure 3(a) we show a *non-informative* node that captures all such cases.

For hierarchical classification, as shown in Figure 3(b), we use a top-down approach for training and classification purpose. At the first level, we are using a binary classifier that classifies all app reviews as either *informative* or *non-informative*, and on the second level we use three one-vs-the-rest binary classifiers that attempt to further classify what passes as *informative* under one or none of the three classes: bug report, feature request, and user experience. Thus, in hierarchical classification, we are training one more classifier than flat classification. This may seem as added complexity, however, the top down approach actually has a better overall computational cost because only the *informative* classifier is trained on all the training examples, the remaining three classifiers train only on the *informative* subset. For example, if we had a training data set of 10k app reviews, 3k of those are informative, then the first level classifier will train on all 10k app reviews, but the second level will only have to train on the 3k app reviews. Whereas, in flat classification, each of the classifiers would need to be trained on the complete 10k dataset.

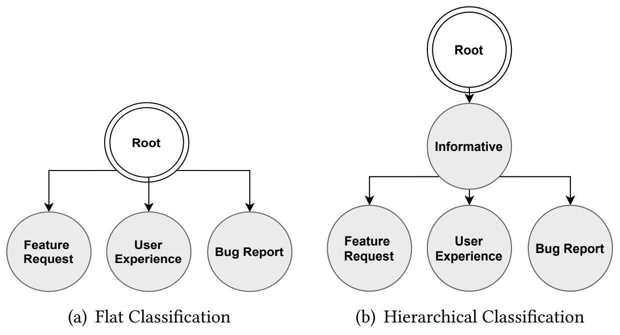
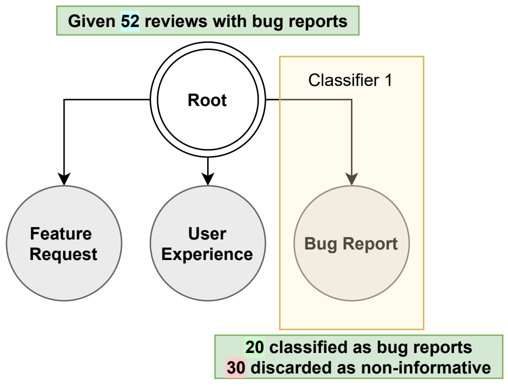


Figure 3: Evaluation of flat and hierarchical app review classification, where each node is a potential label

Experiment Results: We report the average AUC computed from precision and recall (AUC_{PR}), macro F1 (MF_1), and macro recall (MR) in Table 3. We can make a couple of observations. First, Naive Bayes seems to outperform the other classifiers when a simple bag of words model is used, which was also observed in a previous study [25], because a term count representation aligns perfectly with how Naive Bayes works. Second, overall, formulating the problem using hierarchical classification increases the model's accuracy, especially with recall (*i.e.,* increasing the chance that we do not miss any informative app reviews). On the Maalej dataset, we observed on average a 8.4% better AUC_{PR}, 49.8% better F1 measure, and 108% better recall. Similarly on the Panichella dataset we observed 13% better AUC_{PR}, 17% better F1, and 33% better recall. To better understand the results, we analyzed the performance of Random Forest on the Panichella dataset where the recall had an improvement of 61%. It's important to mention that in app review classification, the ability to label all existing *informative* reviews correctly (*i.e.,* recall) is more important than mis-classifying a few *non-informative* reviews as *informative* (*i.e.,* precision) because all reviews labelled as *non-informative* are usually disregarded (*i.e.,* feedback would be lost with low recall). Thus, this significant improvement on the recall when using a hierarchical approach is a perfect match with the app review classification problem.

