

Article

---

# A Phrase-Level User Requests Mining Approach in Mobile Application Reviews: Concept, Framework, and Operation

---

Cheng Yang, Lingang Wu, Chunyang Yu and Yuliang Zhou



## Article

# A Phrase-Level User Requests Mining Approach in Mobile Application Reviews: Concept, Framework, and Operation

Cheng Yang <sup>1</sup>, Lingang Wu <sup>2,\*</sup>, Chunyang Yu <sup>1</sup> and Yuliang Zhou <sup>2</sup>

<sup>1</sup> College of Creativity and Art Design, Zhejiang University City College, Hangzhou 310015, China; yangchengyc@126.com (C.Y.); yucy@zucc.edu.cn (C.Y.)

<sup>2</sup> College of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China; zhouyuliang123@gmail.com

\* Correspondence: lingang\_wu@zju.edu.cn

**Abstract:** Mobile application (app) reviews are feedback about experiences, requirements, and issues raised after users have used the app. The iteration of an app is driven by bug reports and user requirements analyzed and extracted from app reviews, which is a problem that app designers and developers are committed to solving. However, a great number of app reviews vary in quality and reliability. It is a difficult and time-consuming challenge to analyze app reviews using manual methods. To address this, a novel approach is proposed as an automated method to predict high priority user requests with fourteen extracted features. A semi-automated approach is applied to annotate requirements with high or low priority with the help of app changelogs. Reviews from six apps were retrieved from the Apple App Store to evaluate the feasibility of the approach and interpret the principles. The performance comparison results of the approach greatly exceed the IDEA method, with an average precision of 75.4% and recall of 70.4%. Our approach can be applied to specific app development to assist app developers in quickly locating user requirements and implement app maintenance and evolution.

**Keywords:** app reviews; user requests; request priority; sentiment analysis



**Citation:** Yang, C.; Wu, L.; Yu, C.; Zhou, Y. A Phrase-Level User Requests Mining Approach in Mobile Application Reviews: Concept, Framework, and Operation. *Information* **2021**, *12*, 177. <https://doi.org/10.3390/info12050177>

Academic Editor: Spina Damiano

Received: 28 February 2021

Accepted: 16 April 2021

Published: 21 April 2021

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The growth of the app market has been boosted by the maturity of smartphones, allowing users to conveniently browse and download apps from app stores (e.g., Apple App Store and Google Play) and leave reviews for apps they have used, including star ratings and text-based feedback. Many studies have proven that app reviews, which contain problem feedback, feature requests, and other suggestions, can be regarded as references for the iterative design and development of the app [1–3]. Rapid iteration, which is one of the main factors for the success of an app's development [4], mainly includes bug fixing, feature modification, and the addition of new features. Hence, by analyzing user data from the new versions, iteration strategies can be conducted by app designers and developers. The timely and accurate gathering of information revealed by user reviews can help developers maintain and update their apps and achieve effective word-of-mouth marketing [5,6].

Unfortunately, analyzing app reviews is a challenge, especially through manual methods. First, there are a great number of app reviews. For some popular apps, each version is appended with hundreds or thousands of reviews [7]. Second, the quality of reviews vary widely, and some are simply emotional evaluations (e.g., “Great!”) that are not valuable for the development of apps. Third, the language expression of reviews is relatively informal, containing lots of noise, such as misspellings, casual grammatical structures, and non-English words [8]. To address these issues, there are many studies dedicated to automatically filtering out non-informative reviews [9], categorizing reviews

and user requirements [10], and obtaining valuable topics from massive reviews [11] for the purpose of app maintenance and evolution.

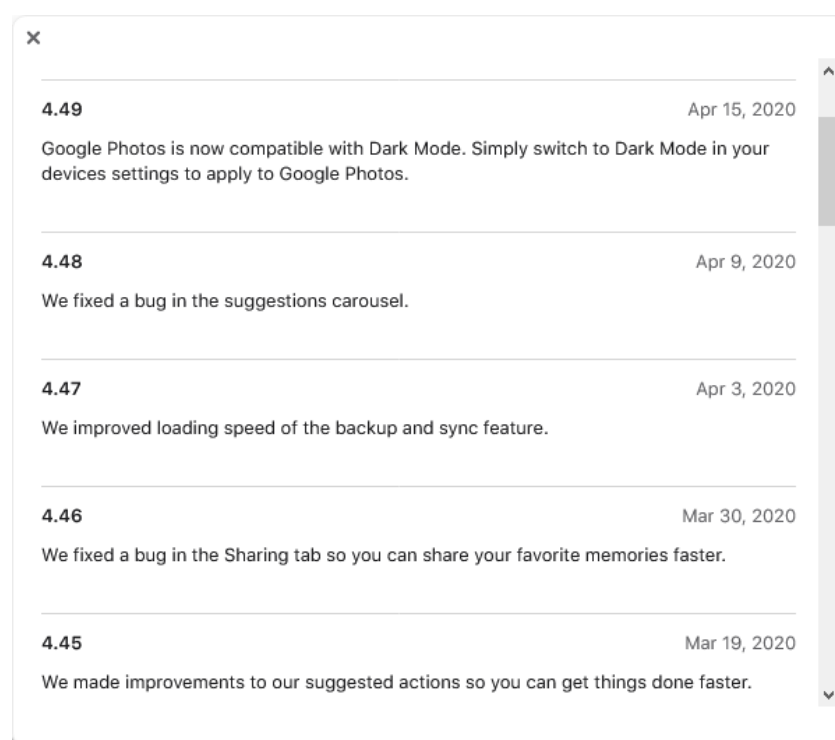
These studies automatically extract useful information for developers (e.g., bug reports and feature requests). Developers can quickly bug test and implement fixes, but for user requirements, whether to implement user suggestions or not is not simply a subjective decision. In other words, the studies above can only extract user request topics or sentences from the reviews, but cannot speak on to what extent the requests are worthy of improving or adding, or which feature requests have a high priority to implement. For example, the review “I wish we had the option of making our own stylized photos though” can be classified as a feature request, and the topics can be extracted as “make”, “stylized”, and “photo”, which means the request of “editing photos”; however, whether the request should be implemented in the next few releases is hard to say. Developers will consider the questions “how many users proposed such requests”, “how the users think about the present functionalities”, etc. Recently, a few studies address this question, such as Nayebe and Ruhe [12], who proposed a bi-criterion integer programming model to select optimized app functionalities based on feature value (e.g., rating) and cost (e.g., effort to implement). However, the study focuses mainly on the trade-off solution for the functionalities and neglect requirements discussed in reviews based on user feedback.

In this paper, we focus our attention on the user requirements in reviews and propose a novel approach to (1) extract requirement phrases for the app functionalities; (2) calculate features such as the occurrence frequency and rating for the requirement phrases; (3) automatically predict those requirement phrases with high priority to be implemented based on the features extracted for the phrases.

A total of 44,893 real-world reviews from six apps on the Apple App Store were collected to verify the feasibility of the approach. The results indicate that the optimal model can reach an average accuracy, precision, recall, F-Measure, and ROC\_AUC of 67.6%, 67.3%, 69.2%, 68.0% and 71.4%, respectively, after optimization. To annotate the true priority of the requirement phrases, a half-automated method is proposed to link the requirement phrases with app changelogs and the requirement phrases linked successfully to changelogs are annotated with high priority. The app changelog (the list of changes in each release) is a short introduction presented in the App Store that is written by the publisher and describes the issues that were addressed and the new features in the latest version. The purpose of a changelog is encourage users to update and experience the new version. We conjecture that the requirement phrases mentioned in changelogs can be considered as high priority requirements. Figure 1 illustrates an example of a changelog history for Google Photos in the Apple App Store.

The main contributions of this study are as follows. (1) A novel framework is implemented to automatically extract and predict high priority requirement phrases from app reviews with fourteen features calculated for the phrases. (2) A novel method is proposed to semi-automatically annotate requirement phrases with high or low priority with help of app changelogs and the effectiveness is verified. (3) An empirical study was designed and performed to examine the effectiveness, interpretation, and comparison of the novel approach in terms of high priority requirement mining.

The rest of the paper is structured as follows. Section 2 introduces the related work. Section 3 explains the details of the framework of the approach. Section 4 describes the main research questions and the method of evaluation for our approach. Section 5 presents the results and discussions. Section 6 discusses the threats to validity, and Section 7 concludes the paper.



**Figure 1.** A screenshot of a part of the changelog history for Google Photos.

## 2. Related Work

In this section, we introduce the related work in three main categories: app review classification, requirement ranking, and aspect extraction from reviews.

### 2.1. App Review Classification

Classifying app reviews into different categories is quite a hot issue in app review analysis research, and many studies consider classification as a preprocessing phase [9,13,14], and then further analysis can be performed for specific categories of reviews. Panichella et al. [15] combined the methods of natural language processing, text analysis, and sentiment analysis to extract the reviews of verifiability for app maintenance and updates and classified reviews into four categories—information seeking, information giving, feature request, and problem discovery. Maalej et al. [16] proposed a classification model to categorize reviews into bug reports, feature requests, user experiences, and simple ratings. McIlroy et al. [17] analyzed the types of app reviews and proposed an automated method to assign multiple labels to reviews since one review might have multiple types. Guzman et al. [18] compared the performance using individual machine learning methods and their ensembles for automatically classifying reviews and found the ensembles can reach a better result. Jha and Mahmoud [19] focused on the non-functional requirements (e.g., security and performance) mining and used classification methods to capture those requirements in reviews. These studies can filter out some noise of app reviews and provide developers with specific types of reviews. However, the developers still have to deal with plenty of reviews in each category since the priority of the reviews is not considered.

The most obvious difference between the aforementioned studies and ours is that instead of selecting reviews classified as one certain category, our approach deals with all reviews. except for duplications, and extracts phrases that mention app functionality. We assume that both a positive or negative review or a feature request will address the app functionalities to some degree, and the extracted phrases can identify the subject. If a review does not contain such subjects, no phrases will be extracted, so non-informative information can be filtered out during the phrase extraction phase.

## 2.2. Sentiment Analysis for App Reviews

Sentiment analysis can obtain the positive and negative emotions and their intensity in user reviews. Using this method, users' satisfaction or dissatisfaction with functional features can be quantified for future requirement analysis [20]. Although the satisfaction of users can be reflected to some extent from the ratings put forward by the user themselves, sentiment analysis can be specific to a sentence or a word in the review, so as to accurately match the extracted topics. Sentiment analysis has great value and is widely applied in review analysis.

