**Empirical Software Engineering**
(Manuscript No. will be inserted by the editor)

# FeatCompare: Feature Comparison for Competing Mobile Apps Leveraging User Reviews

**Maram Assi · Safwat Hassan · Yuan Tian · Ying Zou**

Accepted: 26 May 2021 / Published online: 08 July 2021

Author pre-print copy. The final publication is available at Springer via:
https://doi.org/10.1007/s10664-021-09988-y

**Abstract** Given the competitive mobile app market, developers must be fully aware of users' needs, satisfy users' requirements, combat apps of similar functionalities (i.e., *competing apps*), and thus stay ahead of the competition. While it is easy to track the overall user ratings of competing apps, such information fails to provide actionable insights for developers to improve their apps over the competing apps [2]. Thus, developers still need to read reviews from all their interested competing apps and summarize the advantages and disadvantages of each app. Such a manual process can be tedious and even infeasible with thousands of reviews posted daily.

To help developers compare users' opinions among competing apps on *high-level features*, such as the main functionalities and the main characteristics of an app, we propose a review analysis approach named *FeatCompare*. FeatCompare can automatically identify high-level features mentioned in user reviews without any manually annotated resource. Then, FeatCompare creates a comparative table that summarizes users' opinions for each identified feature across competing apps. FeatCompare features a novel neural network-based model named **G**lobal-**L**ocal sensitive **F**eature **E**xtractor (GLFE), which extends Attention-based Aspect Extraction (ABAE), a state-of-the-art model for extracting high-level features from reviews. We evaluate the effectiveness of GLFE on 480 manually annotated reviews sampled from five groups of competing apps. Our experiment results show that GLFE achieves a precision of

Maram Assi · Yuan Tian · Ying Zou
Queen's University, Canada
E-mail: {maram.assi, y.tian, ying.zou}@queensu.ca

Safwat Hassan
Department of Engineering, Thompson Rivers University, Kamloops, BC, Canada
E-mail: shassan@tru.ca

79%-82% and recall of 74%-77% in identifying the high-level features associated with reviews and outperforms ABAE by 14.7% on average. We also conduct a case study to demonstrate the usage scenarios of FeatCompare. A survey with 107 mobile app developers shows that more than 70% of developers agree that FeatCompare is of great benefit.

**Keywords** Mobile applications, User reviews, Competitor analysis, Competing apps, Feature extraction, Google Play Store

## 1 Introduction

Mobile applications (apps) have become an integral part of our daily activities. To get an app, users visit a mobile app store, such as the Google Play Store, and search for an app that fulfills their needs [29]. After downloading an app, users can post reviews to express their opinions on the downloaded app. Table 1 shows three sample reviews from two popular weather apps. In this example, two users of the *"AccuWeather"* app complain about the low accuracy of the forecast given by the app. In contrast, one user of the *"Weather Live"* app praises the accurate forecast of weather provided by the app.

Table 1: User reviews of the *"AccuWeather"* app and the *"Weather Live"* app.

| App | User reviews | Rating | High-level feature |
|---|---|---|---|
| *AccuWeather* | *"**Terrible accuracy**. Keeps telling me it's snowing. No snow. Not a flake."* | ★☆☆☆☆ | Forecast accuracy |
| *AccuWeather* | *"**Inaccurate** weather **forecasts**. Says its partly sunny where it is actually overcast"* | ★★☆☆☆ | Forecast accuracy |
| *Weather Live* | *"**Excellent accuracy** in forecasting the weather for my area"* | ★★★★☆ | Forecast accuracy |

User reviews are crucial for mobile app developers [15, 42] as it has been reported that mobile users are not likely to download an app with an average rating that is less than three stars [34]. To maintain a high average rating, developers need to rapidly implement the requested features and fix the reported bugs mentioned in user reviews [42]. The manual process of reading a large number of user reviews can be tedious, expensive, and error-prone. Hence, in literature, researchers have proposed approaches to help mobile app developers gain insights from their user reviews by automatically extracting features mentioned in reviews [15, 16, 21, 23, 52], summarizing [12, 57], categorizing [33, 36, 43, 55], and prioritizing user reviews [6, 13, 24].

However, given the highly competitive nature of the mobile app market, analyzing the reviews of a specific app is not enough for developers of the app

to succeed. El Zarif et al. [10] find that user complaints can lead to user churn, which means users can switch to start using the competitors' products. Developers need to expand their focus beyond their own apps and understand how users perceive their apps with respect to their competitors' apps [2, 40, 48, 55] via a competitor analysis [53]. Conducting user-based competitor analysis support developers in eliciting requirements requested by users [48, 55], planning the app releases [40] and performing apps enhancements [2]. Competitor analysis can be conducted in many ways and for different purposes. In this work, inspired by the existing user review analysis approaches, we specifically explore the opportunity of supporting mobile app competitor analysis via automatically mining useful information from reviews of closely related apps (i.e., a group of *competing apps*). Studies show that reading about app features in the app description section does not provide any information about the quality of the features and how it is perceived by users [14, 32]. Therefore, researchers show interest in techniques that provide developers with suggestions (e.g., bug fixes and feature requests) extracted from user reviews for future app releases [2, 55].

Nayebi et al. [39] suggest that accommodating user opinions is crucial for planning the future releases of the apps in the competitive market. Through the feature-level comparison, developers can focus on either improving the most frequently mentioned low rate features (i.e., requests and complaints) [39] or on enhancing features that are less rated (i.e., most negative as compared to the same feature rating of competitor app. For example, for the "Forecast accuracy" high-level features of the "Weather" app shown in table 1, the developer can know how many users report negative opinions about this particular feature and compare it to the same feature rating across competitors along with the number of satisfied users.

Few prior studies [8, 28, 50, 53] have shed light on the potential of extracting useful information, i.e, user opinions (sentiment and rating) associated with specific app *features*, for app comparisons. These app features are fine-grained, and explicitly mentioned in reviews via word pairs such as "*photo edition*" and "*upload video*". However, comparing competing apps on fine-grained features can be challenging and time-consuming as the number of extracted fine-grained features can reach up to hundreds or even thousands per competing group [8, 51]. Instead of mining the fine-grained features, which describe the details of an app feature, we propose to mine and compare user opinions on *high-level features* from reviews of competing apps. High-level features refer to the main functionalities (e.g., video streaming or daily forecast) and the main characteristics (e.g., UI appearance and stability) of an app. One can regard high-level features as semantic clusters of fine-grained features. For instance, the reviews shown in Table 1 mention multiple fine-grained features specified by phrases *"Terrible accuracy"*, *"Inaccurate Forecast"*, and *"Excellent accuracy"*. The three fine-grained features can be grouped into one high-level feature, i.e, " *"Forecast accuracy"*, indicating the common concern from all sample reviews. By identifying all high-level features from reviews of com-

peting apps, developers can summarize users' attitude on forecast accuracy among all competing apps.

