*electronics*

*Article*

# Identifying Users and Developers of Mobile Apps in Social Network Crowd

**Ghadah Alamer *, Sultan Alyahya and Hmood Al-Dossari**

Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11574, Saudi Arabia; sualyahya@ksu.edu.sa (S.A.); hzaldossari@ksu.edu.sa (H.A.-D.)
* Correspondence: 438204456@student.ksu.edu.sa

**Abstract:** In the last fifteen years, an immense expansion has been witnessed in mobile app usage and production. The intense competition in the tech sector and also the rapidly and constantly evolving user requirements have led to increased burden on mobile app creators. Nowadays, fulfilling users' expectations cannot be readily achieved and new and unconventional approaches are needed to permit an interested crowd of users to contribute in the introduction of creative mobile apps. Indeed, users and developers of mobile apps are the most influential candidates to engage in any of the requirements engineering activities. The place where both can best be found is on Twitter, one of the most widely used social media platforms. More interestingly, Twitter is considered as a fertile ground for textual content generated by the crowd that can assist in building robust predictive classification models using machine learning (ML) and natural language processing (NLP) techniques. Therefore, in this study, we have built two classification models that can identify mobile apps users and developers using tweets. A thorough empirical comparison of different feature extraction techniques and machine learning classification algorithms were experimented with to find the best-performing mobile app user and developer classifiers. The results revealed that for mobile app user classification, the highest accuracy achieved was ≈0.86, produced via logistic regression (LR) using Term Frequency Inverse Document Frequency (TF-IDF) with N-gram (unigram, bigram and trigram), and the highest precision was ≈0.86, produced via LR using Bag-of-Words (BOW) with N-gram (unigram and bigram). On the other hand, for mobile app developer classification, the highest accuracy achieved was ≈0.87, produced by random forest (RF) using BOW with N-gram (unigram and bigram), and the highest precision was ≈0.88, produced by multi-layer perception neural network (MLP NN) using BERTweet for feature extraction. According to the results, we believe that the developed classification models are efficient and can assist in identifying mobile app users and developers from tweets. Moreover, we envision that our models can be harnessed as a crowd selection approach for crowdsourcing requirements engineering activities to enhance and design inventive and satisfying mobile apps.

**Keywords:** tweets; classification; machine learning; feature extraction; users; developers; mobile apps; crowdsourcing requirements engineering



**Citation:** Alamer, G.; Alyahya, S.; Al-Dossari, H. Identifying Users and Developers of Mobile Apps in Social Network Crowd. *Electronics* **2023**, *12*, 3422. https://doi.org/10.3390/electronics12163422

Academic Editors: Iouliia Skliarova, Jinhyun Kim, Seonah Lee and Suwon Lee

Received: 4 July 2023
Revised: 8 August 2023
Accepted: 10 August 2023
Published: 12 August 2023



## 1. Introduction

Owing to the continuous progression of communication and computer-based technologies, various paradigms have emerged, such as mobile and social computing. Indeed, the overwhelming transformation in software applications and their communicative and dynamic means of use have emphasized the necessity for extending user involvement to a global level throughout the phases of requirements engineering (RE) [1,2]. The practice of narrowing user involvement to a set of selected representatives, as performed by conventional requirements engineering, may not be reliable enough to keep pace with these new computing paradigms [2,3]. Currently, applications are surrounded by a community of stakeholders that could possibly range from hundreds to even millions of heterogenous and distributed users [1,4].

A promising method for scaling user involvement is employing the principle of crowdsourcing (CS), which has potential in a vast range of requirements engineering activities [2,5]. A recently emerged approach called Crowd-Based Requirements Engineering (CrowdRE) has attracted attention from researchers in the discipline of requirements engineering. CrowdRE aims to harness the power of the crowd in requirements engineering. In essence, it is a term pertaining to all the automated activities of requirements engineering engaging the crowd, and it was first introduced by Groen et al. [6]. In fact, identifying appropriate sources and collecting the right needs, which are the main tasks in the requirements elicitation process, are non-trivial tasks [7]. Keeping in consideration that contemporary applications' stakeholders may be a crowd such as users of mobile apps, it is thus challenging to identify [1,4] and recruit sufficient representatives from the substantial pool of users [8]. Moreover, it is of paramount importance to find a proper crowd of stakeholders and gather their needs and requirements to establish a solid basis within the early phases of the mobile application development process [8,9]. Therefore, crowd selection is one of the quality control mechanisms that can be applied when crowdsourcing. Crowd selection focuses on selecting a well-suited segment of the crowd to contribute in a crowdsourcing task [10], which can be leveraged when crowdsourcing RE activities.

Mobile app users express their thoughts and opinions in various channels such as app publishing platforms (e.g., Apple App Store and Google Play Store) and also social media platforms (e.g., Twitter) [11]. Apparently, when observing tweets on Twitter, we realized that numerous Twitter users tend to voice their interests and opinions about mobile apps they have used, and might also recommend others to use them. In addition, mobile apps developers are keen to post mobile apps they have developed on Twitter. According to that, users and developers of mobile apps could be the best participants to be nominated when crowdsourcing RE tasks for mobile apps. This is due to the knowledge they have gained by using or developing a mobile app. For instance, when crowdsourcing requirements elicitation for a certain mobile app, users and developers of similar mobile apps would most probably contribute with valuable requirements than a randomly selected crowd. Hence, Twitter can be considered as a suitable stage for identifying and selecting a crowd to crowdsource mobile app requirements engineering tasks, where both users and developers of mobile apps are accessible.

This research study extends our position paper in [12] that proposed a crowd selection approach that mainly consists of two parts: mobile app user and developer identification and then mobile app domain identification. The approach aims to select users and developers of mobile apps in a certain domain to crowdsource mobile app RE (e.g., requirements elicitation). In this research, we worked on the first part of the proposed approach, which is mobile app user and developer identification via Twitter. After surveying the landscape of research in the area of crowdsourcing RE, we have found that there is no method or strategy particularly designed to overcome this gap in the area. Therefore, we work on training two classification models, one for identifying mobile app users and another for identifying mobile app developers. To find the best-performing model, we conduct a comprehensive comparison between a selected number of feature extraction techniques and classification machine learning algorithms using tweets from Twitter that were collected for this study. Bag-of-Words (BOW), Term Frequency Inverse Document Frequency (TF-IDF), N-gram (unigram, bigram and trigram), Word2vec, BERT Base and BERTweet were all used for feature extraction. In addition, the following set of well-known classification machine learning algorithms were used for training the models: Support Vector Machine (SVM), logistic regression (LR), random forest (RF), Decision Tree (DT), multi-layer perception neural network (MLP NN) and Naïve Bayesian (NB). This article presents related work (Section 2), methods and results which describe all the phases for building the models and showcase the produced results (Section 3) and, finally, conclusions and future work (Section 4).

## 2. Related Work

To the extent of our current knowledge, only the related studies discussed in this section have investigated crowd identification and selection to crowdsource requirements engineering activities. The studies have applied different techniques and approaches to handle the crowd identification process. StakeNet was introduced by Ling Lim et al. [13], which is a stakeholder analysis method that could cope with wide-scale software projects having a crowd up to hundreds of stakeholders. The method employs a variety of social network measures to identify and prioritize a crowd of stakeholders based on the impact level they have on a software development project. To construct a social network of stakeholders, stakeholders are asked to recommend other stakeholders who also recommend others—this is called the snowballing technique—through a face-to-face survey until a social network is generated. Additionally, the study [14] complements the previous one by developing an enhanced tool called StakeSource, which supports StakeNet in its identification and prioritization process. In the proposed StakeNet method [13], experts are required to manually approach stakeholders so that they recommend others, which is a tedious task, especially for large software projects. Alternatively, the StakeSource [14] web-based tool has utilized crowdsourcing instead to crowdsource the process of stakeholder analysis and alleviate the workload on experts by only prompting them to start off with an initial list of stakeholders. Furthermore, StakeSource2.0 [15] extended the work performed on the stakeholder analysis tool StakeSource [14]. StakeSource2.0 is an improved tool that uses the following three main techniques: crowdsourcing, social network analysis and collaborative filtering to conduct the identification and prioritization of stakeholders along with their requirements. Moreover, StakeRare [16] made use of StakeSource2.0 for identifying and prioritizing the crowd of stakeholders and their elicited requirements. From another perspective, Mughal et al. [17] has pointed out a shortcoming in the previous studies which is the in-group bias issue. They have defined the in-group bias as the situation when stakeholders lean towards recommending others with whom they have a positive rapport. This bias might lead to inaccurate identification and the prioritization of stakeholders. Hence, for the sake of reducing this issue, the authors sought to propose a social network-based approach that proved its effectiveness.