However, sentiment analysis is only used as an initial review processing method to quantify user sentiment and satisfaction, and a series of subsequent complex calculations for specific demand analysis are essential. In terms of the implementation methods of sentiment analysis, different studies apply slightly different methods, but the overall principle is based on sentiment lexical database and lexical association to realize the calculation of sentiment value. For instance, Ranjan and Mishra [21] used TextBolb, Suprayogi et al. [11] applied Sentistrength, and Gu and Kim [13] implemented Deeply Moving.

## 2.3. Requirement Ranking

To rank the significance of requirements or features for apps is another attractive topic for app maintenance and evolution. Nayebi and Ruhe [12] calculated the optimized functionality for a certain app category. Nayebi and Ruhe [22] proposed an asymmetric release planning to maximize satisfaction and minimize dissatisfaction by predicting whether an app feature should be offered or not in the next release. These two studies both use bi-criterion integer programming to solve the trade-off between feature values and costs or satisfaction and dissatisfaction. However, the verification of the proposed models uses app features extracted from app descriptions and how to extract features from user reviews is not considered, so the authors are coming from the perspective of developers. Our approach differs from theirs in mainly two aspects. First, we extract requirements from reviews, thus looking at the issue from the user's perspective. Second, we mainly focus on analyzing and prioritizing user requirements, while not considering the cost to implement the requirements.

Chen et al. [9] proposed AR-Miner for extracting informative reviews and prioritizing reviews by the effective review ranking scheme after grouping reviews. The difference is that we directly extract requirement phrases from reviews so we do not need to filter out non-informative reviews first in which step loss may generate, i.e., the classification step cannot assure one hundred percent accuracy. For the prioritizing step, AR-Miner depends on the volume and rating for the reviews in each group and has flexible weighting parameters to calculate the priority scores, while our approach considers more parameters (e.g., sentiment analysis results) and applies machine learning algorithms to predict the priority of a requirement. In addition, the true set for the ranking scheme of AR-Miner is created by another user feedback forum with a user voting mechanism, while ours is created by the changelogs, so if an app does not have such feedback forums, our method can be useful.

Another model called CLAP, developed by Scalabrino et al. [14], also used machine learning algorithms—random forest was used to predict the priority of review clusters. CLAP first classifies reviews into seven more detailed categories and uses clustering techniques to group similar reviews into clusters in each category. The truth set for priority also uses changelogs, in which, the authors believe if a review is implemented by developers in the next release, the review can be labeled as a high priority. We argue that some important requirements will not be implemented immediately in the next release, i.e., the priority cannot be decided only by the changelogs for the very next release. Usually, changelogs for one release are quite short as shown in Figure 1, as such, we consider the historical changelogs for greater clarity. Another difference is that we focus on the priority of requirement phrases while CLAP focuses on reviews in clusters, so the features used for prediction are totally disparate.

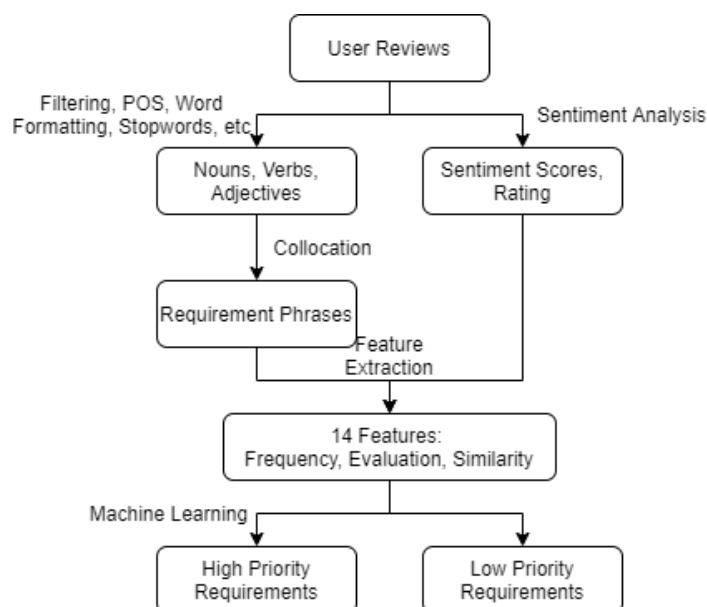
### 2.4. Aspect Extraction from Reviews

Generally, aspect extraction is to extract meaningful keywords from reviews such as words, phrases, or short sentences, which is also a widely investigated area. Chen et al. [9] used topic modeling methods to extract topics for reviews and then grouped reviews with the same topics. Gao et al. [8] proposed an IDEA model based on Online Latent Dirichlet Allocation to identify topics about emerging issues in the current release of apps. Jo and Oh [23] discovered aspects evaluated in reviews with their sentiments by adapting a topic model named ASUM. These approaches all applied topic modeling methods to mining review aspects; however, topics mainly contain a few separated words and can result in confusion so developers can meet the trouble to immediately interpret the topics.

Guzman and Maalej [24] applied a collocation method to extract feature phrases in reviews and analyzed their sentiments. Gu and Kim [13] proposed SUR-Miner to extract aspect–opinion pairs based on the monotonous structure and semantics of reviews. The two studies both extracted aspects at the phrase level and analyzed opinions about the aspects, which has a better interpretation for the extracted information. The aspects extraction method for our approach combines the mentioned studies, but besides sentiment analysis, we analyze more features for extracted aspects (e.g., frequency for aspects and similarity analysis between aspects) and the features are used to predict the priority of aspects; both of the studies did not do this. Additionally, the true set creation for both studies is manually judging the validity of aspects extracted, whereas we consider the changelogs in a more objective way.

### 3. The Framework of the Approach

The main purpose of the study is to identify and extract high priority requirements from the reviews for app maintenance and evolution. Therefore, we propose a novel approach to extract phrases from reviews and use machine learning techniques to predict those high priority requirements. The overall architecture of the approach can be diagrammed as Figure 2. The procedure of the approach is briefly summarized as the following four main steps. (1) Preprocessing the reviews and removing the noise for the stage of requirement phrase extraction. (2) Extracting requirement phrases in the reviews by the collocation finding method. (3) Calculating and obtaining multi-dimensional features for the extracted requirement phrases. (4) Training and testing different machine learning models with the features for predicting whether the requirement phrases are highly prioritized to implement. The details are explained in subsequent sub-sections.




**Figure 2.** The overall architecture of the approach.



### 3.1. Review Preprocessing

Figure 3 shows an example of raw reviews. The reviews mainly include the textual titles (in bold font) and contents, and star ratings (totally five levels) for the apps. To prevent the noise from interfering with the phrase extraction from the raw reviews, the textual data are required to be preprocessed as follows:

Rating	Reviews
 5.0	<b>Extremely Helpful and Organized</b> I especially love when I'm sent collages from past photos and videos!

**Figure 3.** An example of raw reviews.

1. Review filtering. Since all users can write reviews for apps, noise can arise. First, we remove duplicated reviews. These duplicated reviews have a high possibility of maliciously copying others' reviews and do not represent the reviewers' own opinions. Since further steps include calculating the frequency of phrases and duplication will affect the results, the duplicated ones are only considered to appear once. Second, we analyze the language of the reviews by the LangDetect method and then deleting non-English reviews. Due to different grammatical rules and processing methods existing in different languages, it is not proper for preprocessing with mixed languages. The LangDetect method is based on Google's language-detection project, which gets over 99% precision for detecting 53 languages [25].
2. Sentence splitting and sentiment analysis. Usually, one review contains more than one sentence, and each sentence can review different aspects of the apps and have different opinions. In addition, although the review has a corresponding holistic star rating, each sentence may not match the overall rating. Therefore, we split reviews into sentences and apply sentiment analysis for each sentence with TextBlob. TextBlob is a popular Python library for processing textual data such as sentiment analysis and other NLP tasks [26]. The result of sentiment analysis is a polarity score ranged in  $[-1, 1]$  where  $-1$  means very negative and  $1$  means very positive. Meanwhile, we mark each sentence in the review with the original overall star rating as a contrast to the sentiment result since rating is also widely studied [6,27–29].
3. Part-of-Speech (POS) tagging and filtering. Nouns, adjectives, and verbs in reviews are most likely to describe the functionalities of apps according to Guzman and Maalej [24]. Hence, we apply the POS technique to tokenize sentences into words and tag the POS for each word using the Natural Language ToolKit (NLTK). NLTK is another famous NLP tool containing tokenization, stemming, tagging, and many other libraries, and it is widely used in app review preprocessing [16,24,26]. Only the words tagged as nouns, adjectives, or verbs are preserved for further steps.
4. Word formatting. English words that have different syntactical forms share equal semantics (e.g., the tense of verbs, the comparative forms of adjectives, and the singular or plural forms of nouns). To prevent words with the same meaning from interfering with the frequency calculation due to their various forms, the Lemmatization module in NLTK is used to restore the words to their original forms after tokenization. For example, the verbs "takes" and "took" will be lemmatized to "take". Besides, since misspellings occur frequently in raw reviews, we apply a Python library AutoCorrect to correct those words.
5. Stopword removal. Some words (e.g., "is", "this") have no specific meanings but appear frequently in the reviews. These words are not helpful for app functionality descriptions and could severely affect the results of phrase extraction. The NLTK

standard English stopwords corpus defines such words and we remove these words from word tokens. Meanwhile, referring to the stopwords used for app reviews in previous studies [8,24] and combining app and review characteristics, some stopwords (e.g., the name of the apps and “please”, “plz”) are appended into the stopwords list.

After all the steps in preprocessing, the raw reviews are formed into a table where a sentence is a sample with its tokens, sentiment score, and rating as shown in Table 1.

### 3.2. Requirement Phrase Extraction

User requirements can generally be described in terms of phrases. For example, if a user proposes a requirement about a specific user interface, “user interface” is a phrase describing such requests. Because of the controversy of definition for the requirement types, and the classification of reviews can produce certain errors, the request mining work in this paper does not apply the classification process. In contrast, we focus on the user requirement phrase extraction, namely, extracting related phrases about app features and analyzing user opinions as well as the advantages and disadvantages of these features.

The textual reviews are constituted by words, and two adjacent words can compose a bigram collocation. Because of the continuity of semantics, the occurrence of collocations has a certain regularity [30,31]. Some collocations appear frequently, while others only occur occasionally. Hence, the number of occurrences of collocations in the entire corpus can be counted and those recurring collocations can be considered as meaningful phrases for user requirements.

