Acta Scientiarum
http://periodicos.uem.br/ojs
ISSN on-line: 1807-8664
Doi: 10.4025/actascitechnol.v46i1.62867

App store mining for feature extraction: analyzing user reviews

Zulfiqar Ali Memon*, Nida Munawar and Maha Kamal

Department of Computer Science, National University of Computer and Emerging Sciences, ST-4, Sector 17-D, Shah Latif Town (On National Highway) Karachi, Pakistan. *Author for correspondence: E-mail: memon.zulfiqar@gmail.com

ABSTRACT. A recent study shows that the most commonly used app stores, iOS app store and Android's play store had up to 2 million apps from where users can explore, purchase, share, download and install applications on a single click. More and more apps are being added daily which makes the app stores a large software repository. An enormous amount of data is provided by the end users in the form of user reviews. This data can provide valuable insights for acquiring requirements. User reviews include plenty of information as they contain information about faulty features (Bug reports), ideas for new features and improvements (feature requests), or user experience that can help app developers and vendors to achieve software enhancement and evolution tasks. As feature requests are the ones that are most helpful for the purpose of eliciting new requirements, the work is done on feature requests out of the 3 categories mentioned above. This study is conducted to provide a general approach that extracts feature request from user reviews. The proposed approach has five main building blocks, namely, (i) Extraction of Feature Requests, (ii) Feature Extraction from Feature Requests, (iii) topic modelling, (iv) sentiment analysis and (v) Classification into Functional Requirements (FR) and Non-Functional Requirements (NFR). Firstly, it finds the feature requests out of the reviews, then perform extraction of features from feature requests, then further work on grouping the features into topics, next apply sentiment analysis to mine the user opinions on the extracted topic and finally group them into Functional and Non-Functional requirements. This article provides the app developers a more user-centred definition of requirements and improvements. At the topic modelling phase, the results received the highest coherence score, 0.70, with k=22 topics. Sentiment analysis is used to classify feature request, with an accuracy of 80.20%, precision of 84.25%, and recall of 80.42%. With an accuracy of 84.4%, requirements are classified quite successfully as well.

Keywords: feature enhancement; feature requests; requirement engineering; user reviews; non-negative matrix factorization.

Received on March 12, 2022.
Accepted on May 15 2023.

Introduction

Apple AppStore, Blackberry app store, Android Play store, and Windows Phone Apps that are most commonly used had up to 3 million apps (Maalej & Nabil, 2015). It is a large software repository from where users can explore, purchase, share, download and install apps on a single click. Customer information is available in the form of reviews by giving a text feedback and a numeric value star rating after downloading a specific app. Ratings are simply numeric star rating ranges from 1 to 5 that are not much informative as they only involve user content and discontent. More than hundreds of reviews in the textual form are submitted daily for the top apps. However, the top most apps contain very large volume of reviews and hence manual inspection becomes infeasible for developers and analysts to analyse, process, and extracts useful information from the user reviews and the quality of review varies largely. A great challenge in mining the App store is to filter, organize, analyse and effectively extract relevant information from the user feedback to meet the needs of users. Mobile Phone app reviews by users include plenty of information as they contain information about faulty features (Bug reports), ideas for new features and improvements (feature requests), or user experience that can help app developers and vendors to achieve software maintenance and evolution tasks. As feature requests are the ones that are most helpful for the purpose of eliciting new requirements, this article is about feature requests.

Requirement engineering (RE) is the process of eliciting, analysing, documenting, validating and managing the requirements (Laghari, Memon, Ullah, & Hussain, 2018). Requirement gathering is considered to be the most important aspect of the software development life cycle in which user's necessities and

expectations are determined for the particular software product. In fact, the software development life cycle begins by collecting the requirements for the particular application or software. At the end of this phase, we have requirements that envision the functionality and quality of a software. The success of a software system depends on how well it fits the needs of its users and its environment (Laghari et al., 2018). In order to make a good software the requirements must be gathered accurately.

There are many traditional methods for the purpose of gathering new requirements. These methods gather requirements through some specified techniques like interviews, questionnaires, document analysis, prototyping etc. However, they deal with a limited number of users and customers for the purpose of gathering requirements. Whereas, now we have thousands and millions of users downloading softwares and mobile applications from app stores online. The users download the apps and submit their valuable feedback online. In contrast to these old methodologies for requirement gathering which dealt with a few numbers of users to collect the requirements; we need a procedure which makes use of a large amount of user feedback available online on the app stores. The requirements should be extracted from these online reviews which are from a large audience.

With the emergence of app stores as a software market place and a deployment infrastructure, users can now easily submit their feedback, review new releases, report bugs, rate the app and its features, or request new features (Maalej, Nayebi, Johann, & Ruhe, 2016a). The feedback involves how the users' experience with app was, what difficulties they faced when they used the app and which features, they liked. They also request for new features or updates and additions to be made in existing features. This data has lot of in-depth information. This data can provide valuable insights for acquiring requirements.

The user reviews are an important element of mobile app markets. Google Play, Apple App Store are well known distribution platforms where users can download mobile apps, rate them and write review comments about the apps they are using (Maalej et al., 2015). These markets enable a user to provide their valuable opinion for downloaded apps in the form of star rating or in textual reviews. The star rating is very basic and easy to understand. In fact, the total count for each star is all given in the mobile app markets. Hence, it is very easy for a user or someone from the software team to know the amount of likes and dislikes this app has. The ratings in general can give an idea whether the app is being appreciated or not in the market place. However, these ratings alone are not of much use as they do not provide an in-depth detail of why the app is admired or what issues it has. In comparison to this, the textual reviews can offer a broad picture and can be helpful for gathering requirements. Moreover, star ratings are assigned to the overall app and do not give the breakdown on feature level that which particular features is admired or not.