Table 3: Classification results of flat and hierarchical app review classifiers

Classifier	Maalej Dataset						Panichella Dataset					
	Flat			Hierarchical			Flat			Hierarchical		
	AUC_{PR}	MF_1	MR	AUC_{PR}	MF_1	MR	AUC_{PR}	MF_1	MR	AUC_{PR}	MF_1	MR
Logistic Reg.	0.349	0.369	0.381	0.393 (+12%)	0.433 (+17%)	0.562 (**+47%**)	0.622	0.599	0.594	0.699 (+12%)	0.681 (+13%)	0.731 (**+23%**)
Random Forest	0.399	0.195	0.136	0.433 (+8%)	**0.531** (**+172%**)	0.603 (**+343%**)	**0.739**	0.541	0.428	0.768 (+4%)	0.699 (**+29%**)	0.692 (**+61%**)
SVM	0.346	0.358	0.385	0.353 (+2%)	0.423 (+18%)	0.561 (**+45%**)	0.482	0.523	0.572	0.625 (**+30%**)	0.617 (+17%)	0.701 (**+22%**)
Naive Bayes	0.458	**0.474**	**0.529**	0.497 (+8%)	0.507 (+7%)	**0.623** (+17%)	0.681	**0.630**	**0.624**	0.768 (+13%)	**0.705** (+10%)	0.736 (+17%)
RVM	**0.459**	0.375	0.309	**0.514** (+12%)	0.505 (**+35%**)	0.591 (**+91%**)	0.686	0.591	0.512	0.734 (+7%)	0.702 (+18%)	**0.747** (**+45%**)


Given 52 reviews with bug reports
Root
Classifier 1
Feature Request
User Experience
Bug Report
20 classified as bug reports
30 discarded as non-informative
(a) Flat Classification

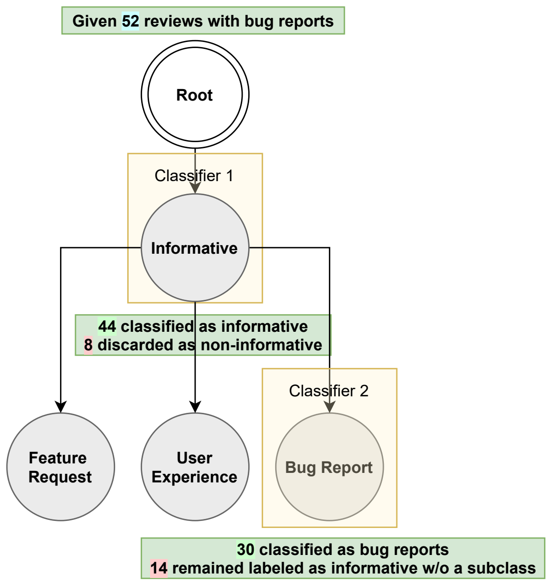

Given 52 reviews with bug reports
Root
Classifier 1
Informative
44 classified as informative
8 discarded as non-informative
Classifier 2
Feature Request
User Experience
Bug Report
30 classified as bug reports
14 remained labeled as informative w/o a subclass
(b) Hierarchical Classification

Figure 4: Given 52 app reviews with bug reports, how were they classified in flat vs hierarchical?

We report in Table 4 the recall of each classifier. In flat classification, we can observe that the classifier's ability to correctly classify all the *bug report* and *user experience* instances is quite poor. As we believe the *bug report* is a more critical category, we further investigated the instances and how they were labelled in both classifiers as shown in Figure 4. In our experiment, the testing sample had 52 app reviews with bug reports. In the case of flat classification, we clearly observed that the classifier missed 32 of the bug reports (62%). However, the hierarchical classifier mislabelled 8 bug reports out of the 52 as *non-informative*, and mislabeled 14 bug reports out of the 44 *informative* reviews as *other* type of informative reviews. Overall, the classifier mislabelled 42% of the bug reports, a much better recall than the flat classifier. Upon further checking, we can observe that the first level performance in the hierarchical classifier is excellent as we were able to capture 85% of the bug reports as informative reviews. However, the second level performance was less ideal (*i.e.,* missing 14 out of 44), but we can argue that it is still better than the flat classifier as we were still able to label those app reviews as *informative*, *i.e.,* they were not completely missed, but were incorrectly classified as *other* types of informative reviews.

Table 4: Analyzing random forest: the flat vs. the hierarchical classifiers on the Panichella dataset

Type	Info.	Feature Request	Bug Report	User Experience
	Recall Measure			
Flat	0.517	0.666	0.385	0.145
Hierarchical	0.727	0.831	0.682	0.526

We credit the better performance of the hierarchical classifier to two main factors. First, it is not affected as much by the class imbalance as the flat classifier. In the case of flat classification, the frequency of each class is dominated by the negative class, *e.g.,* the *bug report* classifier had 91% instances of the negative class so it needs to distinguish from the *non-informative* and other *informative* classes, which is quite challenging. However, in hierarchical classification, the first level uses the combined knowledge from all three classes to first filter out *informative* from *non-informative* app reviews, which is an easier task, *i.e.,* due to the different nature of *non-informative* reviews from *informative* along with a much higher positive class frequency. Second, we observed that the *bug report* classifier can distinguish itself better from other *feature request* and *user experience* reviews (*i.e.,* informative reviews) when *non-informative* reviews are removed, which is what the hierarchical top-down classification is inherently doing.