We apply the NLTK Collocation module to extract and analyze collocations. The words in the collocations do not necessarily need to be adjacent, and several words can separate the two words in a collocation, which is called skip-grams method [32]. Hence, we adjust the length of a context window with the window\_size parameter and the words in the context windows can compose bigram collocations in pairs. The window\_size is experimentally set to 3, which means that a collocation can be composed of two words separated by one word, considering that POS filtering is already applied and many invalid words are removed.

Based on the number of overall reviews and collocations, collocations that appear less than five times are filtered out in the whole collocation collection [24]. Those collocations are presumed to occur by chance and have no specific meanings for user requirements. In addition, at least one noun is necessary for the requirement phrases for the user requirements should focus on nouns and without nouns, the requirements cannot point to concrete objects. So, such collocations in the form of “adjective-adjective” or “verb-verb” are removed. The experimental results show that with the filtering step, many meaningless or equivocal collocations can be removed.

In this step, we not only extract collocations for each sentence preprocessed, but summarize overall collocations, filter out unqualified collocations, and finally acquire the requirement phrase collection. Table 1 shows an example of requirement phrase extraction for sentences (noting, some collocations have been filtered out).

**Table 1.** An example of preprocessing results and requirement phrase extraction for sentences.

<b>Sentence</b>	Automatically updates photos to cloud easy sharing and great organization.
<b>Tokens</b>	‘update’, ‘photo’, ‘cloud’, ‘easy’, ‘sharing’, ‘great’, ‘organization’
<b>Sentiment score</b>	0.6166666666666667
<b>Rating</b>	5.0
<b>Requirement phrases</b>	(‘update’, ‘photo’), (‘photo’, ‘cloud’), (‘photo’, ‘easy’), (‘great’, ‘organization’)

### 3.3. Feature Extraction for Phrases

Each extracted requirement phrase has different features (e.g., occurrence frequency). We conjecture that the features of high priority requirement phrases are different from



the features of low priority requirement phrases on account of a basic hypothesis that those high priority requirements are generally discussed more frequently than with low evaluations in user reviews [14,33,34]. Hence, features about requirement frequency and evaluation can be extracted to predict requests in high priority.

In our work, a total of fourteen features were extracted for each requirement phrase and summarized in Table 2 with their category, name, range, and example. The fourteen features can mainly be divided into three categories: frequency, evaluation, and similarity. The frequency and evaluation categories directly reflect the conjecture, while similarity considers the different representations of the same requirement and it is an adjustment for the raw frequency and evaluation. Details on features are clarified by category in sub-sections.

**Table 2.** Feature summary for prediction.

Category	Name	Range	Example (for the Phrase “Save Photo”)
Frequency	$f\_grams$		0.008
	$f\_w1$		0.013
	$f\_w2$		0.091
	$f\_ix$	[0, 1]	0.057
	$f\_xi$		0.376
	$f\_io$		0.049
	$f\_oi$		0.368
	$mention\_rate$		0.011
Evaluation	$rating$	[1, 5]	4.69
	$sentiment$	[−1, 1]	0.351
Similarity	$simrate$	[0, 1]	0.050
	$mention\_rate\_s$	[0, 1]	0.177
	$rating\_s$	[0, 5]	4.24
	$sentiment\_s$	[−1, 1]	0.258

### 3.3.1. Features in Category Frequency

For category frequency, the rationale is that a high priority requirement should be discussed in higher frequency. We calculate the frequency of a phrase from three levels—word-level, phrase-level, and sentence-level:

(1) Word-level—The frequency of the first word ( $f\_w1$ ) and the second word ( $f\_w2$ ). Since the phrase includes two words (the first word  $w1$  and the second word  $w2$ ), we calculate individual word frequency by the count for the word divided by the total number of words as in (1):

$$f\_w(p) = \frac{n(p(w))}{N_w} \quad (1)$$

where the parameter  $p$  means the feature is calculated for the phrase  $p$ ,  $f\_w$  means  $f\_w1$  or  $f\_w2$ ,  $n(p(w))$  means  $n(p(w1))$  or  $n(p(w2))$  representing the count of the  $w1$  or  $w2$  of  $p$  in after preprocessing review corpora, and  $N_w$  means the total number of words in the corpora.

(2) Phrase-level—The frequency of the phrase ( $f\_grams$ ), the frequency of the phrase that contains  $w1$  ( $f\_ix$ ) or  $w2$  ( $f\_xi$ ), and the frequency of the phrase that only contains one of the two words, i.e., containing the  $w1$  but not  $w2$  ( $f\_io$ ) and containing the  $w2$  but not  $w1$  ( $f\_oi$ ). The algorithm is shown in (2):

$$f\_p(p) = \frac{n(p)}{N_p} \quad (2)$$

where  $f\_p$  means the five kinds of phrase-level frequency,  $n(p)$  means the count for the corresponding phrase (the initial phrase  $p$ , phrase  $p\_ix$  having  $w1$ , phrase  $p\_xi$  having  $w2$ , phrase  $p\_io$  only having  $w1$ , phrase  $p\_oi$  only having  $w2$ ), and  $N_p$  means the count of all phrases (with repetition). For example,  $f\_grams$  is calculated by the count for the

initial phrase  $p$  divided by  $Np$ , which considers the phrase as a whole part. The other four consider individual words in the phrase— $f_{ix}$  and  $f_{xi}$  reflect the occurrence ratio of  $w1$  or  $w2$  in all phrases, whereas  $f_{io}$  and  $f_{oi}$  indicate the probability that the two words appear independently in phrases. The correlations between the five features are shown in (3):

$$f_{io} = f_{ix} - f_{grams}, f_{oi} = f_{xi} - f_{grams} \quad (3)$$

The phrase-level frequency is inspired by a classical phrase extraction algorithm—PMI (Point-wise Mutual Information) [35] formulated in (4):

$$PMI(w1, w2) = \ln \frac{p(w1, w2)}{p(w1)p(w2)} \quad (4)$$

where  $p(w1, w2)$  means the probability of the co-occurrence of two words (in the same phrase) whereas  $p(w1)$  and  $p(w2)$  means the probability of the two words appearing independently. High PMI value means that the two words have a higher probability of combining into a meaningful phrase. We do not use the PMI value as a feature but its meta-data are more convenient and direct for model interpretation.

(3) Sentence-level—The frequency of how many sentences mention the phrase (*mention\_rate*). It is calculated as in (5):

$$mention\_rate(p) = \frac{n(s)}{N_s} \quad (5)$$

where  $n(s)$  means the count for the sentences that mention the phrase, and  $N_s$  means the total count of all sentences.

### 3.3.2. Features in Category Evaluation

For category evaluation, the rationale is that a high priority requirement is generally with lower user ratings and lower sentiments. We use both star ratings and sentiment analysis results to present evaluation features:

(1) Average rating (*rating*). In the preprocessing step, we preserve the rating for sentences and rating for a requirement phrase is the arithmetic mean of all the ratings of sentences that mention the phrase as (6):

$$rating(p) = \frac{\sum_{i=1}^{n(s)} r(p_i)}{n(s)} \quad (6)$$

where  $r(p_i)$  is the raw rating for the  $i$ -th sentence mentioning the phrase  $p$ , and  $n(s)$  means the count for the sentences that mention the phrase.

(2) Average sentiment (*sentiment*). The method to compute this feature is like rating except that we use the sentiment results for the sentences instead of raw ratings, which is formulated in (7):

$$sentiment(p) = \frac{\sum_{i=1}^{n(s)} s(p_i)}{n(s)} \quad (7)$$

where  $s(p_i)$  is the sentiment analysis result for the  $i$ -th sentence mentioning the phrase  $p$ .

The difference between rating and sentiment is that rating represents the autonomous evaluation by users but the rating is for the whole review, not the sentence or the requirement phrase, whereas sentiment indicates the evaluation for the sentence so it is more accurate for a concrete request but the sentiment is a statistical result from a predefined sentiment corpus that may not represent the original evaluation of users. The two features have their strengths and weaknesses, so we keep both.

### 3.3.3. Features in Category Similarity

Since the same requirement can be stated with different words and phrases, previous studies use synonym skills to group phrases or sentences [14,15,24]. In this study, we used

a novel method to extract features regarding the similarity between phrases [36], as it is another perspective to demonstrate the heat of discussion with certain requests considering synonyms in reviews. The similarity is calculated by the cosine similarity between the two phrases with a pre-trained word2vec model [37] trained by about 100 billion words from the Google News dataset [38,39], which can reflect the semantic meanings of words by their vector representations. The rationale is that we hypothesize that the requirement phrases with the same semantics should be considered as one phrase and the frequency and evaluation should be adjusted.

The features in category similarity are as follows:

(1) The rate of similar phrases (*simrate*). It is a feature summarizing how many phrases have the same meaning to the phrase and is normalized to rate by dividing the count of different phrases as in (8):

$$\text{simrate}(p) = \frac{n(sp)}{N_{pd}} \quad (8)$$

where  $n(sp)$  means the number of similar phrases of the phrase  $p$ , and  $N_{pd}$  means the count of all phrases (without repetition). High *simrate* indicates that the phrase has many similar phrases and the corresponding requirement is hotly discussed.

(2) The frequency of how many sentences mention the phrase and their similar phrases (*mention\_rate\_s*). It is the adaption of *mention\_rate* that considering similar phrases and calculated by the sum of all the *mention\_rate* for all phrases having the same requirement as in (9):

$$\text{mention\_rate\_s}(p) = \sum_{i=1}^{n(sp)} \text{mention\_rate}(p_i) \quad (9)$$

where  $p_i$  means the  $i$ -th phrase similar with the phrase  $p$ .

(3) Adapted average ratings (*rating\_s*) and sentiment (*sentiment\_s*). Considering the similar phrases, the average rating (or sentiment) is not for the identical phrase but all similar phrases as in (10):

$$\text{rating\_s}(p) = \frac{\sum_{i=1}^{n(sp)} \text{rating}(p_i)}{n(sp)}, \text{sentiment\_s}(p) = \frac{\sum_{i=1}^{n(sp)} \text{sentiment}(p_i)}{n(sp)} \quad (10)$$

After all the fourteen features are extracted, a matrix stores all the data with each row as a sample for a requirement phrase corresponding to its 14 features, and each column represents an individual feature.

### 3.4. High Priority Requirement Phrase Predicting

In this section, we applied supervised machine learning techniques to classify requirement phrases using the fourteen extracted features so that we can obtain high priority requirement phrases automatically. Requirement phrases obtained can be roughly divided into two categories—high priority requirement and low priority requirement. High priority requirements are those features that will be implemented or those bugs that will be fixed preferentially in the next few versions. We consider that requirement phrases covered in changelogs are high priority requirements [8,14], since changelogs state the main addressed requirements or bugs. The annotation of the priority for requirement phrases is detailed and discussed in Section 4.2.