Studies that were previously discussed have leveraged social network analysis for identifying a suitable crowd depending on the impact level they have on a software project. However, the studies [8,18,19] have rather based the crowd identification process on the domain knowledge the crowd possesses. Wang et al. [8] stated that requirements acquisition tasks require participants with certain domain knowledge. Therefore, a framework was presented by the authors which seeks to recruit a crowd of stakeholders that hold certain domain knowledge depending on their spatiotemporal availability using opportunistic network for distributing the task among the crowd. They observed that people that cluster in the same spatiotemporal space are more probably to have the same domain knowledge. In addition, Srivastava and Sharma [19] have used crowdsourcing for requirements elicitation for the MyERP application. LinkedIn, a social media platform, was used as a source to find well-suited stakeholders with expertise in ERP by utilizing a web-based crawling tool to obtain LinkedIn users who have added ERP to their skill set. Furthermore, LinkedIn was also used by Lim et al. [18] as a source to reach and find an inaccessible and hidden crowd of stakeholders of business-to-business solutions in large firms. The authors have designed AdvisorNet, which is a systematic method. AdvisorNet can help in bridging the gap between the outside world and the hidden stakeholders in a target organization through a set of selected advisors who are found using hypothesis-driven social media search.

On the other hand, there are studies that have proposed an identification process by focusing on the crowd's domain of interest, such as in [4,20]. A study conducted by Lim et al. [20] has utilized the Twitter social networking site as a source to identify a crowd to carry out complex tasks such as the process of eliciting requirements. A tool named PseudoGravity was proposed to form social presence in a target audience's domain of interest. This is achieved by having PseudoGravity control a Twitter account. The

tool aims to auto-generate content to attract the crowd and increase their engagement and participation in a task. The content is generated using external public sources, and also, a set of rules were applied to ensure relevant and suitable content are generated for distribution in the form of tweets. In a study by Kolpondinos and Glinz [4], the authors have utilized multiple social media platforms as a source for finding a well-suited crowd. A stakeholder identification approach was designed to identify an interested crowd of stakeholders beyond the reach of an organization and invite them to participate in requirements engineering activities. To spot a potential crowd of stakeholders for the SmaWoMo system, the authors have selected a number of online channels (e.g., Facebook and LinkedIn) for distributing the questionnaire, they have and used search keywords to find groups in these channels that are expected to show interest in the SmaWoMo system. In addition, player types were used which are similar to personality traits to create persona-based advertisements tailored to a potential crowd of stakeholders.

Furthermore, Condori-Fernandez et al. [21] have mentioned ResearchGate, LinkedIn and Twitter as possible sources to assist in identifying a crowd of experts. The identification of a crowd of experts is considered as one of the phases of the adapted methodology they have proposed, which is based on the nichesourcing method Accurator [22]. Their proposed methodology seeks to analyze sustainability requirements along with their dependencies. The methodology consists of multiple phases where one of the phases that took place in the initial part of the methodology is discovering community niches to choose potential experienced contributors. Additionally, persona-based methods have been applied by Alvertis et al. [23] for crowd identification, wherein anonymous personas and persona builder were used. Via their proposed persona builder, software teams can either construct a persona from scratch by defining specific characteristics and inserting input parameters or by using predefined personas or reusing previously created and stored personas. By using these personas, a crowd possessing characteristics which match these personas is produced. Subsequently, the produced crowd can participate in tasks like requirements elicitation. Moreover, Guzman et al. [24], as part of their study, have worked on identifying stakeholder groups from tweets. They have defined three stakeholder groups, which are technical, non-technical and general public, and the tweets they have collected were about 30 popular desktop and mobile applications. The highest precision values achieved were as follows: technical stakeholder classifier, 0.54; non-technical stakeholder classifier, 0.77; general public classifier, 0.78.

Overall, several limitations have been noticed in the previously discussed studies. To discover an interested crowd of stakeholders, the strategy designed by Kolpondinos and Glinz [4] was conducted in a manual manner using specific search keywords. Therefore, an entire manual search would be required for every software project when adopting such a strategy. The same applies to Lim et al. [18], where they have proposed AdvisorNet, an unautomated method to perform a step-by-step search on the LinkedIn social site. To boost the method's scalability and effectiveness, it should be highly automated. Additionally, relying on spatiotemporal availability as a basis to select a suitable portion of the crowd, as performed by [8], might not be an accurate indicator to rely upon when selecting the crowd.

Alvertis et al. [23] applied persona-based approaches to identify a suitable crowd, where personas are used to match with crowd members that hold a defined set of characteristics. Few details have been clarified about the matching process, and neither did they specifically declare the crowd's source and their profiles according to which matching is performed. In the same way, in the methodology proposed for nichesourcing by Condori-Fernandez et al. [21], the crowd identification phase has not been described in depth. Details have not been provided about if it would possibly be automated or conducted manually. Moreover, utilizing social network analysis as an approach for crowd identification, as performed in the studies [13–17], is practically more beneficial with a well-connected crowd of stakeholders [20], and there is a high chance that this may not occur in many cases.

Therefore, we fill the gaps by developing an automated approach that uses machine learning algorithms and natural language processing techniques. This research endeavors to build a crowd selection strategy that identifies two critical roles in any software development process: users and developers. Particularly, we focus on mobile apps due to their wide prevalence and since they are an integral part of our daily lives. Furthermore, the crowd is selected from publicly available tweets from Twitter where crowd members are not necessarily connected.

## 3. Methods and Results

Our aim is to build a mobile app user classification model which can distinguish mobile app users from tweets, and we also aimed to build a mobile app developer classification model which can distinguish mobile app developers from tweets. Figure 1 illustrates our methodology, which has been applied to develop the models.

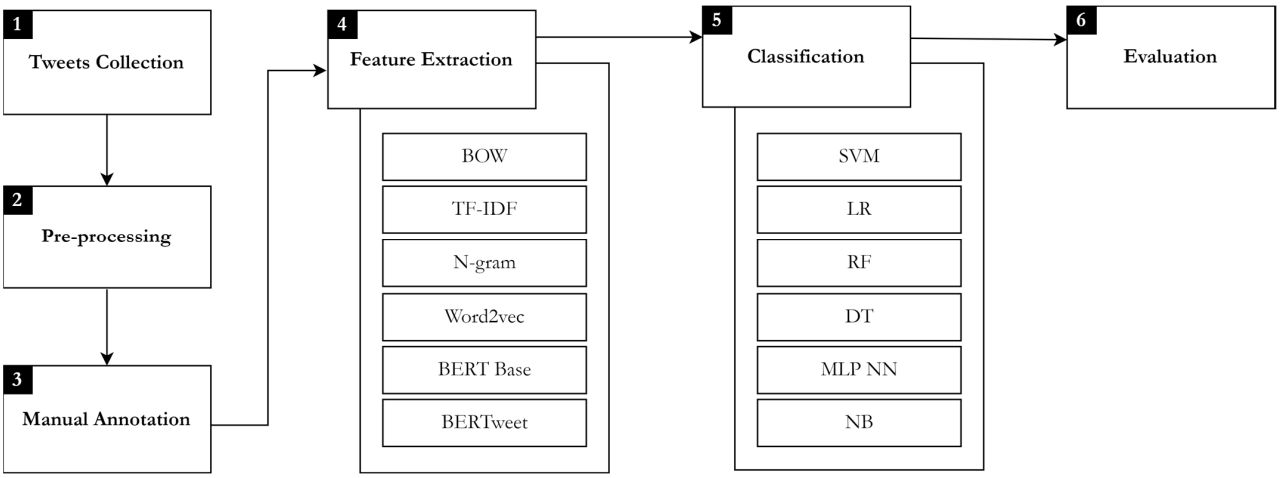


**Figure 1.** Research methodology for building mobile app user and developer classification models.

### *3.1. Tweets Collection*

A total of 266,618 tweets were collected through the Twitter API. Several run instances of three queries were carried out between September 2021 and June 2022 in separated periods. In fact, the reason behind the large number of retrieved tweets is that the collection process involved all tweets, including duplicate tweets and retweets. In addition, we focused on mobile app-related tweets and, more specifically, Apple and Android apps due to their wide prevalence among mobile users. Unlike studies such as [24], where they have used apps names as keywords to retrieve app-related tweets, we adopt a quite different approach to theirs. Indeed, we have noticed that tweets involving links pointing to mobile apps in the app store are a good target to compile our dataset and serve our research objectives. The text string "apps.apple" is a main segment of any Apple mobile app link to the app store; likewise, the text sting "play.google" is a main segment of any Android mobile app link. Accordingly, during tweets collection, both text strings were used as a search filter. Moreover, during the collection process, tweets from accounts that had fewer than 5 followers were filtered out to reduce spam. Three keyword-based search queries were run, namely $q_1$, $q_2$ and $q_3$, using AND and OR search operators.