RQ$_2$: How accurate is the classification of the proposed HMK-RVM approach compared to the state of the art?

Baselines: To evaluate our proposed approach, we compared against five baselines and using two different datasets. The proposed approach and all the baselines presented are trained using the textual content of the reviews and the meta data information.

First, the **AR-Miner** [11] baseline used a Naive Bayes model [30], where the hidden topics of the reviews were discovered using Latent Dirichlet Allocation (LDA) [7] and used alongside the rating of the app review to construct the feature space. To implement their approach, we selected the number of topics k for LDA using cross-validation. Specifically, we chose 85 topics for both datasets.

Second, the **Maalej** [25] baseline also adopted a Naive Bayes model due to its previously reported high performance with text classification. However, [25] used a bag of words approach and extracted the ratio of past, present, and future tenses in the review to represent the textual content, claiming that reviews with bug reports tend to use past tenses, whereas reviews with feature requests tend to use future tenses. Additionally, they used the review's rating, length, and sentiment score as part of their features.

Third, the **ARdoc** [33, 34] baseline leveraged a decision tree (J48) model. The authors manually constructed 246 linguistic patterns each mapping to a specific app review label, *e.g.,* reviews with pattern *[someone] should add [something]* are mapped to *feature requests.* Moreover, they generated a TF-IDF representation from the textual content of the reviews and used the review's sentiment score in their feature space. Due to the difficulties in recreating the 246 linguistic patterns, we did not implement this approach ourselves but rather used the tool provided by the authors to generate labels. As such, we do not have the AUC and ROC scores for this baseline, since computing them requires access to the model itself to evaluate performance under different decision thresholds.

Finally, we include the proposed approach with two variant baselines. The **RVM** baseline, where the Relevance Vector Machines (RVM) [42] with fast marginal likelihood maximization is used as a baseline using our complete feature space (features are concatenated into a single large feature space) and presented as an alternative to using the multi-kernel learning approach. The (**MK-RVM**) baseline where the multi-kernel approach is added to the previous baseline and the problem is approached using a flat classification approach as an alternative to using a hierarchical approach. Finally, (**HMK-RVM**), which is our proposed model, the hierarchical multi-kernel RVM which combines the power of RVM with multi-kernel learning and follows a hierarchical classification approach that leverages the existing hierarchical structure.

Experiment Setup: We formulated the learning task as a binary one-vs-the-rest problem by following earlier work. We chose this formulation due to two reasons: the first is that it supports multi-label classification (*e.g.,* a review can contain both a bug report and a feature request), and the second is due to its higher reported performance than multi-class classification. For example, [25] reported that using multiple binary classifiers for app review classification performed significantly better than a single multi-class classifier in all cases. To measure the accuracy of the models, we used a train/test split of 80/20. To measure the robustness, *i.e.,* performance on different datasets, we conducted the experiment on both *Maalej* and *Panichella* datasets.

Experiment Results: In Table 5, we show a summary of the results. We can observe that the traditional RVM that uses our proposed feature space performs on par with the other baselines introduced in prior work. This highlights the usefulness of leveraging the information from multiple aspects and shows that RVM is on equal footing to other models such as Naive Bayes and Decision Trees in terms of accuracy. Moreover, it shows that using a larger feature space on its own is not enough to gain a competitive advantage as the difference between it and other baselines is not that significant. Once we utilize the multi-kernel approach we can observe a 2%-4%improvement in the overall model's performance (AUC_{PR}) over traditional RVM on both datasets, and a 3%-15% improvement with the proposed hierarchical version of the multi-kernel RVM classifier. We can also observe that most of this improvement is due to a boost in the recall (93% increase on Maalej and 40% on Panichella). As we discussed earlier, this is the main advantage of leveraging the existing hierarchical relationship between labels. Breaking the prediction task into multiple levels, whereby in the first, we predict (informative vs. non-informative) and use the collective knowledge between the different children of each branch can significantly boost the model's recall, *i.e.,* increases our chance of identifying informative reviews correctly.