Since the features' distribution varies in different apps (e.g., one app has a high average rating but the other is not, so the rating feature is not on the same scale), rather than training an integrated model with all features to predict all apps, we train a model for each app with its own features. It is reasonable in actual app development that developers focus on one app and determine its requirement.

In real-world datasets, high priority requests in reviews are generally less than low priority requests. For classical classifier algorithms, different labeled samples need to be balanced to have better training results [40]. Therefore, after all the requirement phrases being annotated, a stratified random sampling method is applied to construct balanced samples, which have an equal number of high priority and low priority requirement phrases.

Before classification model training, it is necessary to perform preprocessing work for the feature matrix. The unprocessed data with differences in distribution may have a greater impact on the prediction results. We adopt two steps to preprocess the feature matrix with the preprocessing module in scikit-learn [41]: (1) Individual samples are L2 normalized to have the same unit norm. (2) Individual features are scaled to a standard normal distribution.

Five classical supervised learning algorithms were trained with the default parameters in scikit-learn using balanced samples to classify requirement phrases into high priority or low priority: support vector machine (SVM), multiLayer perceptron (MLP), random forest (RF), Gaussian naïve Bayes (GNB), and decision tree (DT). Performance results were compared between algorithms. These methods are not randomly selected since previous studies have proven the effectiveness of the techniques in review classification [9,11,15,42,43], fake review detection [44], and product demands prediction [34], in which these studies likewise extract features from reviews and automatically accomplish their prediction tasks.

To avoid overfitting, it is standard to split samples into a training set and a test set where models are taught in the training set and tested with unseen data in the test set; however, the random choice of splitting sets will have an unstable result, so cross-validation was applied to evaluate the performance of the model. The samples were divided into k-folds (in this paper, we experimentally set k to 10), one of which was in turn used as a test set, and the remaining k-1 folds were used for training sets. After k-times of model training, the average scores were computed as the final assessment of the model. Since each cross-validation will randomly split samples with different folds, results can vary so we repeated the cross-validation 30 times to obtain the average results of higher stability. Moreover, to guarantee that each split has balanced samples, stratified sampling was used in the splitting step.

#### 4. Evaluation Method

Real-world reviews were used to evaluate the results of the approach for predicting high priority requirement phrases. In this section, we explain the used datasets, how the truth set was created, and the performance metrics. For specific evaluation purposes, we focus on the following research questions (RQ):

- **Effectiveness (RQ1):** What is the performance of different machine learning techniques when predicting high priority requirement phrases and which one is the best?
- **Comparison (RQ2):** How does our approach compare to other techniques for requirement extraction in app reviews?
- **Model interpretation and optimization (RQ3):** What is the importance of different classification features? Is the basic hypothesis that high priority requirements are generally discussed more frequently while being discussed less in low evaluations in user reviews correct? How do we optimize the model?

##### 4.1. Review Datasets

We mainly collected reviews from the Apple App Store. Given that the reviews and changelogs for the apps are needed in pairs, the selected apps should meet the following requirements: (1) The app should be active and currently being updated; (2) The number of users and reviews of the app should be sufficient; (3) A detailed changelog statement is required, simply stating “bug fixes and experience improvements” is not sufficient, and any changelogs that do not specifically point to app functionalities are excluded.

The App Annie is a third-party app data analysis platform. It collects data from mainstream app stores around the world, including app introductions, popularity rankings for a

certain period, user reviews, and ratings for the apps in each historical version. Considering the popularity rankings, the number of reviews, and the quality of changelogs from this platform, six apps from three categories in Apple App Store were chosen according to the selecting criteria. In detail, the review data of the target apps from September to November 2019 in the region of the United States were retrieved, including the review titles, review contents, app version numbers, user ratings, and historical changelogs. Table 3 shows the subject apps with their names, categories, number of versions, and number of reviews. Overall, 44,893 original reviews from 88 versions assure the generalization of our approach. The review filtering step does not remove too many of the reviews (about 10%), so we can infer that the raw reviews collected are not severely duplicated and generally written in English.

**Table 3.** Summary of selected apps and reviews.

App	Category	#Versions	#Reviews (Original)	#Reviews (after Filtering)
TikTok	Entertainment	15	15,545	13,314
Netflix	Entertainment	17	2544	2406
Google Photos	Photo & Video	13	6794	5607
Snapchat	Photo & Video	15	10,148	9622
Yelp	Travel	13	5216	4902
Uber	Travel	15	4646	4367

#### 4.2. Truth Set Creation

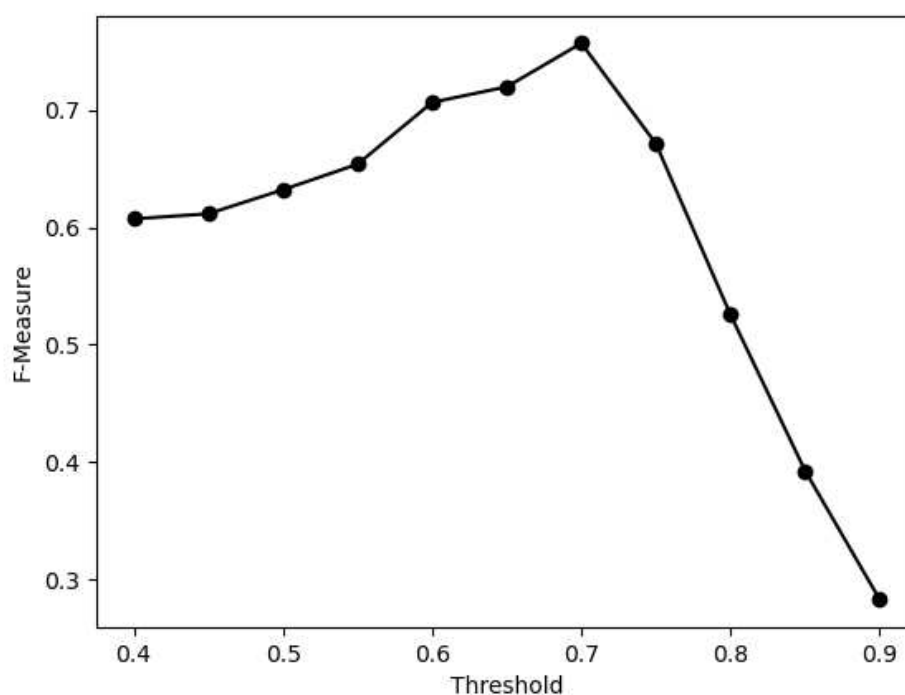
It is essential to create a truth set for the extracted requirement phrases with their priority to be implemented not only for the supervised learning procedure but for the evaluation of the model. However, as is stated before, it is a challenge to get all the requirement phrases annotated due to the large quantity of the phrases and the complexity of priority. Changelogs are widely used as ground truth for issues or request identification from user reviews [5,8,14] since emergent issues or requests will be implemented and then logged as new functionalities in changelogs. We annotate those requirement phrases covered in historical changelogs as high priority requirements for the whole review time period. We do not distinguish reviews between versions because we only collect reviews for a three month period and changelogs for each version are not sufficient to state all high priority requests with only a few sentences. Using changelogs for just the next version to annotate the high priority requests is unreasonable. As shown in Table 3, an app gets a new release every two weeks on average, and some releases are routine bug fixes and performance improvements, generally speaking; however, users still propose their requests in reviews and it is unfair to conclude that there are no high priority requests for these versions. In other words, request mining from reviews should also analyze reviews in the previous few versions rather than the present version. Although we do not further explore what the concrete time period proper for request mining is, we do analyze reviews in a longer time period and annotate requests with their corresponding historical changelogs.

Normally, manual methods are used for annotating. For labeling high priority reviews, annotators are required to compare reviews to changelogs sentence by sentence. In order to label topics or phrases, annotators should first extract functionalities from changelogs and then compare word by word. We explore a novel semi-automated method to reduce the effort for manual annotation. Specifically, the first step is to extract functionality phrases from changelogs. The same preprocessing and phrase extraction steps are executed for the changelogs except that we no longer limit the frequency of occurrence for the phrases since changelogs have fewer sentences than reviews and the same functionalities do not duplicate as in reviews unless iterated many times. Then, two authors separately pick out those phrases regarding app functionalities from all the extracted phrases, and proofread the corresponding changelogs. All disagreements are resolved by two annotators until the two authors reach an agreement. To distinguish the requirement phrases extracted from reviews, we define the final selected phrases from changelogs as functionality phrases. We do not

directly manually extract functionality phrases from changelogs because it is necessary to have the extracted functionality phrases use the same format as the requirement phrases so that we can apply the same procedure as used in the reviews, which is more secure.

The second step is to link the requirement phrases with functionality phrases. Once again, we applied the pre-trained Google News word2vec model to calculate the similarity between requirement phrases and functionality phrases. We need to traverse all the functionality phrases for each requirement phrase and obtain the maximum similarity and the most similar phrase pair for the purpose of judging whether changelogs cover the requirement phrases. If the maximum similarity exceeds the threshold, the two phrases are considered to have the same meaning and the requirement phrase is annotated as a high priority request.

Regarding the determination of the threshold, a sample of 200 pairs of phrases (the most similar requirement phrase and functionality phrase pairs) are randomly selected for experimentation, which is enough to reflect the population. The ground truth is created by manually analyzing whether the two phrases are expressing the same or similar semantics by the two annotators independently. Disagreements are likewise discussed and finally eliminated. F-Measure [45] is the performance metric for the evaluation of different thresholds [5]. Since the range of similarity is  $[-1, 1]$ , the thresholds were selected at an interval of 0.05, and the F-Measure results under different thresholds are shown in Figure 4. According to the results, the threshold is finally determined as 0.70, which is also used for the similarity feature extraction.



**Figure 4.** Results for the thresholds selected and corresponding F-Measure.

Table 4 indicates the results for the truth set creation with the semi-automated approach and the percent of high priority requirement phrases is calculated. We believe that Google Photos is more of a tool-like app without too many reviews discussing the content issues in apps with videos, such as TikTok, so the reviews relate more to suggestions for app development. However, since the F-Measure is not high enough to neglect the error for truth set creation, we sampled 1000 phrases for each app and double-check the veracity of the automated annotation. The double-checked manual annotation results and the F-Measure for the automated results are listed in Table 5. Since we need to construct a balanced sample for each app to train the machine learning model, a proper number of 600 requirement phrases (300 high priority requirement phrases and 300 low prior-



ity requirement phrases) were sampled for each app from 1000 unbalanced manually annotated phrases.