Recent studies have shown that reviews written by the users represent a rich source of information for the app vendors and the developers (Zulfiqar & Syed, 2023; Maalej, Kurtanović, Nabil, & Stanik, 2016b; Ambreen, Ikram, Usman, & Niazi, 2016, and Bosse et al., 2012b). As they include information about bugs, ideas for new features, or documentation of released features. If we specifically divide these reviews submitted by users in the mobile app markets, they can be divided into 3 categories namely: Feature Requests, Bugs when they used the app and User Experience. In Feature Requests users ask for a particular feature, some new attributes and convey ideas related to enhancement of the app. They also request for some necessary features that are present in other apps and are missing in that app or for requirements that would satisfy them and make the app better and easier to use. Example: "I wish the camera had more filters." These feature requests can be very helpful for requirement engineers and business analysts for the sake of requirement gathering. They provide detailed reviews which enable the requirement engineers in having a better understanding of the functionalities needed in order to improve the app. In bugs, users report all kinds of issues the faced with different modules of the app whether hardware or software. Example: "This app crashes if I try to write a lengthy note." Likewise, the bugs can be very helpful for QAs' and developers to know the issues in the app. User experiences describe user stories and the experience they had with particular app. They provide information about how the app was used and for what purposes it has been used. It may even give some new usage ideas for which the app was not initially invented and can provide an additional functionality related to it. Thus, they are helpful in making documents and user manuals. Hence, user feedback in the form of comments reviews the existing app. It is beneficial in two ways which are getting new ideas and requirements for the upcoming releases and also in refining the existing functionality and quality of the app.

This article proposes an approach which makes use of the information available online through user feedback. In order to achieve this, we develop a technique which combines natural language processing, sentiment analysis and machine learning to accomplish the goal of app store-oriented user-based requirements. We extract the online reviews from app store and then out of them finds the ones that are feature requests. After finding the reviews which contain feature requests; we find different features from the text of the review. Many similar features can be grouped under topic. Many sub topics can come under the branch of a topic. This article applies Topic Modelling for grouping similar features extracted into a Topic. This helps the requirement engineers in better understanding the requirements and it also increase the readability. After that we distribute the user feedback into positive comments and negative comments using sentiment analysis. This helps even more in understanding the user feedback related to a particular topic. The comments which are positive express about good experiences that the users had. Whereas, the negative ones give an idea about the problems that the users faced and the issues that need to be addressed related to with user feedback. Our goal is to mine the comments from masses in order to improve the features of an app a particular feature. Finally, we group these requirements on the basis of their topics under two well-known requirement engineering categories i.e., Functional and Non-Functional Requirements.

Material and methods

Requirement gathering is an essential phase of the software development life cycle. There are many conventional methodologies present for Requirements Engineering but we need a methodology to fulfil the dire need of acquiring from the virtual world of app distribution markets. We should utilize the huge amount of data present in the form of user feedback existing online at different user forums, app stores and social media for the purpose of requirement elicitation. Therefore, in order to extract meaningful and valuable information for software evolution and requirements optimization a methodology is proposed. It will use the vast amount of user feedback available online, collect requirements from a large audience and facilitate business analysts and requirement engineers to gather requirements through the information available online. In our data set, there are 2500 phrases from 30 distinct apps across six different categories. The data is downloaded in the form of csv files. It contained the attributes Date, Time, App ID, App Name, Language, Author, Stars, Title and Review. All columns other than Review were removed. Then through random sampling few reviews were selected from the large amount of data present and transformed into text files and collect insights for new requirements App Follow was used to compile the data set. App Follow is a website that keeps tabs on sales, keywords, and reviews in the App Store, Google Play, and Windows Store. Then data pre-processing steps undertaken, which involved randomly selecting reviews for each app and segmenting sentences to extract feature requests for classification. Then the reviews that contain feature requests are extracted from the user reviews. the extraction of feature requests from user reviews is done by using a Java-based classifier called ARDOC, followed by data cleansing using natural language processing techniques such as removal of stop words, conversion to lower case, removal of punctuation, and lemmatization. The processed sentences were then used for feature extraction. After data cleansing, three methods were used for feature extraction from feature requests, including bag-of-words model, rapid automatic keyword extraction algorithm (RAKE), and term frequency–inverse document frequency (TFIDF), with TFIDF being the most effective method. TFIDF extracts text features by scoring the importance of words based on how frequently they appear across multiple documents, resulting in individual words as features rather than phrases. When applying topic modelling on the features extracted by TFIDF, better results are obtained compared to selecting phrases yielded by RAKE and bag-of-words model. After getting the features we apply topic modelling. Topic modelling is a statistical technique used to group similar ideas into topics and is composed of three main steps: data representation, latent topic decomposition, and topic extraction. Latent Dirichlet Allocation (LDA) and Non-Negative Matrix Factorization (NMF) are two main models used in the second step. NMF is preferred due to its higher topic coherence score and faster training time. The output of topic modelling contains the topic number with topics and a matrix of features is printed under it. Now All the similar features will be grouped together into a topic. The next step is to apply sentiment analysis to understand which topics and features are liked or disliked by users. After getting the optimal topics, we apply sentiment analysis on the feature requests. Sentiment analysis is used to analyze user reviews and determine the positive and negative feedback on app features, allowing for maintenance and enhancement of the app. The approach includes classification of sentences into polar

and subjective, and calculating subjectivity score using Naive Bayes classifier and Text blob python library. The accuracy of sentiment analysis is 80.20%, with the majority of feature requests being positive. Lastly, they are classified into Functional Requirements (FR) and Non-Functional Requirements (NFR). Topics with their features and feature requests are categorized into functional and non-functional requirements using a supervised machine learning approach, specifically the Naïve Bayes classifier, with an accuracy of 84.4%. Results are evaluated using accuracy, precision and recall.

Table 1 gives a summary of all the tools and platforms used in this experiment. Coding for all of the phases has been done in python3.6 except for "Extraction of feature requests" phase which uses a java base classifier. The natural language processing libraries used in different stages of the experiment has also been stated. The machine learning classifiers used for supervised classification in different stages of this study have also been mentioned.

Table 1. tools and platforms.

Phase	Tools
Data Extraction	App Follow
Data Preprocessing	Natural Language Toolkit (NLTK)
Feature Request Extraction	ARDOC classifier
Data Cleansing	Natural Language Toolkit (NLTK)
Feature Extraction	Code written in Python (Sickit learn used)
Topic Modeling	Code written in Python (Sickit learn used)
Sentiment Analysis	Code written in Python (Naïve bayes analyzer used)
Python Tools	IDLE Python 3, Google Collab and Jupyter notebook

Our approach is depicted through Figure 1. It involves extracting user reviews from app stores.