Overall, we can observe that the proposed hierarchical multi-kernel RVM is outperforming all the baselines as it can offer a boost through the combination of two aspects. First, the multi-kernel learning technique allows it to choose the best kernel(s) for the current learning task through the assigned weights (*e.g.,*

Table 5: Summary of the results comparing proposed approach to the start of the art

Approach	Maalej Dataset						Panichella Dataset					
	AUC$_{PR}$	AUC$_{ROC}$	mF_1	MF_1	MP	MR	AUC$_{PR}$	AUC$_{ROC}$	mF_1	MF_1	MP	MR
AR-miner [11]	0.402	0.804	0.496	0.445	0.363	**0.634**	0.432	0.806	0.472	0.444	0.345	0.699
Maalej [25]	0.472	0.843	0.565	0.513	0.463	0.597	0.668	0.898	0.677	0.640	0.645	0.647
ARdoc [33, 34]	-	-	0.338	0.267	0.341	0.325	-	-	0.376	0.307	0.642	0.344
RVM	0.506	0.869	0.433	0.399	**0.583**	0.308	0.702	0.927	0.655	0.617	0.736	0.536
MK-RVM	0.516	**0.870**	0.421	0.377	0.502	0.304	0.729	**0.930**	0.685	0.652	**0.741**	0.567
HMK-RVM	**0.519**	0.798	**0.615**	**0.541**	0.503	0.594	**0.806**	0.882	**0.771**	**0.729**	0.709	**0.753**

meta-information might be more useful to bug reports than feature requests, leading to a higher weight for the corresponding kernel than other kernels, or LDA topics may introduce more noise than true signals for bug reports, hence setting the LDA kernel's weight to very small can improve the model's accuracy). Second, the hierarchical approach offers a boost in the model's recall through leveraging existing hierarchical relationships between the different labels. Moreover, through the learning process, the proposed approach has identified, on average, 45 relevant vectors (varies by classifier/dataset). Those relevant vectors should be the most representative reviews (*i.e.,* reviews that best summarize the content), which provides us with two additional advantages beyond accuracy. The first is a computational advantage, as we can limit future training and prediction to those relevant vectors since other points are already represented by them, which significantly cuts down the original dataset size. The second is a summarization advantage, as those reviews should highlight the reviews that best summarize the dataset, which developers can use for requirement extraction.

RQ$_3$: How accurate is the summarization of the proposed HMK-RVM approach compared to the state of the art?

Building on the classification step, which helped us identify the set of informative reviews and filter out the non-informative ones, the next goal is to summarize the feedback in the set of informative reviews for the purpose of requirement extraction. We propose to leverage the set of relevant vectors, which HMK-RVM learns as part of the classification task as a way to potentially summarize the users' feedback. As a result, we achieve both the classification and summarization tasks simultaneously using the same model. In this section, we will evaluate the set of reviews identified as the most informative by HMK-RVM for requirement extraction against multiple baselines that were used in the literature for this purpose or for summarization in general.

Baselines: We will use the set of relevant vectors identified by the **HMK-RVM** model presented in RQ2 as our proposed approach and compare it to the following baselines:

First, we will compare against approaches that were proposed by prior work. We will build upon the classification experiment to further summarize the content of the reviews based on the recommendation of the original authors. For **AR-Miner** [11] and **ARdoc** [33, 34], *Latent Dirichlet Allocation* (LDA) will be used to group the set of reviews predicted as *informative*, and then the review with the highest probability for each topic will be picked as the most informative one. The size of the final list of selected reviews will be equal to the number of topics. As for **Maalej** [25], where the original authors did not propose any summarization approach, we will apply *K-means* to the set of reviews classified as *informative* to cluster them, and then use the review at the center of each cluster as the most informative review. We will also compare against **Star Clustering** [1], which creates a graph where each node is a review and an edge is created if the cosine similarity between two reviews is larger than a given alpha, and then use the set of nodes with the highest degree to be the set of *center stars*, *i.e.,* most informative reviews for requirement extraction.

Second, for the purpose of completeness, we will use random sampling as a baseline where we randomly picked *n* points as the set of most informative reviews. Additionally, we will compare against widely used summarization techniques such as K-means and Latent Dirichlet Allocation (LDA) in the same way described earlier but applied to the complete dataset.

To keep this comparison fair, we made the selection of the hyperparameters, *e.g.,* number of topics for LDA, in a manner that provided us with a final set of informative reviews that is equal in size for all baselines.