**Table 4.** Results for semi-automated methods for truth set creation.

App	#Requirement Phrases	#High Priority	Percent
TikTok	6092	1291	21.2%
Netflix	1195	374	31.3%
Google Photos	1679	841	50.1%
Snapchat	6460	2130	33.0%
Yelp	2196	629	28.6%
Uber	3072	981	31.9%

**Table 5.** Double-checked results of sampled 1000 phrases for each app.

App	#High Priority	F-Measure
TikTok	330	0.596
Netflix	387	0.657
Google Photos	577	0.751
Snapchat	439	0.703
Yelp	402	0.679
Uber	436	0.637

#### 4.3. Performance Metrics

A total of five methods were used to evaluate the effectiveness of our approach to predict high priority requirement phrases compared with the truth set: accuracy, precision, recall, F-Measure, and ROC\_AUC, which are widely used in classification tasks [14,46]. The calculations of the first four methods are formulated in (11), where TP means that the requirement phrase is a high priority and was identified as a high priority, FP means that the phrase is a low priority but was identified as a high priority, TN means that the phrase is a low priority and was identified truly as a low priority, and FN represents that the phrase is a high priority but was identified as a low priority.

$$\begin{aligned}
 Accuracy &= \frac{TP + TN}{TP + TN + FP + FN}, Precision = \frac{TP}{TP + FP} \\
 Recall &= \frac{TP}{TP + FN}, F - Measure = \frac{2 * Precision * Recall}{Precision + Recall}
 \end{aligned} \quad (11)$$

Accuracy represents how the model predicts the results and has a reliable validity on balanced samples. Precision is the proportion of correctly predicted samples among all the samples identified as high priority (low priority can be wrongly identified as high priority, FP). Precision represents how the model truly identifies high priority requests. Recall is the proportion of correctly identified samples in all true high priority requests (the true high priority samples may not be classified as high priority, FN). Recall represents how the model identifies all the priority requests. The F-Measure is the harmonic mean of both precision and recall, which can represent both precision and recall. ROC\_AUC means the area under ROC curve and the value represents “the probability that a randomly chosen positive example is ranked higher than a randomly chosen negative example [47]”.

## 5. Results

In this part, results for the predicting task are presented and the three main research questions are deeply analyzed and discussed.

### 5.1. Effectiveness (RQ1)

Table 6 reports the performance results for the five machine learning methods set with their default parameters in scikit-learn. The initial prediction results are acceptable for averagely about 60% score in all performance metrics. There are some differences between apps due to app characteristics, but overall, the results are stable. The results preliminarily verify the effectiveness of these supervised learning techniques trained with the fourteen features.

For techniques comparison, in six apps  $\times$  five performance metrics (a total of 30 optimal results): RF achieves optimal performance 20 times; SVM achieves optimal performance 6 times; MLP achieves optimal performance 3 times; GNB only achieves optimal performance 1 time. For individual performance metrics: RF achieves optimal accuracy on five apps, optimal precision on four apps, optimal recall on three apps, optimal F-Measure on four apps, and optimal ROC\_AUC on four apps, while the other five models only achieve the best results once, very infrequently.

**Table 6.** Performance results for different methods (the results styled in bold font are the optimal performances for the same evaluation methods).

App	Method	Accuracy	Precision	Recall	F-Measure	ROC_AUC
TikTok	SVM	0.602	0.580	<b>0.745</b>	<b>0.650</b>	0.623
	MLP	0.577	0.571	0.633	0.598	0.621
	RF	<b>0.626</b>	<b>0.623</b>	0.646	0.631	<b>0.674</b>
	GNB	0.582	0.565	0.723	0.632	0.596
	DT	0.583	0.587	0.569	0.575	0.583
Netflix	SVM	0.647	0.688	0.538	0.600	0.677
	MLP	0.663	0.678	0.627	0.649	0.708
	RF	<b>0.685</b>	<b>0.689</b>	<b>0.678</b>	<b>0.681</b>	<b>0.754</b>
	GNB	0.620	0.681	0.456	0.542	0.635
	DT	0.624	0.627	0.620	0.621	0.624
Google Photos	SVM	0.578	<b>0.658</b>	0.330	0.436	0.648
	MLP	0.590	0.595	0.575	0.581	0.639
	RF	<b>0.613</b>	0.621	<b>0.590</b>	<b>0.602</b>	<b>0.654</b>
	GNB	0.572	0.644	0.327	0.429	0.592
	DT	0.571	0.573	0.574	0.571	0.571
Snapchat	SVM	0.591	0.577	<b>0.699</b>	<b>0.630</b>	<b>0.644</b>
	MLP	0.593	0.590	0.620	0.602	0.628
	RF	<b>0.605</b>	<b>0.605</b>	0.608	0.604	0.637
	GNB	0.569	0.578	0.530	0.541	0.598
	DT	0.554	0.556	0.552	0.551	0.554
Yelp	SVM	0.616	0.684	0.436	0.527	0.649
	MLP	<b>0.629</b>	0.646	0.577	0.607	<b>0.690</b>
	RF	0.626	0.632	<b>0.611</b>	<b>0.618</b>	0.677
	GNB	0.590	<b>0.705</b>	0.315	0.430	0.620
	DT	0.563	0.565	0.565	0.562	0.563
Uber	SVM	0.553	0.570	0.434	0.488	0.570
	MLP	0.576	0.577	<b>0.575</b>	0.573	0.611
	RF	<b>0.580</b>	<b>0.582</b>	0.572	<b>0.574</b>	<b>0.616</b>
	GNB	0.532	0.572	0.266	0.358	0.577
	DT	0.532	0.533	0.527	0.528	0.532
Average	SVM	0.598	<b>0.626</b>	0.530	0.555	0.635
	MLP	0.605	0.610	0.601	0.602	0.650
	RF	<b>0.623</b>	0.625	<b>0.618</b>	<b>0.618</b>	<b>0.669</b>
	GNB	0.578	0.624	0.436	0.489	0.600
	DT	0.571	0.574	0.568	0.568	0.571

On average, RF achieves a +1~2 percent in terms of all performance metrics except precision where the results for SVM, RF, and GNB are quite similar with only a 0.1 percent difference. Though the superiority is not that huge, we conclude that RF has the best performance in the predicting task and the following model interpretation and optimization are for the RF model only.

## 5.2. Comparison (RQ2)

As far as we know, the research that is most relevant to us is the IDEA framework [8]. The authors intended to detect emerging issues/topics from app reviews, and the emerging issues are basically equal to our defined high priority requests, because the same method is used for truth set creation, in which IDEA extracts bug fixes and request improvements from changelogs as emerging issues, which is consistent with the definition of high priority requirements. While IDEA applies adapted topic modeling methods that outperform other topic modeling methods, we extracted fourteen features and trained RF models to predict the issues.

We run our method on the IDEA dataset, including the same required textual reviews, ratings, and changelogs for another six apps (four from Google Play and two from the Apple App Store). The functionality phrases were manually extracted from changelogs by the authors so as to reduce the work for preprocessing changelogs and annotation. While IDEA extracts issues both in the phrase level and the sentence level, we focus on the phrase level, and we compare our prediction performance only on the phrase level.

The quantitative results shown in Table 7 confirm the overwhelming superiority for RF, and that no matter whether in terms of individual apps or on average, the performance results for RF are much better than IDEA, and RF is more stable and robust for different apps. The RF model can also fit the Android apps in Google Play, which expands the generalization of our approach.

**Table 7.** Performance comparison with IDEA (the number under app name is the total number of reviews).

Platform	App	Method	Precision	Recall	F-Measure
Google Play	Clean Master (44,327)	IDEA	0.677	0.318	0.431
		RF	<b>0.801</b>	<b>0.718</b>	<b>0.755</b>
	eBay (35,483)	IDEA	0.229	0.251	0.227
		RF	<b>0.731</b>	<b>0.609</b>	<b>0.662</b>
	SwiftKey (21,009)	IDEA	0.517	0.653	0.523
		RF	<b>0.804</b>	<b>0.748</b>	<b>0.774</b>
	Viber (17,126)	IDEA	0.625	0.340	0.440
		RF	<b>0.743</b>	<b>0.741</b>	<b>0.741</b>
Apple App Store	NOAA Radar (8363)	IDEA	0.571	0.497	0.531
		RF	<b>0.768</b>	<b>0.741</b>	<b>0.751</b>
	YouTube (37,718)	IDEA	0.592	0.472	0.523
		RF	<b>0.677</b>	<b>0.666</b>	<b>0.669</b>
	Average	IDEA	0.534	0.422	0.446
		RF	<b>0.754</b>	<b>0.704</b>	<b>0.725</b>

In this section, we discuss the reason why RF has a better performance than IDEA. First, IDEA detects emerging issues by the discovery of distribution anomalies in topics between versions. IDEA can identify topic discrepancies between versions, but there is no evidence suggesting a strong connection between user requests and anomalous topics. The discrepancies may come from different users, content, and other non-version related factors. Second, the topic modeling methods are based on the frequency distribution of words, which is considered to be subset features of RF. The sentiment features in RF contribute a lot to the prediction, whereas in IDEA, sentiment analysis is just used for topic interpretation.

Lastly, RF considers the differences between apps and fit estimators for each app, while IDEA applies the same parameters for different apps.

For qualitative results, we present and compare the predicted results for YouTube with the two approaches. We randomly sampled 10 identified phrases for each method and made sure there was no duplicated request; the results are listed in Table 8. Though contingency exists, we can see that both the methods can identify high priority requests quite well and some requests are both identified by both methods. For example, the changelog states “Added slide over and split view support” and IDEA identifies “split screen”, whereas RF predicts “split view”.

**Table 8.** Qualitatively comparison with IDEA (the phrases styled in bold font represent the correctly identified high priority requirements).

Method	Examples for High Priority Requirement Phrases
IDEA	<b>description box</b> ; user interface; <b>split screen</b> ; <b>battery drain</b> ; <b>force quit</b> ; sound quality; <b>home button</b> ; notification center; <b>playback error</b> ; <b>comment section</b>
RF	<b>comment section</b> ; <b>split view</b> ; middle screen; <b>auto play</b> ; <b>video playlist</b> ; keep mess; bring keyboard; <b>video description</b> ; <b>view reply</b> ; <b>drain battery</b>

For one correctly predicted requirement, an example for positive sentences of “split view” is “Well, I have added a star because they have added slide over and split view”, which means that the user is satisfied with the new feature. For negative evaluations, one user reviews “But at that point I can’t write comment in split view”, representing the request of improvement for the split view feature. For one incorrectly predicted requirement, “bring keyboard” in a raw review is “if you do not press it exactly in the right spot, it will automatically bring up the keyboard to write a comment”. There is no positive review for the requirement. We can believe that it is just a mis-operation for a few users and the developers consider it a low priority.