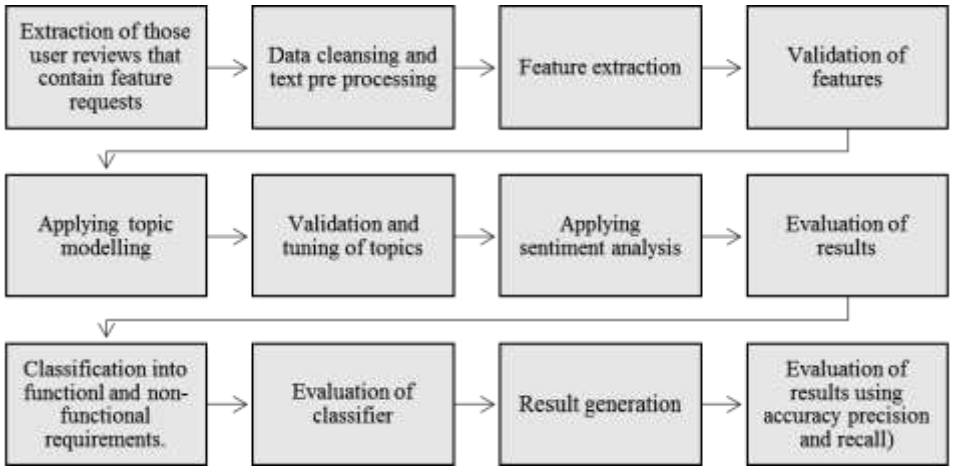

Figure 1. Proposed approach.

The study is conducted on app stores data and its implementation details below.

Implementation

This section provides details about the implementation of the proposed model.

Data set

There are various app stores like Google Play, Apple app store, Windows app store, Blackberry app store, Samsung app store and many others. We needed a good data set with large amount of user reviews in it. Hence Google Play and Apple app store where chosen due to the fact that they are most commonly used and have high number of downloads. As well as, they have a lot of reviews for each kind of apps. There are range of reviews from 3 letter words to a paragraph of 5 sentences discussing multiple aspects. This makes the data set very proficient as it covers a wide range of entities. For our study the user reviews needed to be collected from multiple sources i.e. they had to be from different apps belonging to different categories, to get good quality results. We collected data sets from 2 apps stores Google Play and Apple's app stores. These data sets contain data of 30 apps belonging to 6 different categories. The data was collected from Dec 2018 to Dec 2019 (Laeeq,

Memon, & Jamshed, 2018). The data set was collected using App Follow. App Follow is a website which monitors reviews, keywords, and sales in App Store, Google Play and Windows Store. We used this app to download data from 30 different apps. Table 2 gives the list of app categories belonging to 6 different categories and corresponding.

Table 2. List of apps according to category.

Category	Apps and Store
Photography	PicsArt (Google Play store)
	B16 (Google Play store)
	Candy Camera (Google Play store)
	CutCut (Google Play store)
	Sweet Selfie (Google Play store)
Weather	Accuweather (Google Play store)
	Weather (Google Play store)
	Weather Forecast (Google Play store)
	Yahoo weather (Google Play store)
	Weather Live (Google Play store)
Utilities	UC Browser (Apple App store)
	Truecaller (Apple App store)
	Google Chrome (Apple App store)
	Google (Apple App store)
	Firefox (Apple App store)
Photo & Video	TikTok (Apple App store)
	Instagram (Apple App store)
	Youtube (Apple App store)
	Imo (Apple App store)
	B612 (Apple App store)
Social Networking	FB Lite (Apple App store)
	Facebook (Apple App store)
	Messenger (Apple App store)
	WhatsApp (Apple App store)
	Snapchat (Apple App store)
Travel	Careem (Apple App store)
	Uber (Apple App store)
	Google Earth (Apple App store)
	The Emirates App (Apple App store)

Data pre-processing

We selected reviews through random sampling for each app. The reviews were selected from 3 types of comments short (5 to 7 words), medium (7 to 12 words) and long (12 words and beyond). From the large data set a subset of sentences are drawn. The subset was stored in the form of text files in ANSI encoding. We needed to extract those sentences that contained feature requests from the user reviews. As out of the 3 categories the proposed model will only be working on feature request. To extract those feature request we first segmented the sentences as the classifier we used gave improved results on segmented sentences (Zulfiqar & Syed, 2023). Instead of passing the whole text file as it is, it is better to segment the sentences and store them in a text file. We gave the text file with user reviews as input and generated another text file whose sentences for each review were segmented as output as shown in Table 3.. Then passed that file for classification.

Table 3. Results of segmented sentences.

Exp01	['There is no blacklist/block option to hashtags.']
Exp02	['Too many ads!', 'Interrupt videos a lot!']
Exp03	['I updated WhatsApp yesterday, and then the app is not running for me, where is the problem?']
Exp04	['Please bring the old filters back']

Extraction of feature requests

We needed to extract those sentences that contained feature requests from the user reviews. As out of the 3 categories namely bugs, feature request and user experiences; we will only be working on feature request. We used ARDOC (Zulfiqar & Syed, 2023), a java-based classifier to extract feature requests from the text files. Hence, to extract the feature request we passed the text file containing segmented sentences into ARDOC.

ARDOC classified all the sentences into their respective categories, as shown in Table 4. As we only required feature requests for our research work therefore the sentences classified as feature requests were exported to a text file. Next data cleansing was performed on the feature request.

Table 4. Results of ARDOC classifier.

Exp01	['You guys really need to add the background playback feature.']-feature request
Exp02	['The application should respond faster']- NON-FEATURE REQUEST
Exp03	['I would like there to be a time limit or something to notify customers of how much time they have to get to the driver.']-FEATURE REQUEST
Exp04	['You guys should bring back the feature where you can see who other people's best friends are']-feature request

Data cleansing

Data cleansing is an important process. The data extracted was in csv files. It contained a lot of information but we only needed reviews. Hence, the user reviews were extracted and stored in text files. Reviews less than 4 words in a sentence were deleted. Sentence segmentation was performed on them first. Later those sentences were used for extracting those reviews that contain 'feature requests' out of all of the user feedback. When the feature requests were extracted, they went through a process of data cleansing.