$(q_1)$ apps.apple OR play.google AND this app

The intention behind writing $q_1$ in this from is to retrieve tweets that contain a link to an apple or android app or even both, and it should contain the words "this" and "app" regardless of the order of their occurrence. We added keywords "this" and "app" to eliminate tweets that contain only links and also to reduce the probability of retrieving non-English tweets. Additionally, we are specifically interested in tweets with content, and words like "this" are most likely used when describing apps. Moreover, the following are query $q_2$ and $q_3$, which were run in addition to query $q_1$:

($q_2$) apps.apple OR play.google AND developed OR launched OR released OR implemented OR wrote OR created OR published OR coded OR designed OR worked

($q_3$) apps.apple OR play.google AND develop OR launch OR release OR implement OR write OR create OR publish OR code OR design OR work OR developing OR launching OR releasing OR implementing OR writing OR creating OR publishing OR coding OR designing OR working

The tweets were exported into a CSV file format. In addition to the tweet text attribute which will be used for training the classification models, other attributes were collected such as the following: username, name and bio of the user who posted the tweet.

### 3.2. Tweets Pre-Processing

The collected tweets have a very large number of retweets, replies, duplicate, spam, unrelated and unclear tweets. Therefore, a large portion of the dataset has been filtered out, ending with a total of 21,381 tweets ready for annotation and for training the models. The tweets have been through stages of cleaning, and to decide what cleaning and preprocessing techniques to apply, the tweets were manually explored in advance. Both manual and automatic methods were performed for cleaning and pre-processing using Pandas, Tweet-preprocessor and NLTK Python libraries.

The first step in the pre-processing stage was removing retweets and duplicates. Our dataset has a large number of retweets and duplicates that constitute more than 60% of the collected tweets. This might be due to the queries designed for tweets collection and also due to the search keywords used in these queries. In addition, this is not surprising since we have targeted tweets involving links to mobile apps during collection, which might increase such types of tweets. To reduce uninformative tweets, filtering conditions have been identified using a set of selected accounts' usernames and keywords that would most probably indicate that the tweet is uninformative to our concern. We have eliminated tweets from accounts that post unrelated content and post very frequently, system-generated tweets (e.g., news, music), tweets that lack clear context, unrelated tweets, spam tweets that mention strings indicating spam (e.g., earn money, earn cash), non-English tweets, hashtag-only tweets and tweets less than two words long. As a matter of fact, short tweets can reveal information that help in recognizing the role of the tweet author (user or developer). Due to that, we have decided not to completely neglect short tweets.

Moreover, to further clean the tweets, we have used Tweet-preprocessor Python Library. This library is written particularly for cleaning tweets data to remove mentions, hashtags, URLs, numbers, emojis, smileys and reserved words (RT and FAV). We set the options to remove mentions, URLs, numbers, emojis and smileys. We purposefully kept hashtags since we believe they are informative parts of a tweet and could provide important cues that are useful in training our classification models. Further cleaning and normalization were performed using NLTK and stop words Python libraries. Stop words were removed by customizing the stop words list through excluding pronouns (e.g., I, my, our, we, myself, ours, ourselves, own, by, me, I'll, I've, I'm, we've, we'll, we'd and we're). This was performed to keep them in the tweets which we find crucial when building our classification models. Furthermore, we removed punctuation, lowercased letters and performed tokenization of tweets using word tokenizer and stemming using Porter Stemmer from NLTK library.

### 3.3. Annotation

The annotation phase we followed consists of three major steps starting with content analysis performed by analyzing and annotating a selected sample of tweets from our collected dataset to define the categories for tweet annotation. Secondly, we continue to manually annotate the tweets using the categories that were defined during content analysis. Lastly, annotations were aggregated to produce a final label for each tweet.

#### 3.3.1. Content Analysis

The manual content analysis of the tweets helped in getting aquatinted with the nature of the dataset and determine the categories the tweets belong to. In addition, analyzing and understanding the dataset helped in designing an accurate and consistent annotation guideline for annotators. Manual content analysis was performed on a sample of 2836 tweets from different parts of the dataset produced by query $q_1$. When annotating each tweet, its content has been analyzed to determine the role of a tweet's author from a tweet's content, whether it is a mobile app user or developer. The two main categories are *user* and *developer*; however, additional ones were recognized, such as news, quotes, fan groups, show advertisements and other unrelated tweets. Therefore, we plan to add another category, i.e., *others*, that involves all tweets that do not belong to the user or developer category.

After multiple rounds of analysis and readings of the tweets, we have settled on three main categories: user, developer and others. Since our focus is building models to discover users and developers of mobile apps, annotating tweets into these three categories fits our need. These categories were also suitable for annotating tweets collected using $q_2$ and $q_3$.

#### 3.3.2. Manual Annotation

To enhance the quality of the annotation, the 21,381 tweets comprising our dataset were annotated by three annotators. All three annotators had a background in software engineering to ensure they understood the computer-related terminologies (e.g., flutter, java, beta, etc.) they face during annotation. Each tweet was annotated by all three annotators into one of three categories (user, developer or others), and it took ≈2 months to complete the whole annotation process. The annotators were given a description of the task, a guideline to follow and samples of annotated tweets for more clarification. They were given the tweets along with the bio, name and username. Moreover, they were asked to focus on and read each tweet and check other data once needed to help in decision making when annotating. To keep up with their progress, they were asked to provide samples of their work every now and then. This was to ensure they understood the task handed to them, and they were given feedback accordingly. In addition, they were encouraged to ask whenever needed in case there was any confusion. A description of each category is shown in Table 1, which was also handed to the annotators.

**Table 1.** Descriptions of categories.

| Category | Description |
|---|---|
| User | Includes tweets indicating that they are posted by a user of a mobile app. |
| Developer | Includes tweets indicating that they are posted by a developer of a mobile app. A developer of a mobile app can be an individual developer, company, development studio or a team. |
| Others | Includes unrelated tweets such as and not limited to:<br>• Fan groups (e.g., ask for votes for a celebrity).<br>• Advertisements for a show, series, episode, drama, match, radio show, interview.<br>• News.<br>• Quotes.<br>• Winners announcements for a contest (with no details about the app).<br>• Books releases. |

#### 3.3.3. Annotation Aggregation

A simple majority vote technique was used to find the final aggregated category, where the annotation that receives the greatest number of votes was set as the final aggregated annotation [25]. The final category was one of the following three: user, developer or others. We defined three agreement levels: 3, 2 and 0. Table 2 illustrates the definition of each agreement level and the final number of tweets and percentage in each level in our dataset.

**Table 2.** Distribution of tweets across agreement levels.

| Agreement Level | Description | Num. of Tweets | Percentage |
|---|---|---|---|
| 3 | When all three annotators annotate a tweet with the same label | 10,742 | 50.2% |
| 2 | When two annotators annotate a tweet with the same label | 9230 | 43.2% |
| 0 | When none of the annotators agree on the same label | 1409 | 6.6% |
| | Total Dataset | 21,381 | 100% |

It is evident that all three annotators agreed on the same annotation for more than half of the dataset, and at least two annotators agreed on the same annotation for ≈93.4% of the dataset. The annotators disagreed on ≈6.6% of the dataset (1409 tweets), which is a reasonable percentage since the nature of the tweets requires concentration, and hence, disagreements and unintentional mistakes are expected. To ensure quality annotation, a large part of the tweets with agreement level = 2 were reviewed, and mistakes were corrected. In addition, disagreements were reconciled and further reviewed. The number of tweets in each class, after annotation aggregation, is depicted in Table 3. It is worth mentioning that because of the hard work completed in the pre-processing stage, a relatively smaller number of tweets belonging to the others class resulted.