### 5.3. Model Interpretation and Optimization (RQ3)

This section mainly discusses why random forest works for the predicting task by analyzing the fourteen features, and then models are optimized by the interpretation results.

#### 5.3.1. Model Interpretation

Random forest is an ensemble method that works by constructing multiple decision trees from bootstrap samples in the training set and the final result is voted by all trees, which can improve robustness over single estimators [48]. In each decision tree, yes/no questions about features (e.g., is *rating* > 3.5?) are asked for each input sample (tree split), and the answers decide the classification result for the sample. Tree-based estimators can compute the impurity-based feature importance by calculating the frequency of the feature used in the tree split. High frequency shows that the feature is relatively more important.

We first analyze the importance of features to interpret the model, which is calculated by scikit-learn embedded model parameter *feature\_importances\_* for RF. The feature importance to predict high priority requests for each app is listed in Figure 5, where the value of the y-axis means the importance percent of the feature and the x-axis means the features. As shown in the figures, though there are differences between apps, common results can be summarized. On average, the three most important features are *simrate* (9.0%), *sentiment\_s* (8.9%), and *mention\_rate\_s* (8.8%). In individual apps, the latter three features are always in the top five most important features. The results indicate that the frequency and evaluation features in the similarity category work better than concrete words/phrases frequency and sentiments for individual phrases. In other words, the group technique or similarity methods applied to determine the semantic meaning between phrases and sentences is more appropriate and effective since users can use a varied vocabulary in their feature requests. Moreover, *sentiment\_s/sentiment* has higher importance than *rating\_s/rating*

in the six apps, indicating that the sentiment analysis for the individual sentences makes sense in contrast to the original rating for the whole reviews. Users do propose and discuss more than one request with different sentiments suggesting that the overall rating will mix up opinions, so splitting up the sentences is more scientific.

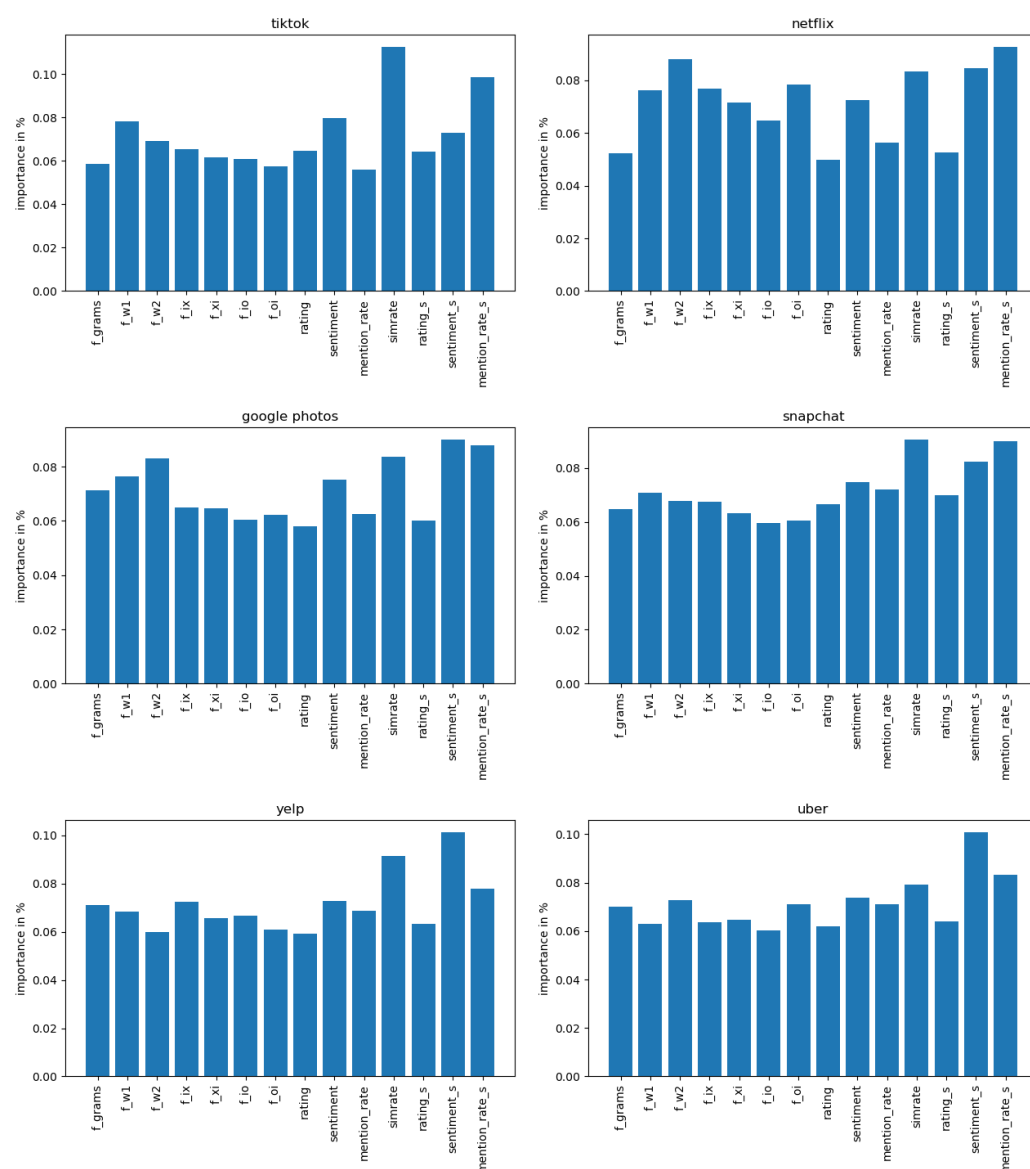
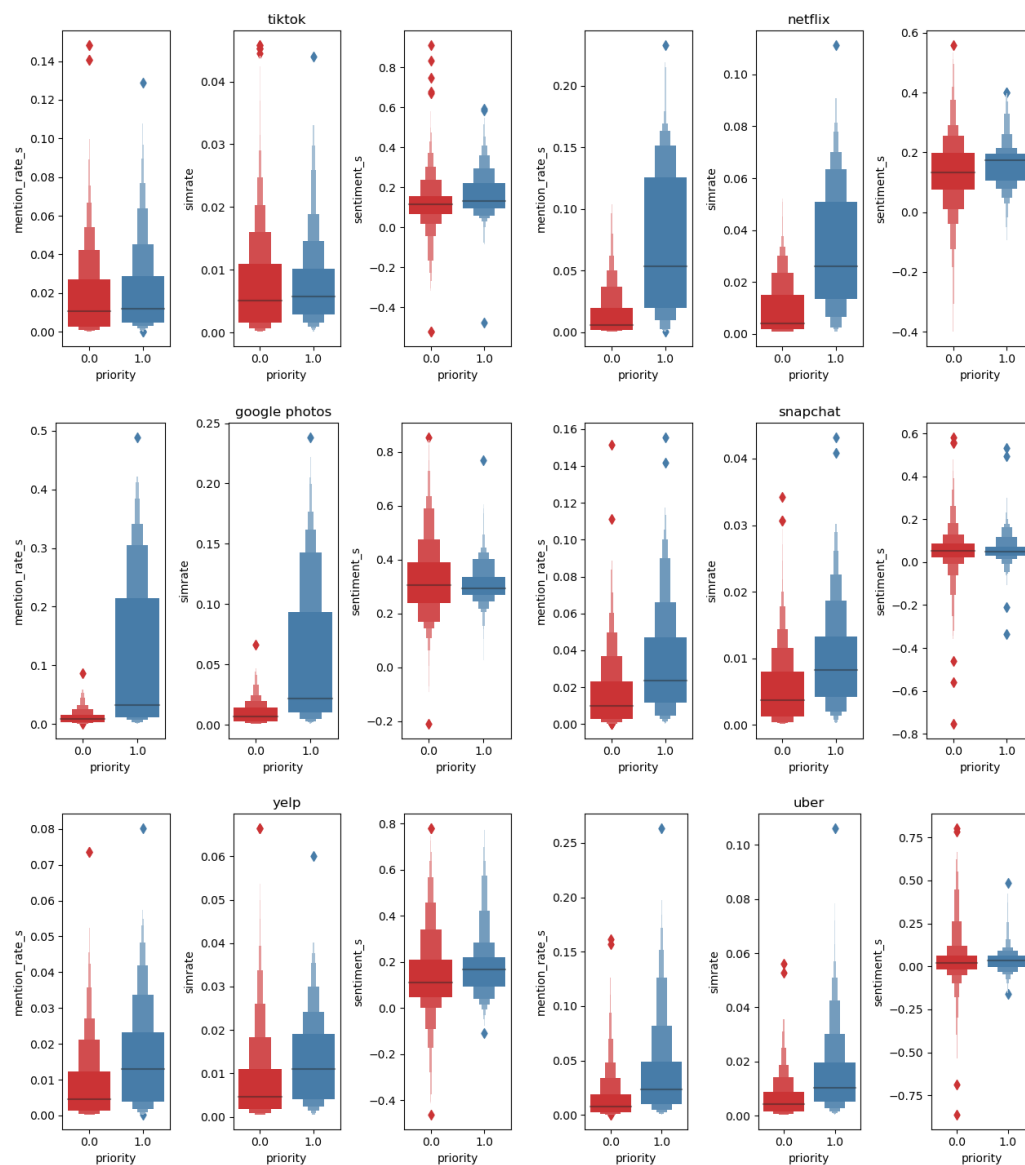


Figure 5. Feature importance for each app.

The three most important features are consistent with our basic hypothesis that frequency and evaluation will affect a request's priority when deciding the features to extract; thus, further exploration for the three features was conducted to examine whether there are differences between high/low priority requirement phrases. We first plot the distribution of the features categorized by the priority with the enhanced boxplot provided by Seaborn [49]. Figure 6 exhibits the plot results where 1.0 for the priority means high priority phrases whereas 0.0 means the low priority. It can be roughly inferred that the average *mention\_rate\_s* and *simrate* in high priority are greater than in low priority, especially for Google Photos, Netflix, and Uber. High priority requests have wider ranges and more phrase samples with the greater *mention\_rate\_s* and *simrate*, while low priority requests have larger quantities but with a few users reviewing the same requests. Additionally, the ranges for *sentiment\_s* in low priority are wider than in high priority, with many samples

having extreme high/low *sentiment\_s*. The reason could be that low priority requests of extremely high or low average evaluation may have only a few mentions by users, so these requirements are not regarded as high priority requests. Moreover, the distribution of the three most important features varies for different apps, so it is necessary to have learning models for each app.