In this paper, we propose **FeatCompare**, an approach that facilitates the analysis of high-level features across competing apps. Our approach contains three components: 1) a *data preprocessor* component that takes as input reviews from multiple competing groups and removes non-informative reviews using AR-Miner [6], 2) an unsupervised neural network-based *feature extractor* component, named **G**lobal-**L**ocal sensitive **F**eature **E**xtractor (GLFE), that extracts high-level features from preprocessed user reviews, and 3) a *review aggregator* component that aggregates ratings for each feature and generates a comparison table summarizing feature-based user opinions for competing apps. GLFE extends a state-of-the-art high-level feature extraction model named Attention-based Aspect Extraction (ABAE) [19] by proposing a threshold mechanism that can identify *global features* (e.g., general software engineering features shared across multiple app groups) and *local features* (e.g., features shared among a specific group of competing apps) discussed in reviews.

To evaluate the effectiveness of our approach, we collected 14,043,999 reviews from 196 popular Android apps. The studied 196 apps are distributed across 20 functionality-similar groups, e.g., weather apps, music player apps, etc. Our experimental results show that GLFE achieves an average precision of 81% and an average recall of 75%, and outperforms ABAE by 14.7%, on 480 randomly selected reviews from five competing app groups. We then conduct a case study on opinions summarized from the studied 196 competing apps. We observe that using FeatCompare to analyzing the reviews of an app, within the context of its competitors, can help app developers spot the potential opportunities for improving their app. A user study involving 107 developers is conducted, and the results show that our FeatCompare approach is useful for app developers to perform competitor analysis.

The main contributions of our work are as follows:

- We propose an approach named FeatCompare to automatically mine and compare high-level feature-based user opinions for competing mobile apps from user reviews.
- We apply the feature extraction component GLFE in FeatCompare on 3.4 million reviews of 196 closely related apps across 20 apps groups and evaluate its performance using 480 randomly selected reviews of apps from five groups.
- We conduct a case study on features and opinions extracted from the studied 196 competing apps and we demonstrate the capabilities of FeatCompare by generating comparison tables for five different app groups.
- We conduct a survey with 107 mobile developers to investigate how the mobile developers perform competitor analysis in practice and verify the applicability of our approach.

– We make our manually labeled data and the implementation of FeatCompare available online in a replication package [1] as a means to enable future work on competitor analysis of mobile apps.

**Paper organization**: The rest of the paper is organized as follows. Section 2 describes the Attention-Based Aspect Extraction (ABAE) model. Section 3 introduces the design details of FeatCompare. Section 4 presents the data collection process. Section 5 discusses the motivation, approach, and findings of research questions that guide our evaluation process. Section 7 discusses the limitations of this work. Section 8 illustrates close related work on mining mobile app reviews. Finally, Section 9 concludes the paper.

## 2 Background

Our work adopts and augments advanced neural network techniques. In this section, we introduce ABAE, the neural network model that we utilize in Section 3.

ABAE is one of the state-of-the-art unsupervised neural attention models proposed by He et al. [19] for extracting high-level features (i.e., "aspect" in their paper) from product/service reviews. Prior to ABAE, topic models such as Latent Dirichlet Allocation (LDA) and its variants have been widely applied to user reviews such as Amazon product reviews or Yelp restaurant reviews to extract high-level features [38, 58]. However, researchers find that while LDA can describe high-level features in text corpus fairly well, the mined individual features are of poor quality with unrelated or loosely-related words, which often leads to low accuracy in identifying high-level feature of a specific review [19]. ABAE is then proposed and has been shown to outperform LDA-based approaches. The main idea of ABAE is to learn the semantic meaning of high-level features, reviews, and words as vectors (i.e., embeddings) so that fine-grained features explicitly expressed in reviews can be mapped to their corresponding high-level features.

Figure 1 shows an example of the ABAE architecture using a mobile app review. At high-level, ABAE is an autoencoder [56] that can learn the embeddings of reviews and high-level features automatically using only reviews (e.g., "*excellent accuracy in forecasting the weather for my area*"). ABAE requires three input elements: 1) a matrix representing pre-trained embeddings of words, 2) an integer representing the number of high-level features expected to be learned, and 3) a set of reviews. The output of ABAE includes embeddings of reviews and high-level features, probabilities of each review belonging to each of the high-level features.

Given the list of words in a review as input, two steps are performed as shown in Figure 1. ABAE first deemphasizes words that are not relevant to high-level features, such as "*the*", "*for*", "*my*" in the example review using an attention mechanism, and constructs a word-based review embedding $z_r$.

[1] https://github.com/maramassi/suppmaterial-featcompare

Next, ABAE reconstructs the review embedding ($t_r$) as a linear combination of high-level feature embeddings. All the parameters in ABAE are learned by minimizing the reconstruction loss of the word-based review embedding $z_r$ and the feature-based review embedding $t_r$, aiming to preserve most of the information of the feature-related words (e.g., "*excellent*" and "*accuracy*") in the embedded high-level features (e.g., "*accurate forecast*"[2]). Next, we introduce the two steps of ABAE in detail.