**Table 3.** Number of tweets in each class.

| Class | Number of Tweets |
|---|---|
| User | 10,768 |
| Developer | 6883 |
| Others | 3730 |
| Total | 21,381 |

The dataset is annotated using three classes. However, since two models will be built, one for identifying users of mobile apps and another one for identifying developers of mobile apps, classes are merged in a way suitable for each model. For the user classification model, all tweets annotated as belonging to the developer or others classes are merged into one class named *not user*. On the other hand, for the developer classification model, all tweets annotated as belonging to the user or others classes are merged into one class named *not developer*. Eventually, we ended up with two datasets with the same tweets, albeit with different categorizations, which shall be used for building two binary classification models.

The dataset prepared for the user classification model was balanced resulting with 10,768 tweets belonging to the user class and 10,613 tweets belonging to the not user class. However, merging the user and others classes for the developer classification model dataset has indeed created an unbalanced distribution of the resulting classes developer and not developer. The developer class has 6883 tweets, and the not developer class has 14,498 tweets, making the developer class ≈ 53% smaller than the not developer class, which is a large difference that cannot be overlooked. To address this issue, we have performed undersampling of the majority class since the oversampling technique leads to overfitting [26]. Random tweets in the majority class of not developer have been eliminated until reaching 8000 tweets to achieve a balanced distribution of tweets among both classes. Figures 2 and 3 illustrates the process for preparing the datasets for building a mobile app user classification model and a mobile app developer classification model, respectively.

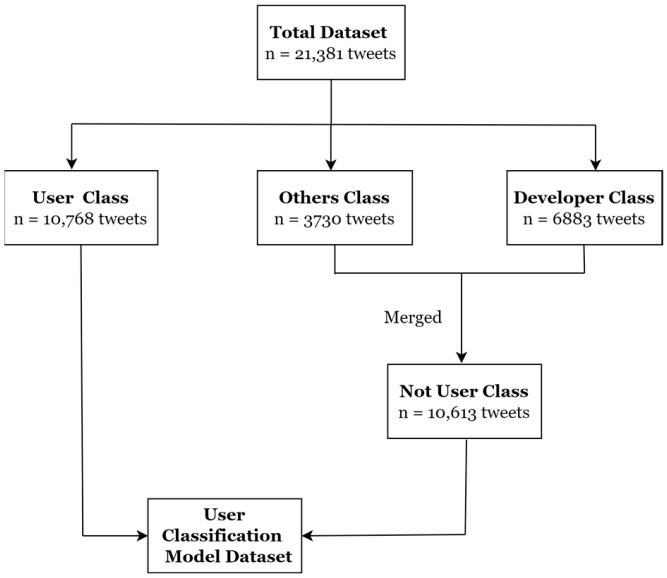


**Figure 2.** Preparing dataset for building mobile app user classification model.

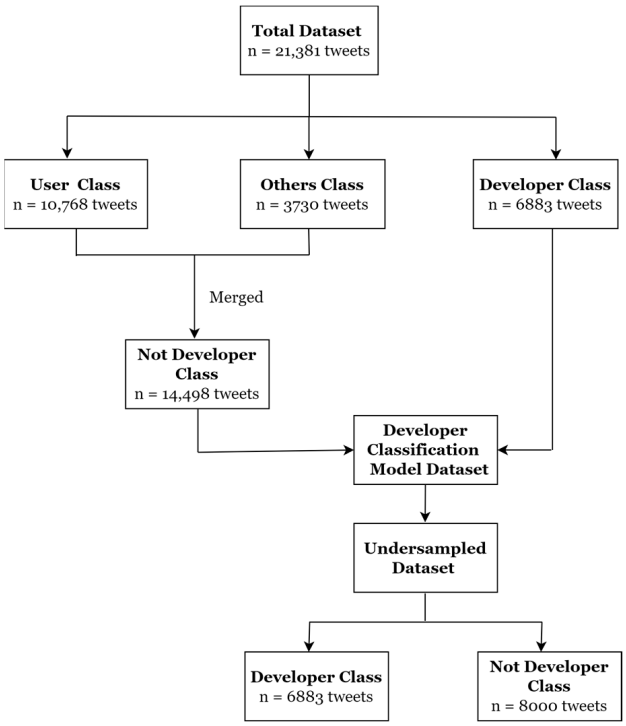


**Figure 3.** Preparing dataset for building mobile app developer classification model.

We have two versions of the dataset that will be used for building the user classifier and developer classifier. We denote the dataset used for user classification as U-version dataset and the dataset used for developer classification as the D-version dataset. Figure 4a shows the distribution of the classes (user, developer and others) in the original dataset, Figure 4b shows the classes (user and not user) in the U-version dataset, and Figure 4c shows the classes (developer and not developer) in the D-version dataset.

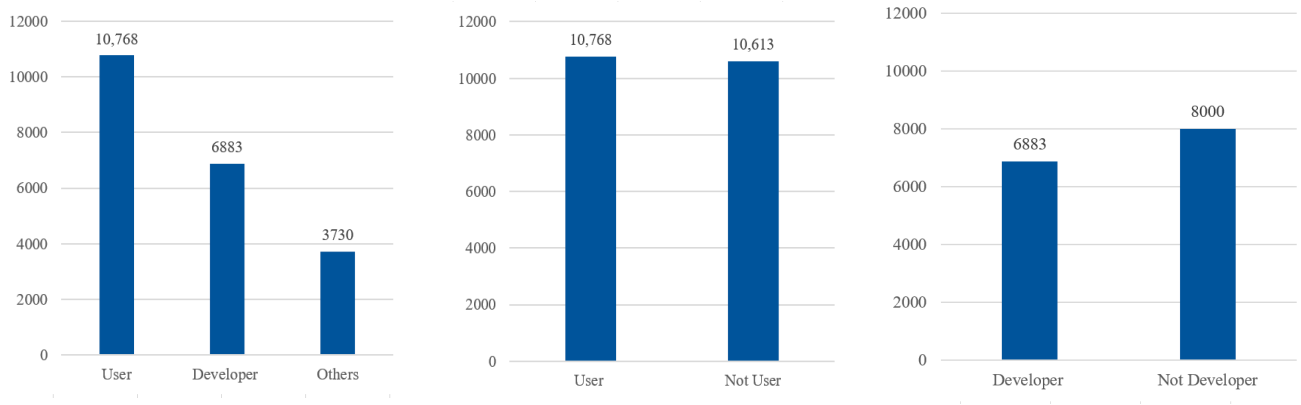


**Figure 4.** Distribution of tweets among the classes in the three built datasets: (**a**) Classes in original dataset; (**b**) Classes in U-version dataset; (**c**) Classes in D-version dataset.

### *3.4. Feature Extraction*

To build mobile app user and developer classification models, several feature extraction techniques have been experimented with using several classification algorithms. A set of well-known feature extraction techniques were selected and applied, which are the following: Bag-of-Words (BOW), Term Frequency Inverse Document Frequency (TF-IDF), N-gram, Word2vec, BERT Base uncased and BERTweet. All these feature extraction techniques were applied with the same settings when building both user and developer classification models.

The Scikit-learn Python library was utilized to perform BOW and TF-IDF. Certain settings were applied that would fit our requirements and yield satisfying results. To remove too infrequently occurring words, words that appear in fewer than 4 tweets were ignored. In addition, the maximum number of features was set to 10,000. Moreover, it is worth mentioning that single characters such as the pronoun "I", which we purposefully have kept in our dataset, is important to consider. Therefore, to ensure single characters are considered during vectorization, the token pattern was set to a regex that accepts single characters. The N-gram technique was used in conjunction with the BOW and TF-IDF techniques. The N-gram range was set in three different ways, where each setting was experimented with separately (for easier readability, ranges are shortened to (x,y) when used in graphs and tables). The lower and upper values were set to the following: (1,2) for unigrams and bigrams, (1,3) for unigrams, bigrams and trigrams, and (2,3) for bigrams and trigrams.

In addition to the frequency-based feature extraction approaches discussed above, pre-trained word embeddings and language models, namely Word2vec and BERT models, were utilized as feature extraction techniques. The pre-trained models were used by preserving the Word2ve and BERT model architectures and freezing the parameters, and hence, the features were extracted directly from the pre-trained Word2vec and BERT models. The feature-extraction approach was adopted rather than finetuning the models since the latter requires large amount of data [27] and is a time-consuming process [28]. For that, the feature-extraction approach is best suited for this research, where interestingly, it was proven that it can indeed achieve results close to finetuning [28,29].