Experimental Setup: To evaluate this aspect of the proposed HMK-RVM approach, we asked two graduate Ph.D. students (in computing) to read the reviews in the Panichella dataset and generate a list of the requirements discussed, and then label each review with 1) a requirement id(s), 2) a level of informativeness ranging from one to three, where *one* is a review with no requirements, *two* is a review that is relevant to a requirement but without enough information to extract it (*e.g.,* due to missing info, poor readability, or not being explicit enough, *i.e.,* requiring the developer to guess/infer the meaning), and *three* is a review with an explicit requirement and enough information to extract it. We show examples of this labeling in Table 6. The two students annotated the dataset separately and then compared their labels. Disagreements were resolved in

Table 6: Examples of real-world reviews from the Panichella dataset and how they were labeled for RQ3. Requirement ID 5 refers to users' request for additional login options.

Review	Req. Id	Informative Level
Blinq Okay	NA	1 (Low)
Login Facebook? Nope. App immediately deleted	5	2 (Medium)
FB and without FB can Blinq not work?? There must also be an alternative logon options!	5	3 (High)

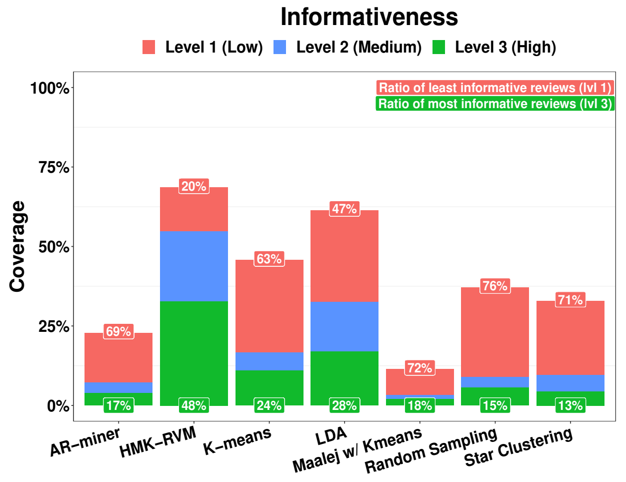


Figure 5: The results of the summarization evaluation. The Y-axis represents the coverage, *i.e.,* percentage of requirements that were captured by the selected set of informative reviews. The higher the score the better. The color provides a visual representation of how informative are the reviews selected by each approach. The more green and the less red, the better the approach.

group discussions. More details on this process can be found in the replication package[1]. We found a substantial inter-rater agreement (kappa=0.87) between the annotators. It is important to know that the reviews are sharing the same context (same app, same version) to assume that they are discussing the same requirement, which is why we only used the Panichella dataset for this evaluation as it provides the app information in addition to the review, whereas, this information is missing in the Maalej dataset.

We use two metrics to evaluate each approach. First, how *informative* are the selected reviews for requirement extraction, *i.e.,* were the selected reviews mostly of level two and three of informativeness (medium and high), or were they mostly level one (noise). Second, as part of the labeling process we compiled a list of requirements that are discussed in the reviews, and using this ground truth, we want to evaluate the *coverage* of each approach, *i.e.,* how many of the existing discussed requirements were mentioned in the selected set. However, we argue that not all requirements are equal. The more mentioned/discussed a requirement is, the more valuable, and vice versa. As such, we measured coverage only for requirements mentioned in three or more reviews. As most machine learning models require a certain level of statistical presence to learn patterns, two may not be sufficient to show the statistical significance. Meanwhile, setting a higher threshold (*e.g.,* four or more) may miss some meaningful requirements.

Experimental Results: We show the evaluation results in Figure 5. We can observe that the proposed HMK-RVM significantly outperforms all the baselines. First, looking at *coverage* where the higher the score the better the model at capturing all the discussed requirements, we can see that it is 11% better than LDA (the second best model) and roughly 50% better than all other baselines. This means it is able to select at least one review for each discussed requirement with a much higher success rate than the state of the art. Second, looking at informativeness, which is a key aspect of requirement extraction, we can see that the reviews selected by HMK-RVM have the highest level of informativeness, and the least level of noise. HMK-RVM had 74% more informative reviews than the second-best baseline. Additionally, it picked 50% less noisy reviews than the second-best baseline. This means that it is far superior at picking the most informative reviews and avoiding the least informative (noisy) reviews for requirements extraction than the state of the art.