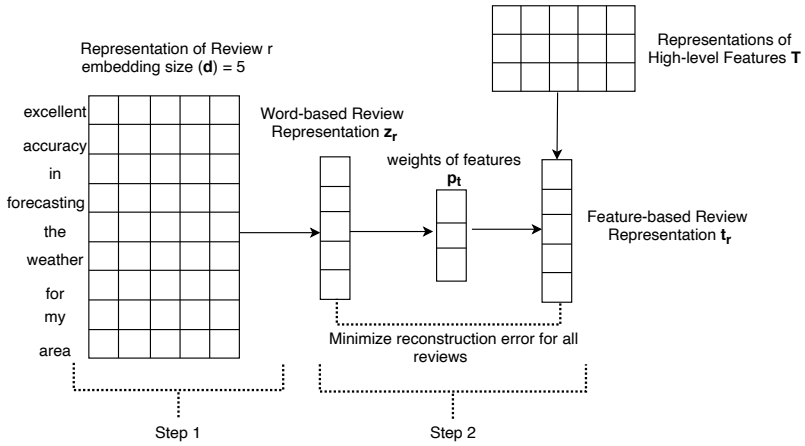


Fig. 1 An example of ABAE architecture. In the above sample, the size of word embedding is 5, the number of high-level features is 3.

The first step in ABAE computes the embedding $z_r \in R^d$ of a review $r$ that contains $n$ words $w_1, w_2, \ldots, w_n$:

$$z_r = \sum_{i=1}^{n} a_i e_{w_i}$$

where $e_{w_i}$ refers to the word embedding $e$ of size $d$ for word $w_i$ in review $r$. The word embeddings are initialized by applying word2vec [37] over a collection of reviews. word2vec is an unsupervised algorithm that learns meaningful embeddings of words from a text corpus. The weight $a_i$ is conditioned on the embedding of the word $e_{w^i}$ as well as the global context of the review:

$$a_i = softmax(e_{w^i}^T \cdot M \cdot y_r)$$

$$y_r = \sum_{i=1}^{n} e_{w^i}$$

[2]Note that all high-level features are hidden, meaning they are represented using embeddings. The semantic meaning of a high-level feature can be identified by searching the most representative words, the embeddings of which are close to the embedding of the high-level feature.

where $y_r$ is the uniformly-weighted bag-of-words embedding of the review, and $M \in R^{d \times d}$ is a learned attention model. This attention layer is designed to reduce the significance of the words that are not relevant to any high-level features, and focus more on the more high-level feature related words.

The second step computes the high-level feature-based review representation $t_r \in R^d$ in terms of a high-level feature embedding matrix $T \in R^{K \times d}$, where $K$ is the number of high-level features:

$$p_t = softmax(W \cdot z_r + b)$$

$$t_r = T^T \cdot p_t$$

where $p_t \in R^K$ is the weight vector over $K$ feature embeddings, and $W \in R^{K \times d}$, $b \in R^K$ are the parameters of a multi-class logistic regression model.

The parameters in ABAE are trained to minimize the reconstruction error, i.e., the cosine distance between $t_r$ and $z_r$. Representative words of a high-level feature can be found by ranking all words based on the cosine similarity between the word embedding and feature embedding. To assign a high-level feature to each review, the learned parameter $p_t$ is used to pick the high-level feature that is of the highest weight.

## 3 FeatCompare

In this section, we introduce the design of FeatCompare, a neural network-based approach that can identify high-level features and summarize corresponding user opinions for competing mobile apps from app reviews. Notably, FeatCompare is an unsupervised machine learning approach, which means it does not require any manual labeled reviews in training. The only human effort needed in FeatCompare is determining hyper-parameter value and feature names.

**Overall Approach.** FeatCompare contains three main components: the rating preprocessor, GLFE and the rating aggregator, as shown in Figure 2. The first component takes as input a set of mobile app reviews collected from a variety of apps, including those selected competing apps, and then **filters non-informative reviews**. The second component is a GLFE, which stands for a global-local sensitive feature extractor. GLFE takes as input the informative reviews and a pre-defined feature number $K$. GLFE then automatically **identifies global (general) and local (domain-specific) high-level features** in reviews. The rating aggregator, last component of FeatCompare, **aggregates user ratings** (opinions) for each identified feature within each selected app and then creates a comparative table of competing apps. In the following sections, we describe the design details and the rationales behind specific design decisions for each component.

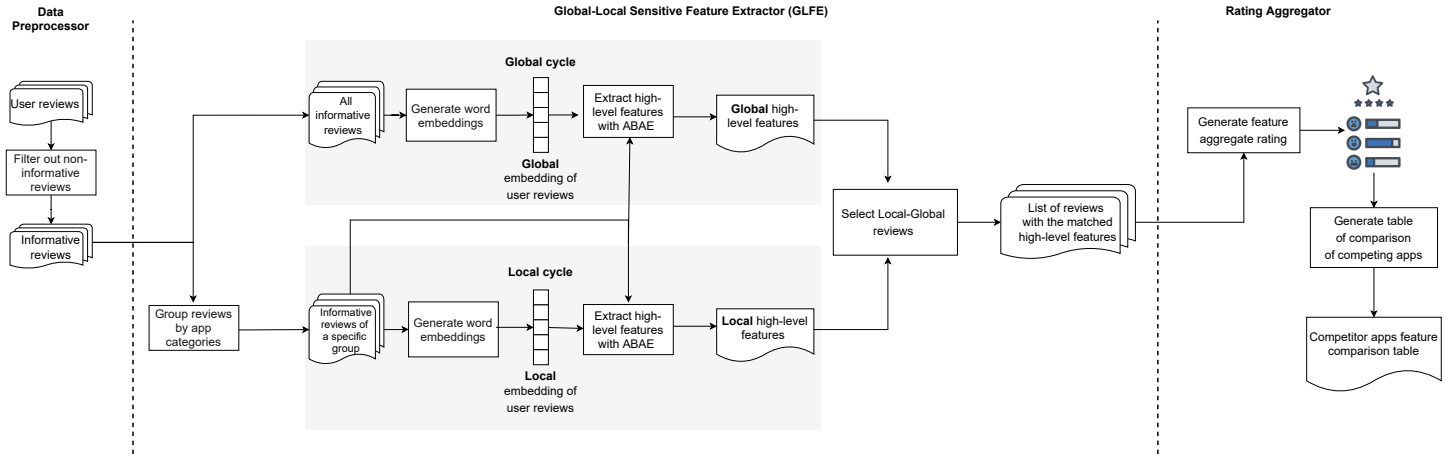


Fig. 2: FeatCompare three main components.