The Genism Python library was utilized to load the pre-trained Word2vec model from the publicly available file "GoogleNews-vectors-negative300.bin" [30]. The model was harnessed to generate word embeddings for words in the tweets, and the mean of the words embeddings for each tweet is calculated. Moreover, two variants of BERT were selected in this experiment, which are the BERT Base uncased pretrained model [28] and the BERTweet pretrained model [31] from Hugging Face. The Pytorch framework and Transformers Python library were used to load and deal with the pre-trained BERT models. To extract vector embedding for each tweet, the [CLS] token from the last hidden state in BERT models was utilized. The [CLS] token is a special classification token which

involves contextualized vector embedding of the entire text of the tweet [32]. Furthermore, when using pre-trained Word2vec and the two variants of BERT models, the tweets were unstemmed since stemming can change the meaning of words, and this is not necessary since pre-trained models were trained on raw text. Additionally, since BERT models are concerned with context, all stop words were kept, since they could provide the context of the entire tweet.

In essence, the following feature extraction techniques have been performed on the tweets: BOW, BOW + N-gram (1,2), BOW + N-gram (1,3), BOW + N-gram (2,3), TF-IDF, TF-IDF + N-gram (1,2), TF-IDF + N-gram (1,3), TF-IDF + N-gram (2,3), Word2vec, BERT Base uncased and BERTweet. The extracted vectors are then fed into six classification algorithms discussed in the following section.

### *3.5. Classification, Evaluation and Results*

A comprehensive comparison between different classification algorithms was conducted to find the most robust model. To train and test the models, six well-known classification algorithms were experimented using Scikit-learn Python library. The following are the classification algorithms: Support Vector Machine (SVM), logistic regression (LR), Decision Tree (DT), random forest (RF), Naïve Bayesian (NB) and multi-layer perception neural network (MLP NN).

Two evaluation metrics were used to showcase the performance results of the developed models, which are accuracy and precision. In addition to accuracy, we chose precision since we are interested in finding a classification model that is precise and exact in detecting a user or developer of a mobile app. Precision is a metric that is concerned with the exactness of the classification model [33]; hence, it is a suitable metric that shall fit this research context. The dataset was split into training and testing portions, where 25% of the dataset was for testing, and the remaining 75% was for training the classifiers. This split is a commonly used train/test split percentage in testing machine learning classifiers, where an adequate portion of the dataset is dedicated for testing to produce reliable results. Each classification algorithm was experimented with each feature extraction technique. The combinations of experiments are shown in Table 4, which are conducted on both the U-version dataset and D-version dataset to find the most efficient mobile app user and developer classification models.

**Table 4.** Combination of experiments.

| | **LR** | **SVM** | **RF** | **DT** | **NB** | **MLP NN** |
|---|---|---|---|---|---|---|
| **BOW** | LR + BOW | SVM + BOW | RF + BOW | DT + BOW | NB + BOW | MLP NN + BOW |
| **BOW + (1,2)** | LR + BOW + (1,2) | SVM + BOW + (1,2) | RF + BOW + (1,2) | DT + BOW + (1,2) | NB + BOW + (1,2) | MLP NN + BOW + (1,2) |
| **BOW + (1,3)** | LR + BOW + (1,3) | SVM + BOW + (1,3) | RF + BOW + (1,3) | DT + BOW + (1,3) | NB + BOW + (1,3) | MLP NN + BOW + (1,3) |
| **BOW + (2,3)** | LR + BOW+ (2,3) | SVM + BOW + (2,3) | RF + BOW + (2,3) | DT + BOW + (2,3) | NB + BOW + (2,3) | MLP NN + BOW + (2,3) |
| **TF-IDF** | LR + TF-IDF | SVM + TF-IDF | RF + TF-IDF | DT + TF-IDF | NB + TF-IDF | MLP NN + TF-IDF |
| **TF-IDF + (1,2)** | LR + TF-IDF + (1,2) | SVM + TF-IDF + (1,2) | RF + TF-IDF + (1,2) | DT + TF-IDF + (1,2) | NB + TF-IDF + (1,2) | MLP NN + TF-IDF + (1,2) |
| **TF-IDF + (1,3)** | LR + TF-IDF + (1,3) | SVM + TF-IDF + (1,3) | RF + TF-IDF + (1,3) | DT + TF-IDF + (1,3) | NB + TF-IDF + (1,3) | MLP NN + TF-IDF + (1,3) |
| **TF-IDF + (2,3)** | LR + TF-IDF + (2,3) | SVM + TF-IDF + (2,3) | RF + TF-IDF + (2,3) | DT + TF-IDF + (2,3) | NB + TF-IDF + (2,3) | MLP NN + TF-IDF + (2,3) |
| **Word2vec** | LR + Word2vec | SVM + Word2vec | RF + Word2vec | DT + Word2vec | NB + Word2vec | MLP NN + Word2vec |
| **BERT Base** | LR + BERT Base | SVM + BERT Base | RF + BERT Base | DT + BERT Base | NB + BERT Base | MLP NN + BERT Base |
| **BERTweet** | LR + BERTweet | SVM + BERTweet | RF + BERTweet | DT + BERTweet | NB + BERTweet | MLP NN + BERTweet |

#### 3.5.1. User Classification Models Results

For better illustration, the results of the user classification models are visualized in Figures 5 and 6. Each figure demonstrates the results using one of the two metrics, where Figure 5 shows the accuracy results, and Figure 6 shows the precision results. The results reveal that the highest accuracy achieved for user classification is ≈0.86, which is produced using a logistic regression classification algorithm and TF-IDF with N-gram (unigram,

bigram and trigram) for features extraction. In addition, the highest precision achieved is ≈0.86, which is produced using a logistic regression classification algorithm and BOW with N-gram (unigram and bigram) for feature extraction.