Table 7: Analyzing the model's insight: What and the learned weights tell us about the underlying data?

Maalej Dataset	$\phi_{we}(x)$	$\phi_{meta}(x)$	$\phi_{tfidf}(x)$	$\phi_{lda}(x)$
Informative	0.496	**0.533**	0.474	**0.511**
Feature Request	0.489	0.504	**0.522**	0.521
Bug Report	0.512	0.475	**0.513**	0.512
User Experience	**0.505**	0.383	**0.001**	**0.892**

RQ$_4$: Beyond accuracy, what insights can we gain from using the proposed HMK-RVM approach?

Experimental Setup: To address this question, we evaluated the weights assigned to each of the kernels.

Experimental Results: For the first aspect, we report the assigned weights per kernel in Table 7 for the Maleej dataset. We can observe that the learned weights per kernel vary between roughly 12% on average, which indicates a different priority based on the learned task. For example, for the *informative* classifier, kernels with higher representation (*i.e.,* Meta and LDA) were assigned higher weights, which can be due to the fact that the majority of reviews at the first level of classification are *non-informative* reviews (70%), mostly rating reviews (*i.e.,* a strong positive or negative rating with a short sentimental text). As such, they can be easily identified with a more broad view of the reviews. Also, for the *feature request* and *bug report* classifiers, we can observe a higher assigned weight to the TF-IDF kernel, which may be due to the fact that such reviews can be identified through a few frequently used words that are captured by TF-IDF (*e.g.,* add, feature, bug, crash, *etc.*). Finally, the *user experience* classifier shows a significant weight difference between kernels. The LDA kernel and word embedding kernel are highly utilized, whereas the TF-IDF kernel is essentially ignored. We believe that this is due to the rich and lengthy nature of such reviews (reviews with user experience are very descriptive). Having that nature in mind with the fact that app reviews are usually full of typos and alternatively spelled words would put a representation that relies on exact terms such as TF-IDF at a disadvantage, whereas a semantic capturing representation such as word embedding or a topic capturing representation such as LDA is at a clear advantage. As such, we can conclude that the *user experience* classifier's predictions rely heavily on the LDA and word embedding representation, *i.e.,* in order to maintain a healthy *user experience* classifier, we need to maintain those representations. Such insight into the classifier's learning patterns is valuable to the understanding and interpretation of the classifier's behavior.

5 DISCUSSION

How is the proposed approach different from the current state of the art (SOTA)? The existing SOTA approaches use a pipeline of two dedicated models, one for classification and another for summarization. This allows them to fine-tune each model for its specific task. However, this also complicates the process of implementation and maintainability. In contrast, our proposed approach is designed to achieve both the classification and summarization tasks using a single model. Although we do not have the option to fine-tune the results for each task, we still demonstrated that we were able to provide equal or better results on one task (*i.e.,* classification) and outperform all baselines on the other (*i.e.,* summarization), which shows that we did not compromise on the accuracy when we attempted to merge the two tasks. In fact, the summarization aspect is the most important aspect for requirements extraction, in which our approach outperforms all baselines in by a large margin.

How does the proposed approach improve the extraction of requirements? How is this tested? The key improvement lies in the amount of effort that the proposed approach can reduce in terms of the human effort for requirements extraction. To measure this aspect, we introduced two metrics, coverage and informative level. For the former, a high coverage implies that analyzing the model-identified subset of reviews would allow the developer to extract most requirements. The saving of effort is achieved as the rest of the reviews can be safely ignored. As for the latter, reviews with a higher level of informativeness can help developers more easily and accurately extract the requirements without cross-checking other reviews. In our experiment, the set of informative reviews constitutes around 35%-40% of the entire dataset whereas the set of representative reviews that HMK-RVM identified includes only around 5% of the dataset. This implies that by using HMK-RVM, we can effectively reduce the human effort needed to extract the requirements from manually analyzing 35%-40% of the dataset to only 5%. In addition, the reviews identified by HMK-RVM are of a high level of informativeness, which can improve the easiness and accuracy of requirement extraction from these reviews.