In order to filter the noise and garbage from data we used a number of natural language processing techniques. NLTK is a very powerful python library which simplify the data cleansing tasks. We used this library in order to retrieve resourceful information. The text file with feature requests was given as input. It is read and stored in a list data frame. Each review is distinguished by means of a ','. At the end of every review there is a comma after which new review starts. All the reviews are loaded and cleansing tasks are performed. Removal of stop words, converting text to lower case, removal of punctuation and lemmatization are the text mining techniques used. Stop words are words frequently used but provide no value for features. (Examples of stop words are a, and. I etc.). These words are removed and after that all the sentences are converted to lower case for uniformity. Then, punctuation marks are removed from the text. Only sentences containing words are left, stop words and punctuation is removed. Lemmatization is the process of reducing words to their base form. Its help in the sense that the same words are replaced by their base words which increases their frequency in the text corpus and we get to know which words are occurring frequently. Lemmatization is applied to the sentences and then they are used for extraction of features, as shown in Table 5.

Table 5. Output of the sentences after applying lemmatization, lowercase conversion, stop words, punctuation, and number removal.

Input sentence	Output sentence
['you guys really need to add the background playback feature.']-feature request	"guys really need add background playback feature."
['Also please add a feature where we can turn off the discover stories.']-feature request	"please add feature turn discover stories."
I would like there to be a time limit or something to notify customers of how much time they have to get to the driver.']-feature request	"like time limit something notify customers much time get driver."
['You guys should bring back the feature where you can see who other people's best friends are']-feature request	"guys bring back feature see people best friends."

Feature extraction from feature requests

After the data cleansing has been performed feature extraction steps are executed. For features' extraction we use the sentences obtained after data cleansing. There are 3 methods used to extract features bag-of-words model (BoW model), rapid automatic keyword extraction algorithm (RAKE) and term frequency–inverse document frequency (TFIDF). In the first method which is bag-of-words we generate the word count for each of the word present in text. The words with high frequency are selected. However, it is just selecting features on the basis of word count which is not such a good idea. Therefore, RAKE was tried, which generates a set of possible words that can be keywords, then calculate their frequency and words below the threshold are rejected. These extracted features are then sorted based on their frequency and given as output. The output gives the features extracted and arranges them according to their frequency in descending order. The extracted features are a set of features of their apps, it generates word phrases as keywords, which do not

properly justify the apps features. They are not as good especially when we apply topic modelling in later phases, they do not provide us with good results. As a consequence, we try, yet another feature extraction method which is TF-IDF.

We applied TF-IDF to extract text features from text documents which is the most popular and widely used method for extraction of keywords. TF-IDF stands for "Term Frequency, Inverse Document Frequency". It is a way to score the importance of words (or "terms") in a document based on how frequently they appear across multiple documents (Kashif & Memon, 2019). It calculates frequency for given word in the document, the value increases proportionally to the number of times a word appears in the document, but is often offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general (Loria, 2013). Therefore TF-IDF returned best features as the features returned were not phrases but single words. Moreover, they are chosen through vectorization not merely on word count.

Data was passed into TF-IDF function and features were returned.TF-IDF yields separate words as features. When we apply topic modelling on separate features extracted by TF-IDF we get better results whereas if we select phrases yield by RAKE and apply Topic Modelling, we don’t get such good results.

Topic identification through topic modelling

To explore similarities among topics (Z) and Corpus (D), and also for the relations among the topics and terms (W) Topic modelling is used (Luiz et al., 2018). This process consists of 3 main steps:

I. data representation
II. latent topic decomposition
III. topic extraction.

In the first step called "data representation", firstly wo do data pre-processing step, secondly the textual corpus that contains string/words are transformed into a vector space model that is of fixed length into its global term weights that give us word(term) occurrence information (Luiz et al., 2018). This step is performed earlier using two techniques Count Vectorizer and TF-IDF.

It was decided to use TF-IDF as the final method for feature extraction as it produced better results. Afterwards Topic Modelling was applied on the extracted features. The text file contains feature of different apps belonging to different categories. It is found out the features for those apps for reviews but there are many features. But only giving the features of a document is not enough. To ease the reader, increase the readability and for make sense of those features it is needed to group them together to into a Topic. A single document may contain many ideas and may discuss many topics. Topic Modelling groups similar ideas under a topic. Topic model is a statistical model for determining the abstract "topics" that occur in a collection of documents.

In the second step called "latent topic decomposition" classified in two main models:

I. probabilistic models
II. non-probabilistic models.

Therefore, LDA (Latent Dirichlet Allocation) was chosen from probabilistic models and NMF non-negative matrix factorization from non-probabilistic models. LDA is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. Bosse et al. (2011b) For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's creation is attributable to one of the document's topics. LDA is an example of a topic model. but LDA often leads to formation of incoherent topics and data sparsity.

The second technique called "non-probabilistic topic modelling" includes approaches such as matrix factorization. In Factorization we manage to break down a bigger number into the product of two smaller numbers, and in matrix factorization method we break down a larger matrix into the product of two smaller matrices. These two smaller matrices are relatively smaller than the original one in order to reduce the doc space. A well-known matrix factorization procedure non-negative matrix factorization (NMF) in which a bigger matrix called A is break down into two smaller size matrices

A: $A \in R^{n \times m}$

H: $H \in R^{n \times k}$

W: $W \in R^{k \times m}$

$A \approx H \times W$

n= total no of the documents in the set
m= total no of terms across the given set of corpora
k=no of topics (latent terms).

Where A spans a large dimensional space (bigger matrix) that is decomposed into two matrix H (defines the link among the documents and the topics) and W (defines the link between the topics and the terms) such that A≈H×W (A is approximated to W and H), with the property that all three matrices have non-negative elements because having negative values in the matrix does not make any sense. As we know that topic modelling is the unsupervised learning technique. The topics in the document are also called latent because they are unknown to us. To find the number of optimal topics (k) from the total no of documents is a great challenge in topic modelling. In order to estimate the optimal no of topics we use Topic Coherence technique. The resultant matric that is produced through topic coherence encodes the semantic relation between the words (terms) that defines a topic "how semantically close are the words that describe a topic.". To evaluate the performance of topic modelling that is an unsupervised technique we use Topic coherence c_v measure. The higher the Coherence score, the better the model. The Topic coherence of NMF for topic modelling is 0.70 for the optimal no of topics K= 22 and the Topic coherence of LDA for topic modelling is 0.62 for the optimal no of topics K=5 and the time to train a NMF model is much less than the time to train a LDA model. Therefore, in this case NMF produces the best result for topic modelling and it is the algorithm chosen for topic modelling (Ubaidullah, Zulfiqar, Shafaq, Balouch, & Batra, 2018).