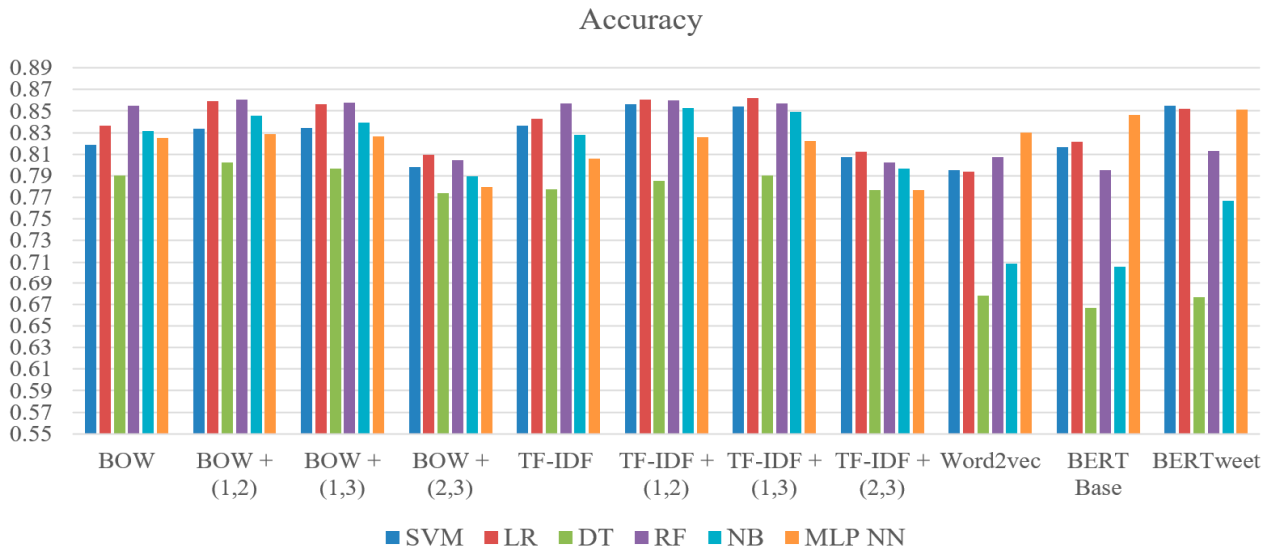


**Figure 5.** Accuracy results of user classification models.

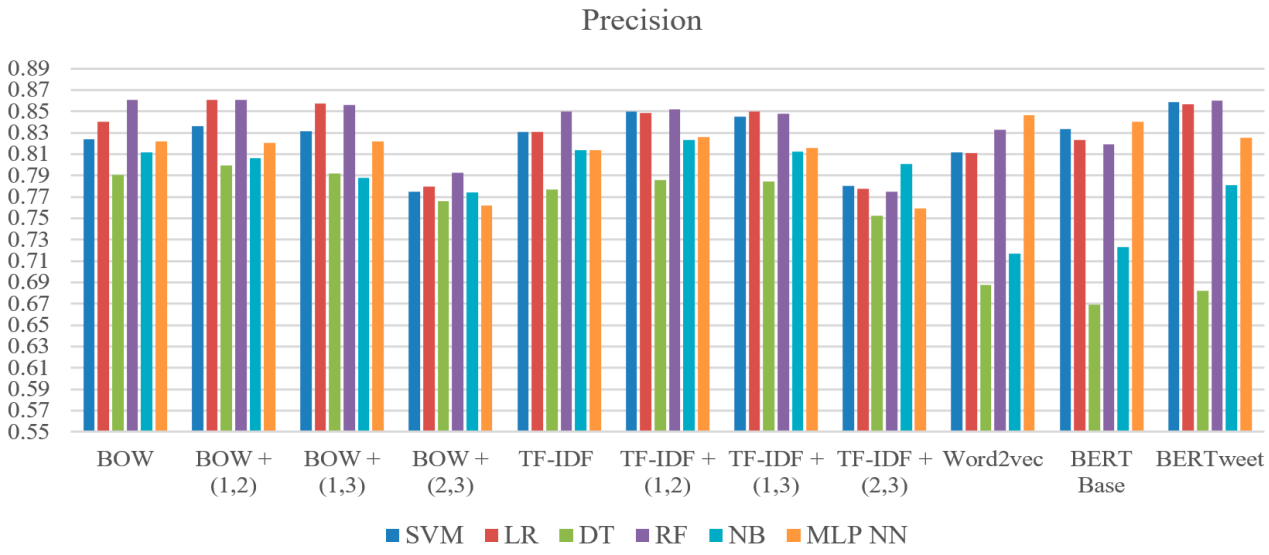


**Figure 6.** Precision results of user classification models.

### 3.5.2. Developer Classification Models Results

In this section, we present the results of the developer classification models. Figure 7 shows the accuracy results, and Figure 8 shows the precision results. The results reveal that the highest accuracy achieved for developer classification is ≈0.87, which is produced using the random forest classification algorithm and BOW with N-gram (unigram and bigram) for features extraction. In addition, the highest precision achieved is ≈0.88, which is produced using the Multi-layer Perception Neural Network classification algorithm and BERTweet for feature extraction.

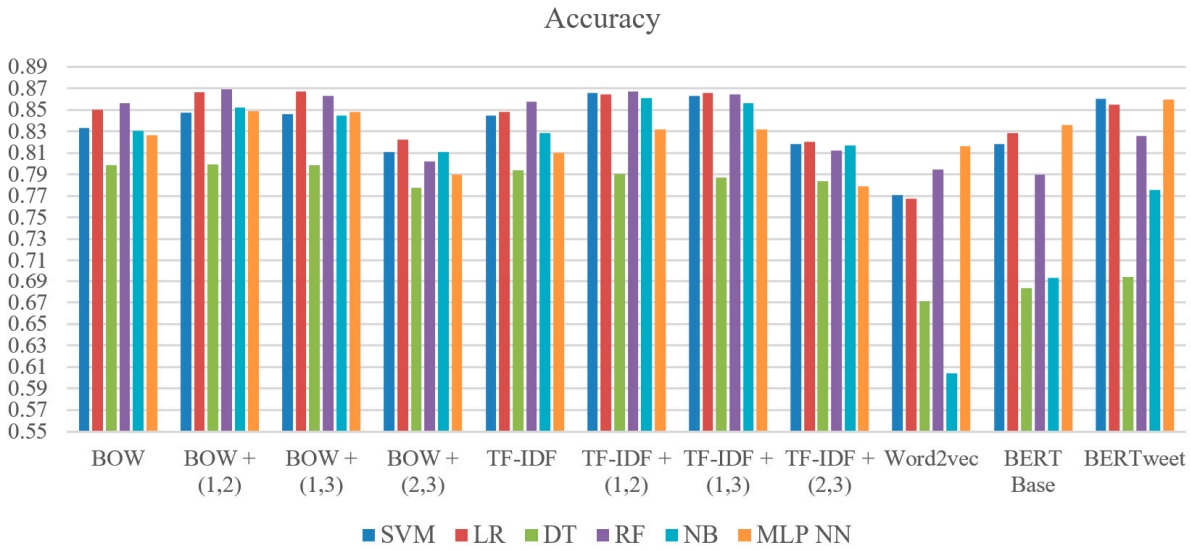


**Figure 7.** Accuracy results of developer classification models.

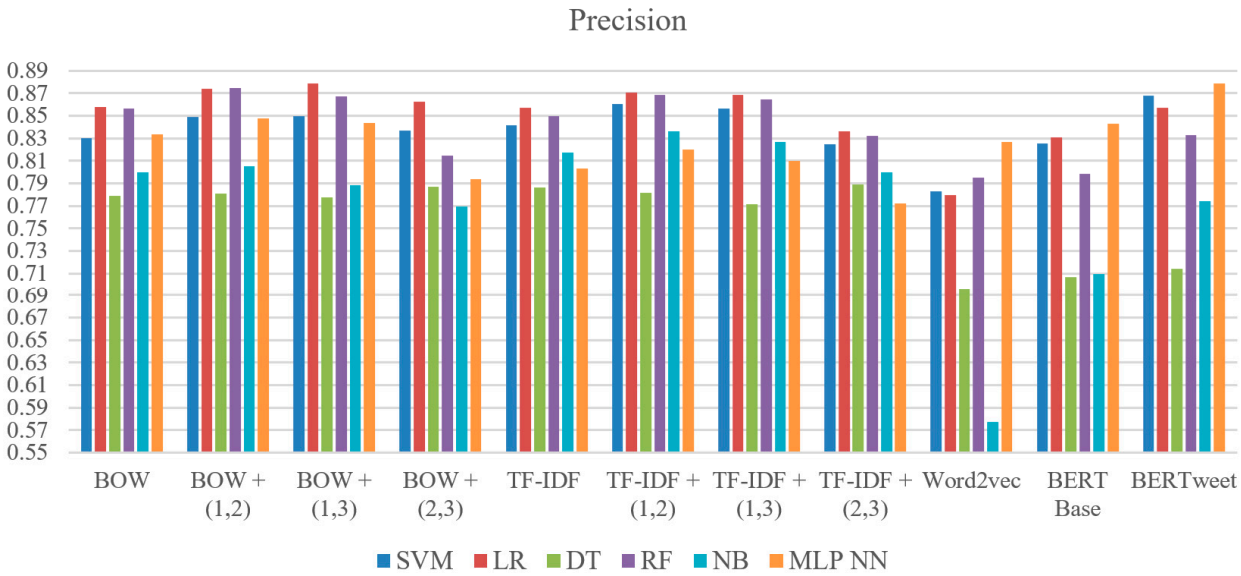


**Figure 8.** Precision results of developer classification models.

### 3.5.3. Findings and Discussion

The user and developer classification models with the highest accuracy and precision are illustrated in Table 5. In addition, we shed the light on some important findings which was observed from the achieved results. Indeed, it was evident that LR and RF performed the best with almost all feature extraction techniques, except when applied on Word2Vec, BERT Base or BERTweet, where MLP NN surpassed.

**Table 5.** User and developer classification model with highest accuracy and precision.

| | | |
|---|---|---|
| **User Classification Model** | | |
| Highest Accuracy | LR + TF-IDF + (1,3) | 0.86 |
| Highest Precision | LR + BOW + (1,2) | 0.86 |
| **Developer Classification Model** | | |
| Highest Accuracy | RF + BOW + (1,2) | 0.87 |
| Highest Precision | MLP NN + BERTweet | 0.88 |

Generally, among all feature extraction techniques, BOW, TD-IDF, BOW + N-gram(1,2), BOW + N-gram(1,3), TD-IDF + N-gram(1,2), TD-IDF + N-gram(1,3) and BERTweet produced the best results. Moreover, utilizing N-gram with BOW and TF-IDF for feature extraction has clearly enhanced the results. On the other hand, the worst-performing classification algorithm was DT, and the worst-performing feature extraction techniques were TF-IDF + N-gram(2,3), BOW + N-gram(2,3) and Word2vec, except when applied with MLP NN, where it performed better.