What are the limitations of the approach? One limitation is that RVM tends to pick from highly representative regions as a result of maximizing the model evidence in Eq. 4. While it is highly desirable to choose a small number of reviews to represent the whole set, it may also miss some requirements from less representative regions. Our results show that HMK-RVM achieved a 70% coverage by just using a small number of representative reviews, which clearly demonstrates its effectiveness. An interesting future direction is to augment RVM's learning process to include a few reviews from less representative regions to enhance the coverage further. Another limitation is that as we go down the hierarchy, we are expected to have less data which may affect the performance. Thus, another interesting direction is to study the effect of adding more hierarchical levels on the performance of the model.

6 THREATS TO VALIDITY

In terms of **internal** validity, the main threat is that we used two datasets in our experiment coming from previous work. We did not participate in the collection or preparation of those datasets. Thus, any issues with the reviews content or the labels are a potential risk factor. The *Maalej dataset* provided both the reviews and the labels, whereas, the *Panichella dataset* provided only the reviews. As such, we had to manually label the reviews ourselves for the *Panichella dataset*. In both cases, whether the ground truth was handed to us, or whether we manually labeled the reviews, there is the risk of human coders mistakes. To reduce this threat to our labels, we created a coding guide that precisely defines the app review types with an example of each, and we employed two teams each with two members to label the dataset separately. Once both teams completed their task, we sat down and extensively discussed any disagreements. In terms of **external** validity, we believe our results should have high generalizability for app reviews as we evaluated it on two different real-world datasets that were carefully constructed, i.e., sampled randomly from different apps and app stores. As such, they should provide a reasonable approximation of the general population.

7 CONCLUSION

In this paper, we proposed Hierarchical Multi-Kernel RVM (HMK-RVM) where we extended and customized the use of RVM in a novel way to facilitate requirement extraction from app reviews by offering an integrated process that is easier to implement, interpret, and maintain. The proposed approach classifies reviews in a hierarchical fashion, leading to a more accurate model. In addition, we showed that the assigned weights to each kernel can provide an insight into what the classifier has learned from the underlying data. Moreover, we leveraged RVM's inner working mechanism to accomplish the summarization task as part of the classification learning process, and we have demonstrated its ability to outperform the state of the art in terms of summarization accuracy while achieving a competitive classification accuracy.

ACKNOWLEDGMENTS

This research was supported in part by an NSF IIS award IIS-1814450 and an ONR award N00014-18-1- 2875. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agency.

REFERENCES

[1] Javed A. Aslam, Katya Pelekhov, and Daniela Rus. 1998. Static and Dynamic Information Organization with Star Clusters. In *CIKM*. ACM, 208–217.

[2] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. 2014. Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR*. ACM, 112–121.

[3] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. 2014. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering* 19, 3 (2014), 619–654.

[4] Andrew Begel and Thomas Zimmermann. 2014. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering, ICSE*. ACM, 12–23.

[5] Dimitri P Bertsekas. 1997. Nonlinear programming. *Journal of the Operational Research Society* 48, 3 (1997), 334–334.

[6] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[7] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.

[9] Laura V. Galvis Carreño and Kristina Winbladh. 2013. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 35th International Conference on Software Engineering, ICSE*. IEEE Computer Society, 582–591.

[10] Eya Ben Charrada. 2016. Which One to Read? Factors Influencing the Usefulness of Online Reviews for RE. In *Proceedings of the 24th IEEE International Requirements Engineering Conference, RE*. IEEE Computer Society, 46–52.

[11] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering, ICSE*. ACM, 767–778.

[12] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason I. Hong, and Norman M. Sadeh. 2013. Why people hate your app: making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*. ACM, 1276–1284.

[13] Cuiyun Gao, Jichuan Zeng, David Lo, Chin-Yew Lin, Michael R. Lyu, and Irwin King. 2018. INFAR: insight extraction from app reviews. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 904–907.

[14] Giovanni Grano, Andrea Di Sorbo, Francesco Mercaldo, Corrado Aaron Visaggio, Gerardo Canfora, and Sebastiano Panichella. 2017. Android apps and user feedback: a dataset for software evolution and quality improvement. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics, WAMA@ESEC/SIGSOFT FSE 2017, Paderborn, Germany, September 5, 2017*. ACM, 8–11.

[15] Xiaodong Gu and Sunghun Kim. 2015. "What Parts of Your Apps are Loved by Users?" (T). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, ASE*. IEEE Computer Society, 760–770.

[16] Emitza Guzman and Walid Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *Proceedings of the IEEE 22nd International Requirements Engineering Conference, RE*, Tony Gorschek and Robyn R. Lutz (Eds.). IEEE Computer Society, 153–162.