Right now, the output contains the topic number with topics and a matrix of features is printed under it. Also, the list of features requests is printed under them. For a reader it is easy to understand that this is the topic with these features and the complete review is also present under. Further we relate each review to its corresponding topic number. The percentage of user reviews for each topic is shown in Figure 2. The plot depicts that the topic 0 has high no of reviews. Next, sentiment analysis will be applied on the review, to give the reader (whoever it is a developer or an analyst) an understanding about which topic and feature is gaining positive comments and which set of features are disliked by their users (Kashif & Memon, 2018).

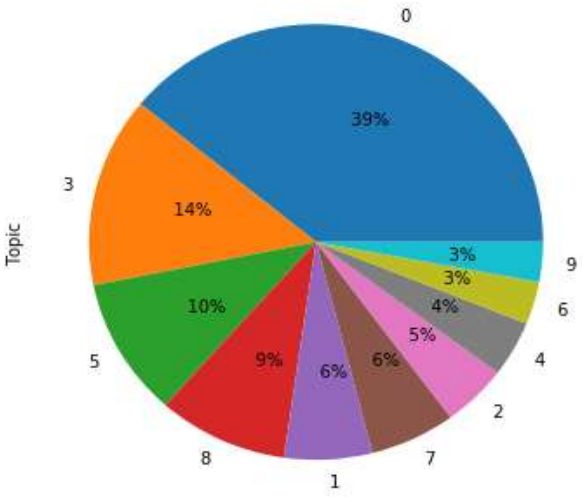

Figure 2. Percentage of user reviews for each topic.

Sentiment analysis on feature requests

Until now we know the topic and features of an app about which users are discussing in their reviews. To develop a better understanding of users' perspective we perform sentiment analysis on the reviews. It depicts customer views for app features. We analyse user review statements to find user opinions. Example of a positive feature request is "i needed something that would allow me to highlight and make notes" and that of a negative is "some text is missing and there is a glitch which if i slide to go next page the option at top will pop out". It tells which features are liked by people are a gaining a positive responses and which features are disliked by people are gaining negative responses. If the reviews for a certain topic have more positive response, which means people are appreciating, therefore it can be maintained and if it has negative response, which means it needs to be enhanced. We also get new requirements from them.

There were three options available either we could simply classify the sentence as positive or negative or we could classify into positive, negative, neutral. The third approach was to classify the statement into polar and subjective and then calculate the subjectivity (i.e., it is positive or negative) along with its score. To perform sentiment analysis on the reviews we use Naive Bayes classifier which is very efficient algorithm when it comes to handling text. Text blob is another major python library for text mining which is very helpful in performing text related tasks. We used this library for mining user views. Text blob has a class Naïve Bayes Analyzer which is trained to find sentiments of a text and it uses Naive Bayes classifier. It is utilized to predict the outcome of our reviews into positive and negative. Firstly, it is checked that the review is neutral or has some biasness in is (i.e. it has negative or positive sentiments). Then we get the classification of a sentence into positive or negative along with the positivity and negativity score. If a sentence is classified is positive, it will tell the percentage of positivity and also the percentage of negativity. The label for which it has been classified will have high percentage. The output is in 2 steps, first we get the polarity and subjectivity; then the subjectivity i.e., negative or positive sentiment of a sentence, is checked. Then we get the output as label positive or negative and the score for both labels. The accuracy for sentiment analysis is 80.20%. Figure 3 shows total no of positive and negative reviews we can see that the bulk of the feature request are positive.

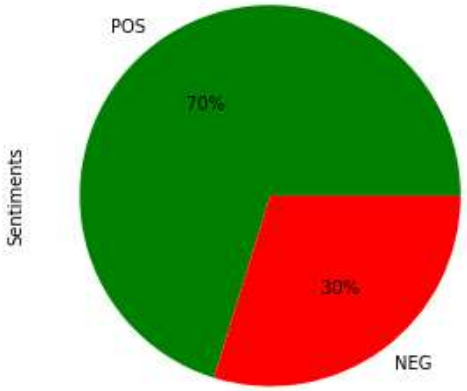

Figure 3. Total no of positive and negative reviews.

Classification into functional and non-functional requirements

Now we have the topics with their features and features requests as shown in Figure 4. For each request we have corresponding sentiment label and scores. We finally categorize them into two popular categories of requirement engineering i.e., Functional Requirement (FR) and Non-Functional Requirements (NFR) as shown in Table 6. As, it is a classification task, therefore the approach chosen was supervised machine learning. There are many algorithms that can be used but as it is a text classification problem, Naïve Bayes classifier was used which is the finest algorithm for text classification. We divide the data into 2 categories functional and non-functional requirements and assign labels to the data set. After that we train the model on 80% percent of the data and test it with 20% of it. We achieve an accuracy of 84.4%.

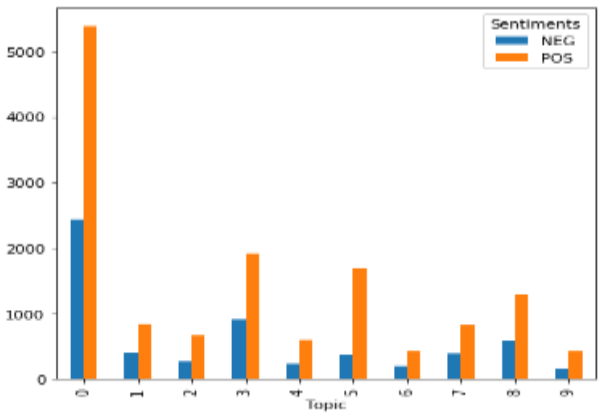

Figure 4. Observes the division of sentiment for each individual topic.

Table 6. Categorization of feature request into functional and non-functional requirements.

Requirement category	Input sentence
US (usability)	"guys really need add background playback feature."
F (functional requirement)	"please add feature turn discover stories."
F (functional requirement)	"like time limit something notify customers much time get driver."
UE(user experience)	"guys bring back feature see people best friends."

Results and discussion

To conduct the experiment, we used classification algorithms like Naïve Bayes and ARDOC, natural language processing libraries NLTK and TextBlob and AppFollow. All the coding is done in python. In our experiment we first pull-out user reviews, then extract feature reviews from them by the help of ARDOC. We have a data set of 2500 sentences belonging to 30 different apps from 6 different categories. Subsequently, we cleanse the feature reviews and extract features from them.