Among all experimented models, the 15 best-performing user classification models out of 66 in terms of accuracy and precision are shown in Figure 9a,b. Furthermore, the 15 best-performing developer classification models in terms of accuracy and precision are shown in Figure 10a,b. The models are ordered in descending order, where the first model represents the one with highest score.

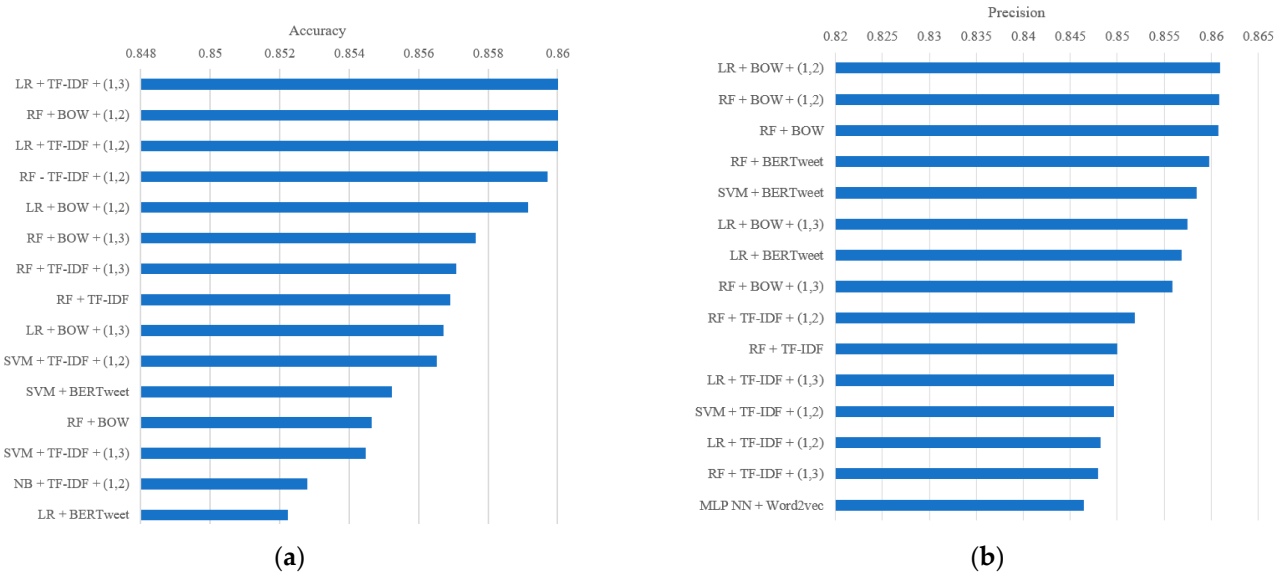


**Figure 9.** Results of top 15 user classification models in terms of accuracy and precision: (**a**) User classification models with highest accuracy; (**b**) User classification models with highest precision.

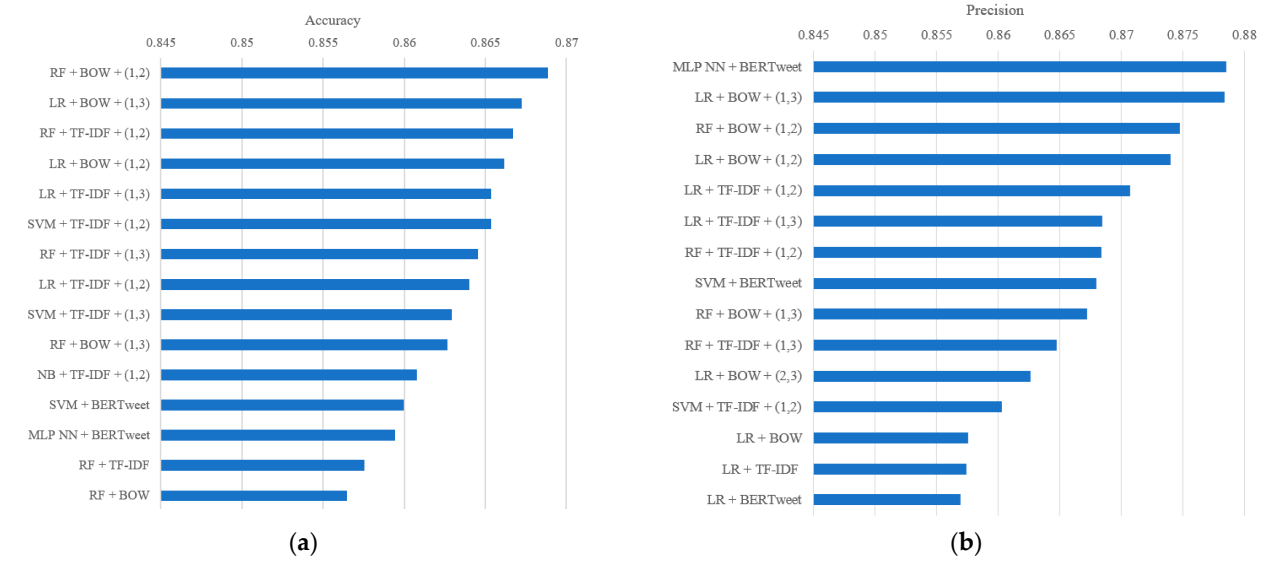


**Figure 10.** Results of top 15 developer classification models in terms of accuracy and precision: (**a**) Developer classification models with highest accuracy; (**b**) Developer classification models with highest precision.

## 4. Conclusions and Future Work

In this study, we have worked on developing two binary classification models, one for identifying mobile app users and another one for identifying mobile app developers from tweets. The results were encouraging and show that the methodology we have undertaken has contributed in producing effectively performing models. The comparison conducted between various feature extraction techniques and classification algorithms was comprehensive enough to find well-performing models, where we have managed to reach an accuracy $\approx 0.86$ and a precision $\approx 0.86$ for mobile app user classification, and we have achieved an accuracy $\approx 0.87$ and a precision $\approx 0.88$ for mobile app developer classification. In addition, we believe that this work contributes to the field of crowdsourcing RE for mobile apps, where our models can identify a suitable segment of the crowd that would participate with more value than a randomly selected crowd. Furthermore, for future work, we plan to continue working on the crowd selection approach by adding another component that focuses on the domains of the used and developed mobile apps mentioned in the tweets. The crowd selection approach will be enhanced using semantic similarity techniques in order to identify users and developers of mobile apps in domains and subdomains similar to the mobile app described by the crowdsourcing requester. This shall assist in finding a suitable crowd of specific mobile app users and developers that can engage in requirements engineering activities such as the process of requirements elicitation for mobile apps. Furthermore, we intend to evaluate the complete approach by crowdsourcing requirements elicitation using our crowd selection approach. In addition, to prove the validity and effectiveness of our proposed approach, we plan on assessing the quality and creativity of the crowdsourced requirements using the approach.

**Author Contributions:** Conceptualization, G.A.; methodology, G.A.; software, G.A.; validation, G.A.; formal analysis, G.A.; investigation, G.A.; resources, G.A.; data curation, G.A.; writing—original draft preparation, G.A.; writing—review and editing, G.A., S.A. and H.A.-D.; visualization, G.A.; supervision, S.A. and H.A.-D. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research is funded by the Deanship of Scientific Research at King Saud University.