**Figure 6.** Enhanced boxplot for the three most important features in each app.

Statistical tests were applied to analyze whether the differences of the three features between high and low priority are statistically significant. Since the number of samples is not equal, we applied the Mann–Whitney U test [50] (two-sided) and the results are listed in Table 9. At the alpha level of 0.001, mention\_rate\_s and simrate in high priority are significantly greater than in low priority. Except for Google Photos and Snapchat, sentiment\_s in high priority is significantly greater than in low priority.



**Table 9.** Mann–Whitney U test results for the features in high/low priority.

App	<i>mention_rate_s</i>	<i>simrate</i>	<i>sentiment_s</i>
TikTok	$U = 3.41 \times 10^5, ***$	$U = 3.37 \times 10^6, ***$	$U = 3.77 \times 10^6, ***$
Netflix	$U = 2.56 \times 10^5, ***$	$U = 2.52 \times 10^5, ***$	$U = 1.79 \times 10^5, ***$
Google Photos	$U = 5.66 \times 10^5, ***$	$U = 5.55 \times 10^5, ***$	$U = 3.39 \times 10^5$
Snapchat	$U = 6.48 \times 10^6, ***$	$U = 6.42 \times 10^6, ***$	$U = 4.46 \times 10^6, *$
Yelp	$U = 6.68 \times 10^5, ***$	$U = 6.52 \times 10^5, ***$	$U = 5.85 \times 10^5, ***$
Uber	$U = 1.49 \times 10^6, ***$	$U = 1.49 \times 10^6, ***$	$U = 1.10 \times 10^6, ***$

*p*-value: \*\*\* < 0.001, \* < 0.05.

The results partially support our hypothesis that high priority requests have a higher probability to be addressed by more users. However, the evaluation of the high priority requests disobeys our conjecture and it seems that requests with a higher evaluation on average are prioritized. We discuss the reason as follows: (1) Reviews with several versions were used, so the requests proposed in early versions may have been implemented. Users who proposed the requests before add new reviews to praise the newly implemented functionality and the related requests gain evaluation feedback. (2) The new functionality attracts and satisfies new users who also give positive evaluations. Overall, the average sentiment for the previously proposed requests is balanced and are even better than the requests that have not been implemented (low priority). This would also explain why the *sentiment\_s* in high priority is dispersed with a narrower distribution. (3) Some low evaluation requests may be ignored by developers due to effort devotion and many other reasons so these requests are marked as impractical requests. Though not matching our conjecture, the differences exist and the *sentiment\_s* is an important feature for the prediction task. However, the results also remind us that one of the limitations of the approach is that the high priority request detection is not time-effective over versions. The developers should pay attention to the identified requests and seek further confirmation on whether the requests have been implemented or need further improvement.

### 5.3.2. Model Optimization

Feature selection is an effective method to optimize the learning model that can help removing irrelevant and redundant features and can improve prediction performance, reduce training effort, and facilitate model interpretation [51]. Here, we applied a widely used recursive feature elimination [52,53] method to select features. Specifically, RF was initially trained with all features, then, the feature importance of the model was analyzed and the least important feature was eliminated. The rest of the features were used to train a new RF model; the procedure was recursive until there was only one feature left. The same cross-validation method was applied to evaluate the performance of each estimator and figure out how many and which features can reach the best performance; we focused on the F-Measure since different performance metrics can lead to different selection results, and F-Measure is an appropriate choice for our study. For six apps, the optimal number of features is different, ranging from 3 to 14. However, the three most important features were all selected and every feature was selected at least one time as the optimal estimator. The results show that, although there are differences in the apps, the same three important features can improve the performance of the estimator; therefore, no feature is fully abundant for the prediction, demonstrating the effectiveness of feature extraction.

With the optimal features, we further tuned the hyper-parameters using the most widely used method—grid search [54,55], which exhaustively searches candidates from a grid of pre-defined parameters and compares the performance of estimators with different parameters. For the RF model, we mainly tuned four parameters—*max\_features*, *n\_estimators*, *min\_samples\_split*, and *min\_samples\_leaf* according to the characteristics of RF [48,56] and the API of RF in scikit-learn. *max\_features* is the number of features to consider when the tree looks for the best split. *n\_estimators* means the number of trees in the forest. *min\_samples\_split* represents the minimum samples used to split at an internal

node. `min_samples_leaf` defines the minimum samples at a leaf node. Since our approach is intended to not only predict as many high priority requests in reviews as possible (high recall), but also assure that as many of the predicted requests are true high priority requests as possible (high precision), the comprehensive F-Measure is thus the most appropriate. We optimized for the F-Measure only so that the tuned hyper-parameters can obtain the best F-Measure for the model. Table 10 enumerates the performance after optimization. After optimization, the average accuracy, precision, recall, F-Measure, and ROC\_AUC for the prediction task are 67.6%, 67.3%, 69.2%, 68.0% and 71.4%, respectively. The improvement is slight with about +3% to +12% and the recall is improved mostly by an average of 7.4%.

**Table 10.** Performance results after optimization (the number in brackets represents the improvement).

App	Accuracy	Precision	Recall	F-Measure	ROC_AUC
TikTok	0.702 (+0.076)	0.707 (+0.084)	0.710 (+0.064)	0.705 (+0.074)	0.728 (+0.054)
Netflix	0.722 (+0.037)	0.714 (+0.025)	0.747 (+0.069)	0.728 (+0.047)	0.785 (+0.031)
Google Photos	0.677 (+0.064)	0.672 (+0.051)	0.687 (+0.097)	0.677 (+0.075)	0.706 (+0.052)
Snapchat	0.642 (+0.037)	0.741 (+0.036)	0.650 (+0.042)	0.644 (+0.040)	0.674 (+0.037)
Yelp	0.697 (+0.071)	0.689 (+0.057)	0.730 (+0.119)	0.707 (+0.089)	0.746 (+0.069)
Uber	0.617 (+0.037)	0.617 (+0.035)	0.627 (+0.055)	0.620 (+0.046)	0.642 (+0.026)
Average	0.676 (+0.054)	0.673 (+0.048)	0.692 (+0.074)	0.680 (+0.062)	0.714 (+0.045)

Classifying reviews in the categories of feature requests is a more complex problem than other categories [15], let alone the priority of features. We believe that there are overlaps in features between high/low priority requests and it is difficult to obtain better results based on the approach; however, the wrongly detected requests also have meaning, because these requests have the same feature distribution with high priority requests, and the conflict may come from the truth set creation with changelogs or the developers neglecting some user requests.

## 6. Threats to Validity

Threats to construct validity mainly refer to the truth set creation, where we used word2vec model to calculate the similarity between requirement phrases and functionality phrases, which are manually selected by two authors. Different word2vec models and annotators can lead to bias. We alleviate the threat by applying the widely used Google News word2vec model and the manual annotation rules are specified, including conflict solving. The annotators are not required to be experts at app development, since the most important work for annotators is to understand the changelogs and judge whether the extracted phrases are meaningful functionalities. The annotation work is without request estimation for high or low priority, which is relatively objective. Finally, we double-check the semi-manual annotation with the word2vec model by sampling results and examining the accuracy.

Another threat is that the requirement phrases extracted from the reviews may be blended with some meaningless phrases or others, which can decrease the performance for the predicting task and increase the burden for result interpretation. In order to mitigate the threat, we applied a series of preprocessing steps to remove noise before collocation and POS techniques to filter out meaningless phrases; however, the probability still exists since we do not examine which preprocessing methods can lead to the best results, but instead apply general approaches.

For internal validity, the method of truth set creation has a basic requirement, which is that the apps' changelogs cover sufficient high priority requests. These requests are derived from the changelogs released by the app development teams. However, the comprehensiveness and accuracy of changelogs are unstable, and usually, the changelog for one release is not detailed enough with only one or two sentences. How the changelogs state the new functionalities will also seriously affect the request annotating stage. For

this, instead of analyzing reviews version by version, we concentrate on all the collected reviews with corresponding historical changelogs; the cost is that the requests detected may have been implemented in previous versions.

The final chosen machine learning method is RF since RF obtains relatively better results but the differences are not significant, as other models also have good results. We cannot exclude that there are more proper algorithms or hyper-parameter settings for the task that can achieve better performance. It is a very complex process for the releasing plan and the prediction results for high priority requests can only be considered as a recommendation for developers.

Threats to external validity relate the generalization of our approach and findings. Since our approach successfully runs on different apps from different categories and platforms, generalization stands; however, whether the approach and findings can be generalized to apps in other categories (e.g., games), or apps with insufficient reviews, was not investigated in this study. Additionally, since the RF estimator was trained for individual apps, the model cannot be applied to different apps.

## 7. Conclusions

This study proposes a novel approach for mining high priority requirements for apps from user reviews. We used real-world reviews to verify the effectiveness of the method. When compared to the state-of-the-art methods, our approach performed better. This method can be quickly applied to app designers and developers to schedule the release plan for apps. With sufficient user reviews, the approach can automatically analyze a large number of reviews, avoiding unnecessary time-consuming efforts by manual methods to extract high priority user requirements.

A suggested future work is to focus on analyzing phrase features in more dimensions or implementing other classification models to improve the performance of predicting. More specific changelogs and larger review datasets for a longer time period can be obtained to find the most proper time period for request mining. Predicting high priority requests for each version and then formulating the trends for the app requests between versions is another direction.

**Author Contributions:** Conceptualization, C.Y. (Cheng Yang) and L.W.; methodology, C.Y. (Cheng Yang); software, L.W.; validation, C.Y. (Chunyang Yu) and Y.Z.; writing—original draft preparation, L.W.; writing—review and editing, L.W. and C.Y. (Chunyang Yu). All authors have read and agreed to the published version of the manuscript.

**Funding:** This research is supported by the National Natural Science Foundation of China (No. 62002321), and Zhejiang Provincial Natural Science Foundation of China (No. Y18E050014).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data are not publicly available due to privacy or ethical.