Next topic modelling is applied. Two approaches were tried; NMF and LDA topic modelling which were implemented by means of a code. As topic modelling is an unsupervised technique, determination of the number of topics is the trickiest and the hardest part of topic modelling (Samad, Asad, Memon, Aziz, & Rahman, 2018). Therefore, the evaluation of topic modelling is done by using two most popular techniques:

1) Perplexity
2) Topic coherence

Perplexity is the weak indicator of the quality of the topics as it cannot capture the semantic closeness between the words even the results are sometimes slightly anti-correlated.

The resultant matric that is produced through topic coherence encodes the semantic relation between the words (terms) that defines a topic "how semantically close are the words that describe a topic". To evaluate the performance of topic modelling that is an unsupervised technique we use Topic coherence c_v measure. Table 7 portrays comparison of topic modelling methods.

The Figure 5, shows the optimal no of topics for LDA. The optimal no of topics relates to higher coherence value. It can be seen that the topic from 2 to 8 gives us a better coherence score 0.62 and also there is a rise for 15 number of topics with the coherence score 0.56.

Table 7. Comparison of topic modelling methods.

Topic Models	Coherence Score	K Optimal no of topics	Time To train a model
NMF	0.70	K=22	1.2568023999999696 seconds for 14 topics
LDA	0.62	K=5	259.35074399999576 seconds for 14 topics

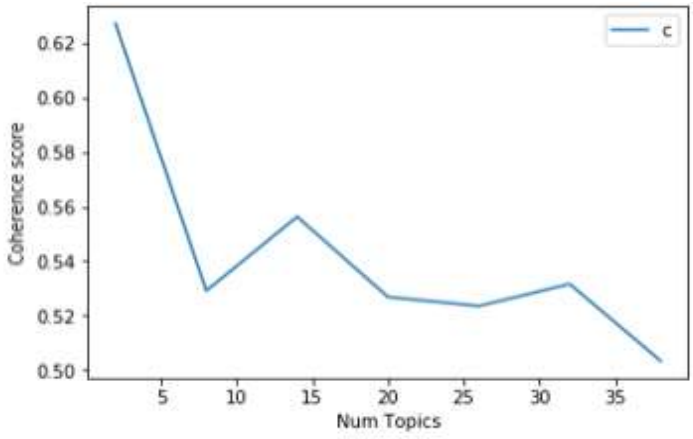

Figure 5. Optimal no of topics w.r.t Coherence Score for LDA.

Figure 6, shows the optimal no of topics for NMF. The optimal no of topics is relating to higher coherence value. It can be seen that the topic from 15 to onwards gives us a better coherence score and from k=22 e obtains higher coherence score 0.70.

Table 8 portrays the results we get when we used Naïve Bayes classifier for sentiment analysis. We obtain an accuracy of 80.20% while the precision and recall are 84.25 and 80.42% respectively.

Next, we used Naïve Bayes classifier for categorizing requirement into functional and non-functional requirements. Table 9 represents a confusion matrix which show how accurately Naïve Bayes has classified the requirements. We labelled the topics as functional and non-functional, the data consisted of 300 topic sets out of which 200 were functional and 90 were non-functional.

Figure 6. Optimal no of topics w.r.t Coherence Score for NMF.

Table 8. Confusion Matrix for Sentiment Analysis.

		Predicted	
	Sentiment Class	Positive	Negative
Actual	Positive	1150	280
	Negative	215	855

Table 9. Confusion Matrix for Functional and Non-Functional Requirements.

		Predicted	
	Categorizing requirements	Functional requirements	Non-functional requirements
Actual	Functional requirements	173	18
	Non-functional requirements	27	72

At first, we randomly divided the dataset into 80% train data and 20% test data. Most of the training data consisted of functional requirements as they were more in number. As a result of which most topics were classified as functional though they belonged to non-functional category and functional were always correct classified. Therefore, the data is divided into 60% train data set and 40% test data set carefully. After that we see that non-functional requirements and functional requirements are classified accurately. The accuracy observed is 84.4%, whereas the precision and recall are 86.5 and 90.5% respectively.

We extracted the app store data (of google play and apple app store) using App Follow in the rom of csv files. Then extracted features requests by means of ARDOC. We collected 2500 feature requests and saved them into 30 text files with mostly files being of 100 sentences and a few of 70, 80 and 85 sentences. We observe that to get the best results the number of sentences in a text file should be 85 to 100 as that way it is not over fitted or generalized. We process these text files to find features, topics, sentiments and requirement categories. We have separately found the accuracy, precision and recall for each stage which are mentioned above. Feature from feature requests have ben efficiently identified. In topic modelling we see some topics are very precisely distributed. Some of the topic words we get are very accurate while some are not so well defined. In sentiment analysis user reviews are classified with an accuracy of 80.20%. The classification of requirements is also done very well achieving an accuracy of 84.4%.

Related work

Since their inception in 2008 till now app stores have many apps which are used by a large audience and then honestly reviewed (Groen & Koch, 2017). They contain much information about the users'

requirements and preferences, which can be very useful to the requirements engineer (Bosse, Hoogendoorn, Memon, Treur, & Umair, 2012a). This enables us to acquire and interpret data from very large and heterogeneous groups of users (Bosse, Memon, & Treur, 2012b). Observing this behaviour in the software market (Laghari et al., 2018) have envisioned a "data-driven requirements engineering". They foresight the emergence of a new trend in which requirements elicitation, negotiation, and analysis as an open process with a stronger degree of interaction between developers, analysts, and users. (Bosse et al., 2012b) has also described this idea by the name of "CrowdRE". The feedbacks from users have much potential valuable information, such as feature requests, bugs or sentiment (LaToza, & Hoek, 2016). Software vendors are gathering this feedback data and use it. Future software and requirements engineering should be able to gather and engineer requirements from user masses when deciding on what to develop (Laghari et al., 2018).