**Data Availability Statement:** The dataset is not publicly available.

**Acknowledgments:** The authors would like to thank the Deanship of Scientific Research at King Saud University for funding and supporting this research through the initiative of DSR Graduate Students Research Support (GSR).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Hosseini, M.; Phalp, K.; Taylor, J.; Ali, R. Towards Crowdsourcing for Requirements Engineering. *CEUR Workshop Proc.* **2014**, *1138*, 82–87.
2. Snijders, R.; Dalpiaz, F.; Hosseini, M.; Shahri, A.; Ali, R. Crowd-Centric Requirements Engineering. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014, London, UK, 8–11 December 2014; pp. 614–615. [CrossRef]
3. Hosseini, M.; Phalp, K.; Taylor, J.; Ali, R. The Four Pillars of Crowdsourcing: A Reference Model. In Proceedings of the International Conference on Research Challenges in Information Science, Marrakech, Morocco, 28–30 May 2014; pp. 1–12. [CrossRef]
4. Kolpondinos, M.Z.; Glinz, M. GARUSO: A Gamification Approach for Involving Stakeholders Outside Organizational Reach in Requirements Engineering. *Requir. Eng.* **2020**, *25*, 185–212. [CrossRef]
5. Snijders, R.; Dalpiaz, F.; Brinkkemper, S.; Hosseini, M.; Ali, R.; Özüm, A. REfine: A Gamified Platform for Participatory Requirements Engineering. In Proceedings of the 1st International Workshop on Crowd-Based Requirements Engineering, CrowdRE 2015, Ottawa, ON, Canada, 25 August 2015; pp. 1–6. [CrossRef]
6. Groen, E.C.; Doerr, J.; Adam, S. Towards Crowd-Based Requirements Engineering a Research Preview. In *Requirements Engineering: Foundation for Software Quality, Proceedings of the 21st International Working Conference, REFSQ 2015, Essen, Germany, 23–26 March 2015*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9013, pp. 247–253.
7. Sharma, R.; Sureka, A. CRUISE: A Platform for Crowdsourcing Requirements Elicitation and Evolution. In Proceedings of the 2017 10th International Conference on Contemporary Computing (IC3), Noida, India, 10–12 August 2017; pp. 1–7. [CrossRef]

8. Wang, H.; Wang, Y.; Wang, J. A Participant Recruitment Framework for Crowdsourcing Based Software Requirement Acquisition. In Proceedings of the 2014 IEEE 9th International Conference on Global Software Engineering (ICGSE), Shanghai, China, 18–21 August 2014; pp. 65–73. [CrossRef]
9. Hu, W.-C.; Jiau, H.C. UCFrame. *ACM SIGSOFT Softw. Eng. Notes* **2016**, *41*, 1–13. [CrossRef]
10. Moayedikia, A.; Yeoh, W.; Ong, K.L.; Boo, Y.L. Framework and Literature Analysis for Crowdsourcing's Answer Aggregation. *J. Comput. Inf. Syst.* **2020**, *60*, 49–60. [CrossRef]
11. Van Vliet, M.; Groen, E.C.; Dalpiaz, F.; Brinkkemper, S. Identifying and Classifying User Requirements in Online Feedback via Crowdsourcing. In *Requirements Engineering: Foundation for Software Quality, Proceedings of the 26th International Working Conference, REFSQ 2020, Pisa, Italy, 24–27 March 2020*; Springer: Cham, Switzerland, 2020; Volume 12045, pp. 143–159. [CrossRef]
12. Alamer, G.; Alyahya, S. A Proposed Approach to Crowd Selection in Crowdsourced Requirements Engineering for Mobile Apps. In Proceedings of the ICISE 2022: 2022 7th International Conference on Information Systems Engineering, Charleston, CA, USA, 4–6 November 2022; pp. 1–5.
13. Lim, S.L.; Quercia, D.; Finkelstein, A. StakeNet: Using Social Networks to Analyse the Stakeholders of Large-Scale Software Projects. In Proceedings of the 2010 International Conference on Computational Intelligence and Software Engineering, Wuhan, China, 10–12 December 2010; Volume 1, pp. 295–304. [CrossRef]
14. Lim, S.L.; Quercia, D.; Finkelstein, A. StakeSource: Harnessing the Power of Crowdsourcing and Social Networks in Stakeholder Analysis. In Proceedings of the 2010 International Conference on Computational Intelligence and Software Engineering, Wuhan, China, 10–12 December 2010; Volume 2, pp. 239–242. [CrossRef]
15. Lim, S.L.; Damian, D.; Finkelstein, A. StakeSource2.0: Using Social Networks of Stakeholders to Identify and Prioritise Requirements. In Proceedings of the 2011 International Conference on Computational Intelligence and Software Engineering, Honolulu, HI, USA, 21–28 May 2011; pp. 1022–1024. [CrossRef]
16. Lim, S.L.; Finkelstein, A. StakeRare: Using Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation. *IEEE Trans. Softw. Eng.* **2012**, *38*, 707–735. [CrossRef]
17. Mughal, S.; Abbas, A.; Ahmad, N.; Khan, S.U. A Social Network Based Process to Minimize In-Group Biasedness during Requirement Engineering. *IEEE Access* **2018**, *6*, 66870–66885. [CrossRef]
18. Lim, S.L.; Bentley, P.J.; Ishikawa, F. Reaching the Unreachable: A Method for Early Stage Software Startups to Reach Inaccessible Stakeholders within Large Corporation. In Proceedings of the 28th IEEE International Requirements Engineering, Zurich, Switzerland, 31 August–4 September 2020; pp. 376–381. [CrossRef]
19. Srivastava, P.K.; Sharma, R. Crowdsourcing to Elicit Requirements for MyERP Application. In Proceedings of the 1st International Workshop on Crowd-Based Requirements Engineering, CrowdRE 2015, Ottawa, ON, Canada, 25 August 2015; pp. 31–35. [CrossRef]
20. Lim, S.L.; Bentley, P.J. Using PseudoGravity to Attract People An Automated Approach to Engaging a Target Audience Using Twitter. In Proceedings of the 2017 Future Technologies Conference, Vancouver, BC, Canada, 29–30 November 2017.
21. Condori-Fernandez, N.; Lago, P.; Luaces, M.; Catala, A. A Nichesourcing Framework Applied to Software Sustainability Requirements. In Proceedings of the International Conference on Research Challenges in Information Science, Brussels, Belgium, 29–31 May 2019; pp. 1–6. [CrossRef]
22. Dijkshoorn, C.; De Boer, V. Accurator: Nichesourcing for Cultural Heritage. *Hum. Comput.* **2014**, *1*, 101–131. [CrossRef]
23. Alvertis, I.; Papaspyros, D.; Koussouris, S.; Mouzakitis, S.; Askounis, D. Using Crowdsourced and Anonymized Personas in the Requirements Elicitation and Software Development Phases of Software Engineering. In Proceedings of the 2016 11th International Conference on Availability, Reliability and Security (ARES), Salzburg, Austria, 31 August–2 September 2016; pp. 851–856. [CrossRef]
24. Guzman, E.; Alkadhi, R.; Seyff, N. A Needle in a Haystack: What Do Twitter Users Say about Software? In Proceedings of the 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW), Beijing, China, 12–16 September 2016; pp. 96–105. [CrossRef]
25. Zhang, P.; Obradovic, Z. Integration of Multiple Annotators by Aggregating Experts and Filtering Novices. In Proceedings of the 2012 IEEE International Conference on Bioinformatics and Biomedicine, Philadelphia, PA, USA, 4–7 October 2012; pp. 131–136. [CrossRef]
26. Mardjo, A.; Choksuchat, C. HyVADRF: Hybrid VADER-Random Forest and GWO for Bitcoin Tweet Sentiment Analysis. *IEEE Access* **2022**, *10*, 101889–101897. [CrossRef]
27. Schm, R.; Wilcox, S. Harnessing Artificial Intelligence for Health Message Generation: The Folic Acid Message Engine. *J. Med. Internet Res.* **2022**, *24*, 1. [CrossRef]
28. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, 2–7 June 2019; Volume 1, pp. 4171–4186.
29. Peters, M.; Ruder, S.; Smith, N.A. To Tune or Not to Tune? Adapting Pretrained Representations to Diverse Tasks. In Proceedings of the 4th Workshop on Representation Learning for NLP, Florence, Italy, 2 August 2019.
30. Google Code Archive-Word2vec. Available online: https://code.google.com/archive/p/word2vec/ (accessed on 12 January 2023).
31. Nguyen, D.Q.; Vu, T.; Nguyen, A.T. BERTweet: A Pre-Trained Language Model for English Tweets. In Proceedings of the 2020 EMNLP (Systems Demonstrations), Online, 16–20 November 2020; pp. 9–14.

32. Duki, D.; Ke˘, D.; Stipi, D. Are You Human? Detecting Bots on Twitter Using BERT. In Proceedings of the 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), Sydney, Australia, 6–9 October 2020.
33. Rustam, F.; Khalid, M.; Aslam, W.; Rupapara, V.; Mehmood, A.; Choi, G.S. A Performance Comparison of Supervised Machine Learning Models for Covid-19 Tweets Sentiment Analysis. *PLoS ONE* **2021**, *16*, e0245909. [CrossRef] [PubMed]