[17] Elizabeth Ha and David A. Wagner. 2013. Do Android users write about electric sheep? Examining consumer reviews in Google Play. In *Proceedings of the 10th IEEE Consumer Communications and Networking Conference, CCNC*. IEEE, 149–157.

[18] Mark Harman, Yue Jia, and Yuanyuan Zhang. 2012. App store mining and analysis: MSR for app stores. In *Proceedings of the 9th IEEE Working Conference of Mining Software Repositories, MSR*. IEEE Computer Society, 108–111.

[19] Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*. ACM, 168–177.

[20] Claudia Iacob and Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR*. IEEE Computer Society, 41–44.

[21] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E. Hassan. 2015. What Do Mobile App Users Complain About? *IEEE Software* 32, 3 (2015), 70–77.

[22] Daphne Koller and Mehran Sahami. 1997. Hierarchically Classifying Documents Using Very Few Words. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML*. Morgan Kaufmann, 170–178.

[23] Johannes V. Lochter, Pedro R. Pires, Carlos Bossolani, Akebo Yamakami, and Tiago A. Almeida. 2018. Evaluating the impact of corpora used to train distributed text representation models for noisy and short texts. In *Proceedings of the 2018 International Joint Conference on Neural Networks, IJCNN*. IEEE, 1–8.

[24] Walid Maalej, Zijad Kurtanovic, Hadeer Nabil, and Christoph Stanik. 2016. On the automatic classification of app reviews. *Requir. Eng.* 21, 3 (2016), 311–331.

[25] Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *Proceedings of the 23rd IEEE International Requirements Engineering Conference, RE*. IEEE Computer Society, 116–125.

[26] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*.

[27] Stuart McIlroy, Weiyi Shang, Nasir Ali, and Ahmed E. Hassan. 2017. User reviews of top mobile apps in Apple and Google app stores. *Commun. ACM* 60, 11 (2017), 62–67.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR (Workshop Poster)*.

[29] Azad Naik and Huzefa Rangwala. 2018. *Large Scale Hierarchical Classification: State of the Art*. Springer.

[30] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom M. Mitchell. 2000. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning* 39, 2/3 (2000), 103–134.

[31] Dennis Pagano and Walid Maalej. 2013. User feedback in the appstore: An empirical study. In *Proceedings of the 21st IEEE International Requirements Engineering Conference, RE*. IEEE Computer Society, 125–134.

[32] Fabio Palomba, Pasquale Salza, Adelina Ciurumelea, Sebastiano Panichella, Harald C. Gall, Filomena Ferrucci, and Andrea De Lucia. 2017. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th International Conference on Software Engineering,ICSE*. IEEE / ACM, 106–117.

[33] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution, ICSME*. IEEE Computer Society, 281–290.

[34] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. ARdoc: app reviews development oriented classifier. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE*. ACM, 1023–1027.

[35] Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. 2015. Leveraging User Reviews to Improve Accuracy for Mobile App Retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*. ACM, 533–542.

[36] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*. ACL, 1532–1543.

[37] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

[38] Christoffer Rosen and Emad Shihab. 2016. What are mobile developers asking about? A large scale study using stack overflow. *Empirical Software Engineering* 21, 3 (2016), 1192–1223.

[39] Dwaipayan Roy, Debasis Ganguly, Sumit Bhatia, Srikanta Bedathur, and Mandar Mitra. 2018. Using Word Embeddings for Information Retrieval: How Collection and Term Normalization Choices Affect Performance. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM*. ACM, 1835–1838.

[40] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. 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*. ACM, 499–510.

[41] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Corrado Aaron Visaggio, and Gerardo Canfora. 2017. SURF: summarizer of user reviews feedback. In *Proceedings of the 39th International Conference on Software Engineering, ICSE*. IEEE Computer Society, 55–58.

[42] Michael E. Tipping. 1999. The Relevance Vector Machine. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. The MIT Press, 652–658.

[43] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2016. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering, ICSE*. ACM, 14–24.

[44] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. 2015. Mining User Opinions in Mobile App Reviews: A Keyword-Based Approach (T). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, ASE*. IEEE Computer Society, 749–759.

[45] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. 2016. Phrase-based extraction of user opinions in mobile app reviews. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE*. ACM, 726–731.