Maalej et al. (2016a) applied frequent item set mining for identifying latent patterns among the topics from reviews. Bosse et al. (2011b) proposed a tool which combines three techniques Natural Language Parsing, Text Analysis and Sentiment Analysis to automatically classify useful feedback contained in app reviews. There are a few authors like Zulfiqar and Syed (2023) and Bosse et al. (2012b) who have earlier classified reviews into categories but Boose et al. (2011a) have done a classic job they have made ARdoc: App Reviews Development Oriented Classifier which efficiently classifies the user feedback into 5 categories information giving, information seeking, feature request, problem discovery and other. (Zulfiqar & Syed, 2023) have also done similar work by classifying the raw user reviews into 4 categories precisely bug reports, feature requests, user experiences, and ratings.

Bosse et al. (2011a) have proposed an SVM based technique for mining user feedback and extracting requirements from Social Q&A sites. Laghari et al. (2018) have applied natural language processing techniques to identify the unique features inside the reviews from apps store, discover user sentiments about the identified features and applied topic modelling techniques to assemble the similar features under a topic. (Bosse, Duell, Memon, Treur, & van der Wal, 2015) have adequately applied topic modelling using LDA and information retrieval techniques to extract word-based topics from reviews.

Abbasi et al. (2021) extracted feature requests from app store reviews by means of linguistic rules and used LDA to group the feature requests. They presented MARA (Mobile App Review Analyser), a prototype for automatic retrieval of mobile app feature requests from online reviews.

Hoogendoorn, Klein, Memon, and Treur (2013) have used topic modelling along with LDA to extract topics from user reviews. Text pre-processing is done using natural language processing techniques. Bosse et al. (2012a) have proposed AR-Miner; a framework to extract useful features requests using topic modelling and then prioritized them using an effective ranking scheme.

Laghari et al. (2018) propose a methodology to automatically extract the features of an app from its corresponding reviews by applying information retrieval methods and grouped the similar using weighted-tree similarity algorithm.

Malik and Shakshuki (2016) have used Natural Language Processing for extraction of features from user reviews. Bosse et al. (2012) propose SURF (Summarizer of User Reviews Feedback), a tool which summarizes user reviews content. It is able to analyse and classify the information contained in app reviews using Natural Language Processing. Laghari et al. (2018) have applied a very interesting and different approach for requirement analysis in which they make use of activity theory for the extraction of requirements for a mobile app they made. Bosse, Memon, and Treur (2011a) have extracted features using information retrieval augmented with ontological analysis and used them to distinguish apps. Abbasi et al. (2023) have applied sentiment analysis on user feedback to gather requirements. Bosse, Duell, Memon, Treur, and van der Wal (2017) have used LDA topic modelling to group similar reviews extracted from app stores.

Kumari and Memon (2022) collected reviews from 7 apps of the Google Play Store they include all the categories bug reports, functionalities, and user experience from the reviews. For topic modelling block they use NMF with SToC and for sentiment analysis they used SACI.

Bosse et al. (2011b) have selected a new approach in which they classified user response into three types Non–Functional requirements, Functional Requirements and Others. Then, further divided Non-Functional requirements into reliability, usability, portability, and performance. Kumari and Memon (2022) have worked on the Non-Functional Requirements as well to classify whether a sentence is a Non-Functional Requirement or not. Yumna, Hina, and Memon. (2018) facilitated the classification of requirements as functional and non-

functional by means of supervised learning. In addition to this they have divided the non-functional requirements into usability, security, operational, and performance requirements.

From the literature review it is evident that research work done focuses on a single or few areas but not on all areas as in our research. In comparison to this, we have devised a comprehensive approach which covers all 4 areas from extraction of features to categorizing into functional and non-functional requirements. Our work will extract reviews, then use a combination of natural language processing techniques (for text pre-processing), topic modelling, feature extraction methods, sentiment analysis and machine learning to enhance the requirement gathering process. Our research combines 4 areas / levels for the sake of requirement engineering and feature enhancement. The areas are Feature extraction, topic modelling of features, sentiment analysis and grouping requirements in categories of functional and non-functional requirements.

Contribution

A novel approach for feature enhancement and requirements evolution from online user reviews. The contribution of this research is that it is a comprehensive approach from extracting user reviews to their classification into functional and non-functional requirements. This procedure has accomplished 4 major tasks feature extraction, assembling features into topics, opining mining and classification into requirement categories as a part of its procedure. This can be productive for feature enhancement and requirement gathering. Another contribution is that by the help of sentiments it exhibits the features that are received well besides the one that are not popular. Feature extraction, assembling features into topics are the phases that have used natural language processing techniques as they are more efficient in filtering noise, extracting features and grouping topics. In short, when dealing with a large text corpus, it is best to use text mining techniques which perform brilliantly instead of manually training new classifiers. Opining mining and classification into requirement categories are the phases in which machine learning techniques were applied. As the sentiment analysis and categorization of reviews into functional and non-functional requirements are tasks which required dividing the data into classes. Supervised machine learning algorithms were the appropriate option for these phases of the study due to the fact that a supervised learning algorithm analyses the training data and classifies unseen data into the class labels very well.

Conclusion

In this paper we develop an exclusive procedure that benefits requirement engineers and business analysts in particular and software distributors in the app distribution markets in general to enhance their app. We have examined feature requests extracted from user reviews for the sake of requirement elicitation. We have done a fine-grained analysis of features present in the feature requests and assembled the similar features under a heading. Following, we benefited from sentiment analysis to know the user opinions on the particular feature. Afterwards, we have classified the topics into two significant high-level categories of functional and non- functional requirements.

References

Abbasi M. A., Yen, L. C., Abdullah, A. K., Zulfiqar, A. M., Nouman, M. D., Jing, Y., … Lip, Y. P. (2023). Enabling iot service classification: a machine learning-based approach for handling classification issues in heterogeneous iot services. *IEEE Access, 11*, 89024-89037. DOI: https://doi.org/10.1109/ACCESS.2023.3306607

Abbasi, M. A. Memon, Z. A., Durrani, N. M., Haider, W., Laeeq, K., & Mallah, G. A. (2021). A Multi-layer trust-based middleware framework for handling interoperability issues in heterogeneous IOTs. *Cluster Comput, 24,* 2133–2160. DOI: https://doi.org/10.1007/s10586-021-03243-1

Ambreen, T., Ikram, N., Usman, M., & Niazi, M. (2016). Empirical research in requirements engineering: trends and opportunities. *Requirements Engineering*, *23*, 63-95.