### 3.1 Data Preprocessor

User reviews can be non-informative (i.e., do not contain useful information), such as *"I will give one star"*. Thus the first step in FeatCompare is to filter out non-informative user reviews and keep only the reviews that contain valuable information. First, we filter out reviews without any text. Next, we feed the filtered user reviews of all apps to AR-Miner [6]. We leverage AR-Miner as it is a commonly used approach to filter non-informative reviews [41, 43]. The definition of informative and non-informative reviews in our work aligns with the definition used by AR-Miner's authors. We consider a user review as informative if the review carries potential helpful information that can be leveraged by the developers to enhance the quality of the app or improve the app's usability. For example, "Pro version still needs an ad free version" and "It does not give location" are informative user reviews that bring insight into feature requests or functional flaws. Non-informative reviews such as "This app is pointless" and "I don't like this app, I will give only 3 stars" provide no constructive information for app developers to guide them on how to improve the quality of the app.

AR-Miner classifies reviews into informative reviews and non-informative reviews. We consider only the subset of informative user reviews in the following steps of our approach. Finally, the length of the review content (i.e., the review title and the review description) can impact the usefulness of the user reviews [25, 54]. Shorter reviews are less likely to be meaningful. Among the subset of three million informative reviews, 191,417 reviews, e.g., 5%, are two words in length or less. We manually check a statistical representative random sample, ie., 96 user reviews, with a confidence level of 95% and a confidence interval of 10% of the up to two words user reviews. We find that 87% of the reviews with less than three words in length, i.e., "useless app", "hate this", "rarely works", and "nothing worked", are non-informative. Therefore, we follow previous work [5, 27] and filter out reviews that contain less than three words in the content.

### 3.2 Global-Local Sensitive Feature Extractor

GLFE is the main component of FeatCompare that elicits high-level features from user reviews. As shown in Figure 2, GLFE contains three steps: identifying global high-level features based on all collected app reviews across multiple groups, identifying local high-level features based on only reviews of selected competing apps, and assigning one high-level feature for each review of the selected apps.

#### *3.2.1 Identifying Global and Local Features*

We observe that in mobile app reviews, users might choose to comment on either group-specific features, such as "*accurate forecast*" for weather apps,

or general features that are common among apps of all categories, such as "*device compatibility issues*" and "*performance issues*". To utilize this unique characteristic of mobile app reviews, we introduce the concepts of local and global high-level features and extract these two types of features simultaneously. *Global high-level features* are general features not related to any specific app domain but rather representing software engineering features shared among all types of mobile apps. *Local high-level features* are specific features shared among the selected competing apps within a particular group.

GLFE contains two cycles, i.e., a global cycle for learning global high-level features and a local cycle for learning local high-level features (ref. Figure 2). Both cycles follow the design of ABAE but with separately learned word embeddings. Several embedding methods have been used in sentiment analysis research to construct vector representations of words.

The local cycle in GLFE takes informative user reviews of selected competing apps as input. Next, the reviews are filtered following standard text normalization steps, including tokenization, removing non-English words, and stop words (i.e., frequent but meaningless English words), and lemmatization. As introduced in Section 2, ABAE requires three inputs (1) an initial lookup table that contains word embeddings, (2) a set of target sentences to extract features, and (3) a parameter determining the number of learned high-level features. In the local cycle, local high-level features are extracted by applying ABAE on reviews only from competing apps in one specific group. The word embeddings are learned from the same set of reviews via word2vec. Following the paper proposing ABAE [19], we use a heuristic approach to determine the value of the number of features based on our observations on the quality (higher feature coherence and a lower degree of overlapping) of learned features. After running ABAE, the local cycle of GLFE assigns a local high-level feature to each review in the selected group.

In parallel and independently, the global cycle of GLFE utilizes an ABAE model to extract high-level global features from reviews of apps in the selected group. Unlike the local cycle, the word embedding used as input for ABAE in the global cycle is learned from all the 3,439,819 informative reviews of apps across multiple groups. We use 3,439,819 user reviews to generate global embedding. Our intuition is that word embeddings learned from various groups can better capture the semantics of words that are relevant to general features because these words appear relatively more often than domain-specific words (i.e., words describing unique features in a specific type of apps) in the reviews of different apps. Like the local cycle, we use manual observation to determine the best number of high-level features associated with each selected set of similar apps.

After running the local and global cycles, for each word appearing in the reviews of selected similar apps, GLFE learns a global word embedding representing the meaning of the word in the context of all app reviews and a local word embedding representing the meaning of the word in the context of selected apps. Meanwhile, two sets of high-level features and their associated most representative words are also learned. Note that, ABAE does not output

a feature name for each learned high-level feature. Following He et al. [19], the first three authors of this paper manually and independently label the features based on the representative keywords using open coding [47, 49]. The authors then discuss the conflicts in a consensus meeting where they resolved one by one the disagreements and determine the final feature names.

The design of global feature and local feature learning in GLFE poses a unique challenge for determining the final high-level feature of each review, i.e., each review now is assigned two high-level features, one from each cycle. We explain in the next subsection how GLFE solves this challenge.

### *3.2.2 Global-Local Feature Selection*

We propose a metric, named *mean-norm-tfidf*, to approximate the likelihood of a review being assigned to its local high-level feature, i.e., reviews with high mean-norm-tfidf select the features learned from the local cycle of GLFE. mean-norm-tfidf is the average of values in the L2 normalized tf-idf [45] vector representation of a review. We formally define the notion of mean-norm-tfidf as follows. Given a review $r_i$ containing $n$ unique words $w_1, w_2, \ldots, w_n$ and a review corpus $C$, the mean-norm-tfidf of the review $r_i$ is defined as:

$$\textit{mean-norm-tfidf}_{r_i,C} = \frac{1}{n}\sum_{k=1}^{n} norm(f_{w_k,r_i} \cdot \log \frac{M}{|d \in C : w_k \in d|})$$

where $f_{w_k,r_i}$ refers to the number of times the word $w_k$ appears in review $r_i$, i.e., the term frequency of $w_k$. $M$ is the total number of reviews in $C$, $|d \in C : w_k \in d|$ refers to the number of reviews in $C$ that contain word $w_k$. $\log \frac{M}{|d \in C : w_k \in d|}$ is the inverse document frequency of word $w_k$. *norm* is a function[3] that normalizes a value to the range of 0 to 1. Note that the corpus $C$ in this paper refers to all informative reviews of apps across multiple groups, not just the reviews of the selected competing apps.