**Conflicts of Interest:** The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## References

1. Genc-Nayebi, N.; Abran, A. A systematic literature review: Opinion mining studies from mobile app store user reviews. *J. Syst. Softw.* **2017**, *125*, 207–219. [[CrossRef](#)]
2. Xie, H.; Yang, J.; Chang, C.K.; Liu, L. A statistical analysis approach to predict user's changing requirements for software service evolution. *J. Syst. Softw.* **2017**, *132*, 147–164. [[CrossRef](#)]
3. Jabangwe, R.; Edison, H.; Duc, A.N. Software engineering process models for mobile app development: A systematic literature review. *J. Syst. Softw.* **2018**, *145*, 98–111. [[CrossRef](#)]
4. Jha, A.K.; Lee, S.; Lee, W.J. An empirical study of configuration changes and adoption in Android apps. *J. Syst. Softw.* **2019**, *156*, 164–180. [[CrossRef](#)]

5. Palomba, F.; Linares-Vásquez, M.; Bavota, G.; Oliveto, R.; Penta, M.D.; Poshyvanyk, D.; Lucia, A.D. Crowdsourcing user reviews to support the evolution of mobile apps. *J. Syst. Softw.* **2018**, *137*, 143–162. [CrossRef]
6. Noei, E.; Zhang, F.; Wang, S.; Zou, Y. Towards prioritizing user-related issue reports of mobile applications. *Empir. Softw. Eng.* **2019**, *24*, 1964–1996. [CrossRef]
7. Pagano, D.; Maalej, W. User feedback in the Appstore: An empirical study. In Proceedings of the 2013 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro, Brazil, 15–19 July 2013; pp. 125–134.
8. Gao, C.; Zeng, J.; Lyu, M.R.; King, I. Online app review analysis for identifying emerging issues. In Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 27 May–3 June 2018; pp. 48–58.
9. Chen, N.; Lin, J.; Hoi, S.C.; Xiao, X.; Zhang, B. AR-miner: Mining informative reviews for developers from mobile app marketplace. In Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 31 May–7 June 2014; pp. 767–778.
10. Li, C.; Huang, L.; Ge, J.; Luo, B.; Ng, V. Automatically classifying user requests in crowdsourcing requirements engineering. *J. Syst. Softw.* **2018**, *138*, 108–123. [CrossRef]
11. Suprayogi, E.; Budi, I.; Mahendra, R. Information extraction for mobile application user review. In Proceedings of the 2018 International Conference on Advanced Computer Science and Information Systems (ICACSIS), Yogyakarta, Indonesia, 27–28 October 2018; pp. 343–348.
12. Nayebi, M.; Ruhe, G. Optimized functionality for super mobile apps. In Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, 4–8 September 2017; pp. 388–393.
13. Gu, X.; Kim, S. What parts of your apps are loved by users? In Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 9–13 November 2015; pp. 760–770.
14. Scalabrino, S.; Bavota, G.; Russo, B.; Penta, M.D.; Oliveto, R. Listening to the crowd for the release planning of mobile Apps. *IEEE Trans. Softw. Eng.* **2019**, *45*, 68–86. [CrossRef]
15. Panichella, S.; Di Sorbo, A.; Guzman, E.; Visaggio, C.A.; Canfora, G.; Gall, H.C. 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), Bremen, Germany, 29 September–1 October 2015; pp. 281–290.
16. Maalej, W.; Kurtanović, Z.; Nabil, H.; Stanik, C. On the automatic classification of App reviews. *Requir. Eng.* **2016**, *21*, 311–331. [CrossRef]
17. McIlroy, S.; Ali, N.; Khalid, H.; Hassan, A.E. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empir. Softw. Eng.* **2016**, *21*, 1067–1106. [CrossRef]
18. Guzman, E.; El-Haliby, M.; Bruegge, B. Ensemble methods for App review classification: An approach for software evolution. In Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 9–13 November 2015; pp. 771–776.
19. Jha, N.; Mahmoud, A. Mining non-functional requirements from App Store reviews. *Empir. Softw. Eng.* **2019**, *24*, 3659–3695. [CrossRef]
20. Cambria, E.; Schuller, B.; Xia, Y.; Havasi, C. New avenues in opinion mining and sentiment analysis. *IEEE Intell. Syst.* **2013**, *28*, 15–21. [CrossRef]
21. Ranjan, S.; Mishra, S. Comparative sentiment analysis of App reviews. In Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 1–3 July 2020; pp. 1–7.
22. Nayebi, M.; Ruhe, G. Asymmetric release planning: Compromising satisfaction against dissatisfaction. *IEEE Trans. Softw. Eng.* **2019**, *45*, 839–857. [CrossRef]
23. Jo, Y.; Oh, A.H. Aspect and sentiment unification model for online review analysis. In Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, Hong Kong, China, 9–12 February 2011; pp. 815–824.
24. Guzman, E.; Maalej, W. How do users like this feature? a fine grained sentiment analysis of app reviews. In Proceedings of the 2014 IEEE 22nd International Requirements Engineering Conference (RE), Karlskrona, Sweden, 25–29 August 2014; pp. 153–162.
25. Shuyo, N. Language Detection Library for JAVA. Available online: <https://github.com/shuyo/language-detection> (accessed on 19 April 2021).
26. Palomba, F.; Salza, P.; Ciurumelea, A.; Panichella, S.; Gall, H.; Ferrucci, F.; De Lucia, A. Recommending and localizing change requests for mobile apps based on user reviews. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, 20–28 May 2017; pp. 106–117.
27. Sarro, F.; Al-Subaih, A.A.; Harman, M.; Jia, Y.; Martin, W.; Zhang, Y. Feature lifecycles as they spread, migrate, remain, and die in app stores. In Proceedings of the 2015 IEEE 23rd International Requirements Engineering Conference (RE), Ottawa, ON, Canada, 24–28 August 2015; pp. 76–85.
28. Banerjee, S.; Bhattacharyya, S.; Bose, I. Whose online reviews to trust? Understanding reviewer trustworthiness and its impact on business. *Decis. Support Syst.* **2017**, *96*, 17–26. [CrossRef]
29. Zhang, J.; Wang, Y.; Xie, T. Software feature refinement prioritization based on online user review mining. *Inf. Softw. Technol.* **2019**, *108*, 30–34. [CrossRef]
30. Manning, C.; Schütze, H. *Foundations of Statistical Natural Language Processing*; MIT Press: Cambridge, MA, USA, 1999.
31. Bird, S.; Klein, E.; Loper, E. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*; O'Reilly Media, Inc.: Newton, MA, USA, 2009.
32. Cheng, W.; Greaves, C.; Warren, M. From n-gram to skipgram to concgram. *Int. J. Corpus Linguist.* **2006**, *11*, 411–433. [CrossRef]

33. Liang, T.P.; Li, X.; Yang, C.T.; Wang, M. What in consumer reviews affects the sales of mobile apps: A multifacet sentiment analysis approach. *Int. J. Electron. Commer.* **2015**, *20*, 236–260. [\[CrossRef\]](#)
34. Chong, A.Y.L.; Ch'ng, E.; Liu, M.J.; Li, B. Predicting consumer product demands via Big Data: the roles of online promotional marketing and online reviews. *Int. J. Prod. Res.* **2017**, *55*, 5142–5156. [\[CrossRef\]](#)
35. Bouma, G. Normalized (pointwise) mutual information in collocation extraction. In Proceedings of the Biennial GSCL Conference, Potsdam, Germany, 30 September 2009; pp. 31–40.
36. Islam, A.; Inkpen, D. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Trans. Knowl. Discov. Data* **2008**, *2*, 1–25. [\[CrossRef\]](#)
37. Rehurek, R.; Sojka, P. Software framework for topic modelling with large corpora. In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta, Malta, 22 May 2010; pp. 45–50.
38. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; pp. 1–9.
39. Mikolov, T.; Yih, W.T.; Zweig, G. Linguistic regularities in continuous space word representations. In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Atlanta, GA, USA, 9–14 June 2013; pp. 746–751.
40. Chawla, N.V.; Japkowicz, N.; Kotcz, A. Special issue on learning from imbalanced data sets. *ACM SIGKDD Explor. Newsl.* **2004**, *6*, 1–6. [\[CrossRef\]](#)
41. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
42. Maalej, W.; Nabil, H. Bug report, feature request, or simply praise? on automatically classifying app reviews. In Proceedings of the 2015 IEEE 23rd International Requirements Engineering Conference (RE), Ottawa, ON, Canada, 24–28 August 2015; pp. 116–125.
43. Wang, C.; Zhang, F.; Liang, P.; Daneva, M.; van Sinderen, M. Can app changelogs improve requirements classification from app reviews? an exploratory study. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Oulu, Finland, 11–12 October 2018; pp. 1–4.
44. Martens, D.; Maalej, W. Towards understanding and detecting fake reviews in app stores. *Empir. Softw. Eng.* **2019**, *24*, 3316–3355. [\[CrossRef\]](#)
45. Baeza-Yates, R.; Ribeiro-Neto, B. *Modern Information Retrieval*; ACM Press: New York, NY, USA, 1999; Volume 463.
46. Carreno, L.V.G.; Winbladh, K. Analysis of user comments: an approach for software requirements evolution. In Proceedings of the 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18–26 May 2013; pp. 582–591.
47. Fawcett, T. An introduction to ROC analysis. *Pattern Recognit. Lett.* **2006**, *27*, 861–874. [\[CrossRef\]](#)
48. Ho, T.K. Random decision forests. In Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, 14–16 August 1995; Volume 1, pp. 278–282.
49. Waskom, M.L. Seaborn: statistical data visualization. *J. Open Source Softw.* **2021**, *6*, 3021. [\[CrossRef\]](#)
50. Mann, H.B.; Whitney, D.R. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Stat.* **1947**, *18*, 50–60. [\[CrossRef\]](#)
51. Guyon, I.; Elisseeff, A. An introduction to variable and feature selection. *J. Mach. Learn. Res.* **2003**, *3*, 1157–1182.
52. Guyon, I.; Weston, J.; Barnhill, S.; Vapnik, V. Gene selection for cancer classification using support vector machines. *Mach. Learn.* **2002**, *46*, 389–422. [\[CrossRef\]](#)
53. Chen, X.W.; Jeong, J.C. Enhanced recursive feature elimination. In Proceedings of the 6th International Conference on Machine Learning and Applications, ICMLA 2007, Cincinnati, OH, USA, 13–15 December 2007; pp. 429–435.
54. Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
55. Hinton, G.E. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 599–619.
56. Bernard, S.; Heutte, L.; Adam, S. Influence of hyperparameters on random forest accuracy. In Proceedings of the International Workshop on Multiple Classifier Systems, Reykjavik, Iceland, 10–12 June 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 171–180.