Bosse, T., Duell, R. Memon, Z. A., Treur, J., van der Wal, C. N. (2017). Computational model-based design of leadership support based on situational leadership theory. *Simulation, 93*(7), 605-617. DOI: https://doi.org/10.1177/0037549717693324

Bosse, T., Duell, R., Memon, Z.A., Treur, J., & van der Wal, C. N., (2015). Agent-based modelling of emotion contagion in groups. *Cognitive Computation Journal, 7*(1), 111-136. DOI: https://doi.org/10.1007/s12559-014-9277-9

Bosse, T., Hoogendoorn, M., Memon, Z. A., Treur, J., & Umair, M. (2012a). A computational model for dynamics of desiring and feeling. *Cognitive Systems Research, 19*, 39-61. DOI: https://doi.org/10.1016/j.cogsys.2012.04.002

Bosse, T., Memon, Z. A., & Treur, J. (2011a). A recursive bdi-agent model for theory of mind and its applications. *Applied Artificial Intelligence*, *25*(1), 1-44. DOI: https://doi.org/10.1080/08839514.2010.529259

Bosse, T., Memon, Z. A., & Treur, J. (2012b). A cognitive and neural model for adaptive emotion reading by mirroring preparation states and hebbian learning. *Cognitive Systems Research, 13*(1), 39-58. DOI: https://doi.org/10.1016/j.cogsys.2010.10.003

Bosse, T., Memon, Z. A., Oorburg, R., Treur, J., Umair, M., & De Vos, M. (2011b). A software environment for an adaptive human-aware software agent supporting attention-demanding tasks. International Journal on Artificial Intelligence Tools, 20(5), 819-846. DOI: https://doi.org/10.1142/S0218213011000310

Groen, E. C., & Koch, M. (2016). How Requirements Engineering can benefit from crowds. *Requirements Engineering Magazine*. Retrieved from https://re-magazine.ireb.org/issues/2016-2-take-the-broader-view/how-requirements-engineering-can-benefit-from-crowds

Hoogendoorn, M., Klein, M. C. A., Memon, Z. A., & Treur, J. (2013). Formal specification and analysis of intelligent agents for model-based medicine usage management. *Computers in Biology and Medicine, 43*(5), 444-457. DOI: https://doi.org/10.1016/j.compbiomed.2013.01.021

Kashif, L., & Memon, Z. A. (2018). An integrated model to enhance virtual learning environments with current social networking perspective. *International Journal of Emerging Technologies in Learning*, *13*(9), 252-268. DOI: https://doi.org/10.3991/ijet.v13i09.8000

Kashif, L., & Memon, Z. A. (2019). Scavenge: An intelligent multi-agent based voice- enabled virtual assistant for LMS. *Interactive Learning Environments NILE*, *29*(6), 954-972. DOI: https://doi.org/10.1080/10494820.2019.1614634

Kumari, S., & Memon, Z. A. (2022). Extracting feature requests from online reviews of travel industry. *Acta Scientiarum. Technology*, *44*(1), e58658. DOI: https://doi.org/10.4025/actascitechnol.v44i1.58658

Laeeq, K., Memon, Z. A., & Jamshed, M. (2018). The SNS-based e-learning model to provide smart solution for e-learning. *International Journal of Educational Research and Innovation*, *10*, 141-152. Retrieved from https://www.upo.es/revistas/index.php/IJERI/article/view/2764

Laghari, A., Memon, Z. A., Ullah, S., & Hussain, I. (2018). Cyber physical system for stroke detection. *IEEE Access, 6,* 37444-37453. DOI: https://doi.org/10.1109/ACCESS.2018.2851540

LaToza, T. D., Hoek, A. D. H. (2016). *Should software developers be replaced by the crowd?*. IEEE Software Blog. Retrieved from http://blog.ieeesoftware.org/2016/03/should-software-developers-be-replaced.html

Loria, S. (2013). *Tutorial: Finding Important Words in Text Using TF-IDF*. Retrieved from http://stevenloria.com/finding-important-words-in-a-document-using-tf-idf/

Luiz, W., Viegas, F., Alencar, R., Mourão, F., Salles, T., Carvalho, D., … Rocha, L. (2018). A Feature-Oriented Sentiment Rating for Mobile App Reviews. In *Proceedings of the 2018 World Wide Web Conference* (p. 1909-1918). International World Wide Web Conferences Steering Committee. DOI: https://doi.org/10.1145/3178876.3186168

Maalej, W., & Nabil, H. (2015). Bug report, feature request, or simply praise? On automatically classifying app reviews. In *IEEE 23rd International Requirements Engineering Conference* (p. 116-125). Ottawa, CA. DOI: https://doi.org/10.1109/RE.2015.7320414

Maalej, W., Kurtanović, Z., Nabil, H., & Stanik, C. (2016b). On the automatic classification of app reviews. *Requirements Engineering*, *21*(3), 311-331.

Maalej, W., Nayebi, M., Johann, T., & Ruhe, G. (2016a). Toward data-driven requirements engineering. *IEEE Software*, *33*, 48-54. DOI: https://doi.org/10.1109/MS.2015.153

Malik, H., & Shakshuki, E. (2016). Mining Collective Opinions for Comparison of Mobile Apps. *Procedia Computer Science*, *94*, 168-175.

Samad, F., Asad A., Memon., Aziz A., & Rahman, A. (2018). The future of internet: ipv6 fulfilling the routing needs in internet of things. *International Journal of Future Generation Communication and Networking, 11*(1)*, 13-22.* DOI: http://dx.doi.org/10.14257/ijfgcn.2018.11.1.02

Ubaidullah, K., Zulfiqar, A. M., Shafaq, S., Balouch, A. R., & Batra, R. (2018). Architectural design of trusted platform for iaas cloud computing. *International Journal of Cloud Applications and Computing, 8*(2), 47-65. DOI: https://doi.org/10.4018/IJCAC.2018040103

Yumna, Z., Hina, K., & Memon, Z. A. (2018). On improving efficiency and utilization of last level cache in multicore systems. *Journal of Information Technology and Control*, *47*(3), 588-607. DOI: https://doi.org/10.5755/j01.itc.47.3.18433

Zulfiqar, A. M., & Syed, M. H. (2023). Predicting movie success based on pre-released features. *Multimedia Tools and Applications*. DOI: https://doi.org/10.1007/s11042-023-16319-4