The main design concerns behind the mean-norm-tfidf metric are two folds. First, the metric should be positively related to the inverse document frequency of words in the review because domain-specific words that do not frequently appear in a variety of app groups have a larger inverse document frequency. If a review contains many domain-specific words, then the review is highly likely to discuss a local high-level feature. On the other hand, if a review contains many words with low inverse document frequency, such as "*ads*" and "*crash*" that frequently appear in reviews of apps across all groups, the review is highly likely to discuss a global high-level feature. Secondly, words that occur many times in the review should be assigned a higher weight as they often present the main concern in a review.

However, calculating a mean-normed-tfidf score for each review is not sufficient. We need a threshold value to determine if the local/global feature should be selected. For instance, if we set a threshold at 0.5, then every review with a

[3]We normalize document vectors to unit Euclidean length.

mean-norm-tfidf value larger than 0.5 is assigned to its local high-level feature, or its global high-level features, otherwise. To simplify the process, we rank all reviews in the selected competing apps in ascending order based on their mean-norm-tfidf scores and pick the $25^{\text{th}}$ percentile (lower quartile), the $50^{\text{th}}$ percentile (median), and the $75^{\text{th}}$ percentile (higher quartile) as the three considered thresholds. In practice, stakeholders could tune this hyper-parameter using few groups of competing apps.

### 3.3 Rating Aggregator

As the final component of FeatCompare, the rating aggregator takes as input the informative reviews extracted from the selected competing apps with their extracted high-level features by GLFE and outputs a comparative table.

We define two main metrics of the rating aggregator: the **F**eature **A**verage **R**ating (FAR) and the **C**ompetitive **F**eature **A**verage **R**ating (CFAR). FAR represents the mean of star ratings of one specific feature of an app. CFAR represents the mean score of one feature across all the apps of the same group. Formally, given the set of $w$ competing apps $A = \{a_1, a_2, ..., a_w\}$, and their $p$ user reviews $R = \{r_1, r_2, ..., r_p\}$ that are relevant to a set of $n$ high-level features $F = \{f_1, f_2, ..., f_n\}$, the rating aggregator first calculates a feature average rating $FAR$ of a specific feature $f_j$ in relation to app $a_i$ as follows:

$$FAR_{a_i,f_j} = \frac{\sum_{k=1}^{m} rating(r_k)}{m}$$

where $r_k$ refers to any review of the app $a_i$ that is assigned to the high-level feature $f_j$. $m$ is the total number of reviews of the apps $a_i$ that are assigned to $f_j$. $rating(r_k)$ refers to the user rating associated with the review $r_k$.

FeatCompare also calculates a competitive feature rating $CFAR$ of a specific feature $f_j$ among $w$ competing apps as follows.

$$CFAR_{f_j} = \frac{\sum_{i=1}^{w} FAR_{a_i,f_j}}{w}$$

$CFAR$ represents how the overall user-base ($R$) belonging to a group of competing apps ($A$) perceives a specific feature $f_j$. If the FAR value of an app $a_i$ is higher than CFAR, this means, on average, app $a_i$ receives more positive feedback than its competitor's apps on feature $f_j$.

In the resultant comparative table, FeatCompare provides the following information:

1 Name of each considered high-level feature.
2 Feature Average Rating (FAR) of each considered feature in each selected app.
3 Total number of reviews within each app that are relevant to each considered feature.
4 Distribution of ratings on the relevant reviews to each considered feature.
5 Competitive Feature Average Rating (CFAR) for each considered high-level feature.

## 4 Dataset

Table 2: Statistics of 20 competing apps groups.

| App group | Google category | Used keywords | # of apps | # of reviews |
|---|---|---|---|---|
| **Taxi and rideshare** | Maps and navigation | **taxi*, *rideshare*, *share-ride** | 9 | 337,009 |
| **Navigation** | Travel and local | **navigation*, *gps*, *map** | 10 | 328,719 |
| **Security** | Tools | **virus*, *malware*, *security** | 10 | 312,758 |
| **Browser** | Communication | **browser** | 10 | 292,103 |
| **Free call** | Communication | **free call** | 10 | 291,034 |
| **News** | News and magazines | **news** | 10 | 283,711 |
| **Weather** | Weather | **weather** | 10 | 276,941 |
| **SMS** | Communication | **sms** | 10 | 264,926 |
| **Dating** | Dating | **dating** | 10 | 200,236 |
| **Wallpaper** | Personalization | **wallpaper** | 10 | 183,405 |
| **Notes** | Productivity | **notes*, *notepad** | 8 | 133,665 |
| **Video editor** | Video player and editor | **video editor** | 10 | 126,357 |
| **Hotel booking** | Travel and local | **hotels** | 10 | 69,699 |
| **Bible** | Books and reference | **bible** | 10 | 64,417 |
| **Mobile banking** | Finance | **mobile banking** | 10 | 64,371 |
| **Music player** | Music and audio | **music player** | 10 | 60,685 |
| **Sports news** | Sport | **sports news** | 10 | 57,582 |
| **Cooking recipe** | Food and drink | **recipe*, *cooking** | 10 | 41,154 |
| **Pregnancy** | Health and fitness | **pregnancy** | 9 | 29,816 |
| **Coloring** | Creativity | **coloring** | 10 | 21,231 |
| **Total** | | | 196 | 3,439,819 |

**Identifying Groups of Competing Apps.** To begin with, we select the top 2,000 free-to-download popular apps from the Google Play Store as the candidate target apps. The popularity of the apps is decided based on the App Annie report [3]. We then collect all general information available on the Google Play Store, including the number of reviews and the app description for the 2,000 popular apps using a web crawler [1]. Our study mainly focuses on the most popular apps as these apps contain rich review data that facilitate our analysis of user reviews. Moreover, developers may wish to compare their apps to the most successful apps in the market.

From the 2,000 popular apps, we identify *groups of competing apps* (i.e., apps sharing similar functionalities and business domains) by applying the following steps. First, we identify a set of keywords that represent a main app feature to form 20 different apps groups in total. The selected 29 keywords (i.e., main functionality of the app) were chosen to identify groups of similar apps. The first two authors have attentively chosen the keywords to cover multiple Google apps categories, so a specific app category does not bias our study. Hence, we covered 17 Google categories. For example, we use the keywords *"*weather*"* and *"*browser*"* to represent the weather forecast apps and the browser apps, respectively. Next, for each of the considered main app features illustrated in table 2, we search apps with the corresponding keywords mentioned in their names or descriptions using the collected information from the 2,000 popular apps. We then rank the matched apps by their review numbers. From the top of the ranked list, the first two authors manually read the name

and the description of each encountered app to verify whether the app indeed contains the corresponding main feature. For each main app feature, we filter out apps that do not contain the feature and stop checking until we collect ten apps. We choose ten apps for each group, so that our analysis is not biased towards any particular app group. Besides, intuitively, we do not expect each app to have a very large (e.g., 50) number of competing apps, implying that an app can be replaced by as many as 50 other apps that are in the top 2,000 popular apps in the whole market.

To validate that the selected candidate apps for each main feature are closely related, the first two authors independently read the page of every selected app in the Google Play Store and check if the other nine apps in the same group appear in the "*similar*" app list recommended by the store. In the end, we find that all the candidate apps satisfy the above requirement. Table 2 summarizes the basic statistics of the 20 selected competing app groups along with their categories at the Google Play Store. Some groups, i.e, "*Taxi and ride share*", "*Notes*", "*Pregnancy*", have less than ten apps because there are less than ten apps match with our searched keywords. Expanding the search to consider more than 2,000 popular apps from App Annie, might lead to find ten competing apps for the aforementioned categories. However, we believe that different categories can have a different number of competitors. Since we have covered a sufficient number of groups (e.g., 20 apps domain) and each group contains enough competing apps (e.g., 8, 9 and 10), we only consider the top 2,000 popular apps.

We follow the above practice as a proof of concept. Nevertheless, we expect that even the developers of an app may have different sets of competing apps formed based on different goals. For instance, developers can choose to compare their apps with the similar paid apps or freemium apps.

**User Reviews Collection.** For each selected app, besides the general information, we collect its user reviews over 3.5 years, starting from April 2016 until January 2019. In total, we collect 14,043,999 user reviews. For each review, we record the title of the user review, the detailed comment text, user rating, and the post date of the review. Table 3 summarizes the basic statistics of the collected reviews.

Table 3: Dataset Statistics.

| | |
|---|---|
| Number of studies apps | 196 |
| Number of initial user reviews | 14,043,999 |
| Number of non-empty text user reviews | 13,847,602 |
| Number of informative user reviews | 3,631,236 |
| Number of user reviews with more than two words | 3,439,819 |

## 5 Evaluation and Results

In this section, we evaluate the effectiveness and usefulness of FeatCompare for comparing high-level features among competing mobile apps based on user reviews. Specifically, we discuss motivation, approach, and findings for the following research questions.

### 5.1 RQ1: How effective is our global-local sensitive feature extraction approach GLFE?

**Motivation:** The effectiveness of FeatCompare relies on the accuracy of its data-driven feature extractor component, i.e., GLFE. Thus for FeatCompare to be useful in practice, we need to evaluate the effectiveness of GLFE in identifying the high-level feature associated with each informative review.

**Approach:** To evaluate the performance of GLFE, we randomly pick five groups from the 20 identified competing app groups (ref. Table 2). The selected groups are "*Weather*", "*Sports news*", "*Bible*", "*SMS*", and "*Music player*", respectively.

Then, for each app group, we select a statistically representative random sample of user reviews with a confidence level of 95% and a confidence interval of 10%. In total, we select 480 informative reviews that belong to the selected five groups. Next, the ground truth high-level features of the testing reviews are obtained by performing the following steps:

1. To achieve candidate high-level features, for each of the five selected app groups, we apply the global cycle and local cycle of GLFE on the reviews output by the data preprocessor component of FeatCompare. At the end of this step, we achieve a set of local and global high-level features for each group of competing apps.
2. The first three authors independently annotated the 480 testing reviews using the candidate high-level features of the corresponding app group. For instance, each testing review from the app group "*Weather*" should be assigned to the corresponding high-level features extracted from the "*Weather*" group in the first step. At least two annotators annotate each testing review.
   It should be noted that one user review can discuss multiple features. For instance, the following review comment *"Very user friendly app and I find the alerts warning of severe weather conditions very helpful."* contains information about the *"User Interface"* feature and the *"Weather Alert Services"* feature. Hence, in this step, the created gold dataset contains all valid features that are mentioned in the review (i.e., the *"User Interface"* and the *"Weather Alert Services"* feature). We find that only 8.6% of the manually labeled reviews contain multiple features.

3. The Fleiss's Kappa agreement score [35] is calculated on the annotated testing reviews using the "irr" package [4] provided in R [5] and we achieve a score of 0.83, which indicates a high level of agreement among annotators. The annotators then discuss the conflict cases and resolve all disagreements.

Following the original design of ABAE [19], we set the default number of high-level features as 14 and vary the number by increasing and decreasing it. We find that the default setup consistently provides high quality high-level features for all app groups. Thus, we decide to keep the number of expected global/local high-level features as 14.

GLFE has one hyper-parameter, the mean-norm-tfidf threshold for the global-local feature selection step. We considered three values for this hyper-parameter, leading to three models, named (*GLFE-25$^{th}$*, *GLFE-50$^{th}$*, and *GLFE-75$^{th}$*), respectively. The 25th percentile means that for the 25% reviews with the lowest mean-normed tfidf, we assign them the identified global features, and the rest 75% reviews are assigned with their local features. GLFE with a higher value threshold (percentile) represents a model that prefers global high-level features over local high-level features. We select the best value for the mean-norm-tfidf threshold by evaluating the performance of three models (*GLFE-25$^{th}$*, *GLFE-50$^{th}$*, and *GLFE-75$^{th}$*) on a validation set, consisting of two competing app groups, i.e., "Recipe cooking" and "Free call". Table 4 shows that on the validation set, GLFE-25th achieves the highest accuracy. Thus we use the 25th percentile as the threshold when applying GLFE on five testing app groups."

The rest of GFLE parameters are set based on the best performing parameter sets reported in the original ABAE paper [19]. Specifically, we initialize the word embedding matrix with word vectors trained by word2vec, setting the embedding size to 200, the window size to 10, and the negative sample size to 5. We initialize the feature embedding matrix with the centroids of clusters resulting from running k-means on word embeddings. Other parameters are initialized randomly. During the GLFE training process, we fix the word embedding matrix and optimize other parameters using Adam [22] with the learning rate of 0.001 for 15 epochs and the batch size of 50.

**Evaluation Metrics:** Since user reviews may contain multiple features, albeit a small percentage, we model the high-level feature identification task as a multi-label classification task. Given a set of testing reviews with *gold* high-level features (i.e., ground truth) and the predicted features, we evaluate the feature extraction approaches as follows. First, we measure the true positive (TP) as the number of successfully predicted features, the false positive (FP) as the number of falsely predicted features (i.e., features predicted by the approach but are not mentioned in the reviews), and the false negative (FN) as the number of features mentioned in the reviews but are not predicted by the feature extraction approach. We consider precision, recall, and $F_1$-Score as the evaluation metrics. Equations 1, 2, and 3 show the computation

[4] https://cran.r-project.org/web/packages/irr/index.html
[5] https://www.r-project.org/

for precision, recall, and $F_1$-Score. Precision measures the percentage of true positive predictions among all the predictions made by the evaluated approach. Recall represents the percentage of the features that can be predicted by the approach among all the features in the ground truth. Finally, $F_1$-Score is the harmonic combination of precision and recall.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F_1\text{-}Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

Table 4: Accuracy of GLFE for different local global hyper-parameter threshold on two validation app groups. Prec. represents the precision.

| **App Group** | **GLFE-25$^{\text{th}}$** | | | **GLFE-50$^{\text{th}}$** | | | **GLFE-75$^{\text{th}}$** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Prec. | Recall | $F_1$ | Prec. | Recall | $F_1$ | Prec. | Recall | $F_1$ |
| Recipe Cooking | 0.78 | 0.70 | 0.74 | 0.40 | 0.36 | 0.38 | 0.34 | 0.31 | 0.32 |
| Free Call | 0.80 | 0.73 | 0.76 | 0.47 | 0.43 | 0.45 | 0.38 | 0.35 | 0.36 |
| Average | **0.79** | **0.71** | **0.75** | 0.43 | 0.39 | 0.41 | 0.36 | 0.33 | 0.34 |

**Baselines:** To investigate the benefit of our adaptation on the original ABAE model, we compare the performance of GLFE with the performance of ABAE using the 480 labeled reviews. Specifically, we consider two different embedding matrices as input for ABAE, reviews from the selected similar apps (i.e., local reviews and the original setup), and reviews from apps across multiple groups (i.e., global reviews). We name the two variants of ABAE on our task as *ABAE* and *ABAE-global*, respectively. Note that ABAE can be treated as *GLFE-0$^{th}$*, where every review is assigned to its local high-level feature, and ABAE-global can be treated as *GLFE-100$^{th}$* where every review is assigned to its global high-level feature.

**Results: GLFE-25$^{\text{th}}$ achieves $F_1$-Score of 77-79% on five testing app groups, which outperforms the baselines and other variants of GLFE.** Table 5 shows the results of five considered approaches, i.e., GLFE with three feature selection thresholds and two ABAE models with global and local word embedding matrices. We can observe that the best performing GLFE model is the one with the 25$^{\text{th}}$ percentile threshold, with an average precision of 81% and an average recall of 75% across five testing groups. In this setting of GLFE, 25% reviews are assigned to their global high-level features, and the rest 75% are assigned to their local high-level features. Our experiment results also show that, on average, GLFE model (i.e., GLFE-25$^{\text{th}}$) can improve

Table 5: Accuracy of GLFE and ABAE on five testing app groups. Prec. represents the precision.

| **App Group** | **GLFE-25$^{th}$** | | | **ABAE** | | | **ABAE-Global** | | | **Global embedding reviews** | **Local embedding reviews** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec. | Recall | $F_1$ | Prec. | Recall | $F_1$ | Prec. | Recall | $F_1$ | | |
| Weather | **0.81** | **0.74** | **0.78** | 0.70 | 0.64 | 0.67 | 0.32 | 0.29 | 0.31 | 3,439,819 | 276,941 |
| Sports news | **0.82** | **0.75** | **0.78** | 0.71 | 0.65 | 0.68 | 0.28 | 0.26 | 0.27 | 3,439,819 | 57,582 |
| Bible | **0.81** | **0.77** | **0.79** | 0.72 | 0.69 | 0.70 | 0.36 | 0.34 | 0.35 | 3,439,819 | 64,417 |
| SMS | **0.80** | **0.74** | **0.77** | 0.67 | 0.62 | 0.64 | 0.15 | 0.14 | 0.14 | 3,439,819 | 264,926 |
| Music player | **0.79** | **0.75** | **0.77** | 0.70 | 0.66 | 0.68 | 0.21 | 0.20 | 0.20 | 3,439,819 | 60,685 |
| Average | **0.81** | **0.75** | **0.78** | 0.70 | 0.65 | 0.68 | 0.21 | 0.20 | 0.20 | | |

the $F_1$-Score of ABAE trained with reviews from competing apps by 14.7% ((0.78 − 0.68)/0.68).

**Both global high-level features and local high-level features contribute to the overall performance of GLFE.** As shown in Table 5, the two ABAE models trained from global and local embeddings perform worse than the GLFE-25$^{\text{th}}$ model, with an average $F_1$-Score of 0.20 and 0.68 respectively. As reviews in the ABAE model all choose the learned local high-level features and reviews in the ABAE-global model all choose the learned global high-level features, the above results imply that only relying on one type of features does not lead to a good performance. Another observation we can make is that GLFE-25$^{\text{th}}$ performs better than the other two settings, i.e., GLFE-50$^{\text{th}}$ and GLFE-75$^{\text{th}}$, on all five testing app groups. This observation indicates that the threshold value used for global-local high-level feature selection can impact the performance of GLFE, and local high-level features are more valuable than global high-level features.

**Summary of RQ 1**

The feature extractor GLFE in FeatCompare with 25 percentile feature selection threshold achieves a promising average $F_1$-Score of 78% on 480 manual annotated reviews, which surpasses the performance of the state-of-the-art neural network-based high-level feature extractor model ABAE by 14.7%.

### 5.2 RQ2: Is FeatCompare able to find and compare meaningful high-level features among competing apps? Is FeatCompare useful for mobile app developers?

**Motivation:** In RQ1, we perform a quantitative evaluation of the GLFE component in FeatCompare. The results show that our GLFE model (i.e., GLFE-25$^{\text{th}}$) can successfully identify high-level features discussed in user reviews. However, it remains unclear whether FeatCompare is able to compare features among competing apps and how the comparative table provided by FeatCompare can be utilized for developers. Thus in this RQ, we conduct a qualitative case study on the effectiveness of FeatCompare in comparing competing apps. Since our work aims to provide an approach that can help mobile app developers perform competitor analysis (i.e., analyze how users perceive an app concerning its competitors) and to better understand developers' practices and validate the usefulness of FeatCompare, we conduct a qualitative study with 107 participants.

**Approach:** We run FeatCompare on the five groups of competing apps which are randomly sampled from 20 identified app groups in RQ1, i.e., group "*SMS*", "*Weather*", "*Sports news*", "*Bible*", and "*Music player*". For each app group,

Table 6: Inferred top ten features with their representative words and an example review for the *"Weather"* group, the *"Sport news"* group, and the *"Bible"* group.

| | Feature | Representative words | Sample review |
|---|---|---|---|
| **Weather** | Detailed weather info | detailed, info, precise, concise, weather | *"Very nice app no problems love the minute by minute forecast"* |
| | Daily forecast feature | day, prepare, know, weather, clothes | *"Helps me plan my days ahead. Love it!"* |
| | Accurate forecast | accuracy, overcast, predict, percent, reality | *"Accurate on most days spot on"* |
| | App stability | sync, update, reload, recently, restart | *"This crappy app won't let you update or do things correctly"* |
| | User interaction | header, slider, banner, menu, animation | *"Less options of locations in widgets previous version was best"* |
| | In-app ads | pay, ads, commercial, subscription, dollar | *"Inappropriate advertising adds"* |
| | Device compatibility | tablet, phone, ipad, app, android | *"This app goes on every device that I have"* |
| | UI appearance | font, size, color, style, skin | *"New Color scheme makes the information hard to see/read"* |
| | Weather alert services | warning, flood, dangerous, alert, notify | *"Awesome it gives warnings 10 min earlier than the tv and radio do"* |
| | Location-aware services | location, enter, save, zip, address | *"Great app lots of info for multiple locations"* |
| **Sport news** | Video streaming | watch, behind, ahead, stream, buffer | *"Love that i can watch the hockey feeds but they are so far behind live"* |
| | Playoff coverage | season, playoff, basketball, nhl, cup | *"At least it works most of the time for the nhl playoffs this season"* |
| | Subscription service | service, paying, cable, subscription, satellite | *"Excellent way to get access to shows when paying high provider prices"* |
| | Chromecast | android, phone, chromecast, cast, device | *"Embarrassed that i downloaded this no chromecast having app"* |
| | Notification | alert, information, notification, push, daily | *"Stupid thing can t disable notifications wow this is bad"* |
| | Games playback | video, playback, stop, buffering, freez | *"Had great playback but now stutters and skips during live playback"* |
| | Bug reports | reinstall, uninstall, crash, start, delete | *"Won't load i submitted my carrier and still won't load waste of time"* |
| | Blackout broadcasting | watch, blacked, restriction, access, blackout | *"Too many blackouts"* |
| | User interaction | bar, screen, button, page, scroll, menu | *"Headlines don't take you to the story"* |
| | Version update | ruin, new, version, compare, worse | *"Become worst after updating"* |
| **Bible** | In-app ads | ads, shop, pay, flash, interruption | *"It is a great app but the ads are unnecessary they take up my entire screen"* |
| | Educative | learn, child, teach, help, interesting | *"Handy teaches u a lot of new things"* |
| | Highlight and bookmark | highlight, content, suggest, bookmark, search | *"Love the highlights bookmarks and how easy it is to get to a specific verse"* |
| | Guided prayer | lord, soul, pray, god, amen | *"Perfect one cannot ask for anything better the word of god really available"* |
| | Easy narration | simple, term, adequate, magnificent, keyword | *"There are no sub headings on electronic bibles compared to hard copies"* |
| | Audio feature | listen, multitasking, hear, play, pause | *"I like the audio reading while im doing other things"* |
| | Bible versions | kjv, niv, king, james, esv | *"The bible has many different versions i haden't even heard of"* |
| | Practicality | quick, convenience, useful, portable, ease | *"Easily accessible switches between translations quickly and easily"* |
| | App stability | issue, error, fix, stop, bug | *"After update app is stuck on start screen after the most recent update"* |
| | Sharing | post, facebook, friend, link, social | *"I can't share any of my daily scriptures to instagram or snapchat please fix"* |

Table 7: Inferred top ten features with their representative words and an example review for the *"SMS"* group, the *"Music Player"* group, the *"Cooking Recipe"* group and the *"Free Call"* group.

| | Feature | Representative words | Sample review |
|---|---|---|---|
| **SMS** | Scam identification | block, scam, spam, robot, anonymous | *"Its useful to avoid spam calls and to identify the fake numbers"* |
| | Location connectivity | address, city, state, network, operator | *"It does not give location"* |
| | App stability | close, reboot, restart, hang, crash | *"Very useful not always 100 updated or synced to my phone"* |
| | Customization | simple, customizable, design, interface, replace | *"Far better than i expected full of customizations"* |
| | User experience | section, content, folder, undo, add | *"Much easier to find the people in our contact list"* |
| | In-app ads | advertiser, interruption, profit, commercial, cost | *"I do not want your notifications of promoting your stuff"* |
| | Account authentication | sign, login, register, error, connect | *"Not working unable to connect"* |
| | UI appearance | style, wallpaper, emoji, ugly, front | *"More skin and background options as well as font options"* |
| | Security | hacking, dangerous, trust, cheater, steal | *"People are misusing it to create a fraud saving fraud number"* |
| | Premium version | subscribe, pay, lifetime, purchase, membership | *"After i upgraded to premium the call record function disappeared"* |
| **Music Player** | App stability | stop, close, randomly, crash, freeze, anonymous | *"Bugged wont let you open music player closes itself"* |
| | Music library | rapper, metallica, band, artist, song | *"App only shows your list and option to search youtube"* |
| | Download music | store, mp3, file, storage | *"Won't let me listen to music I've downloaded"* |
| | Complain | explain, understand, argument, respond, prefer | *"Needs instructions i have no idea how to get music on it"* |
| | User Interface | toolbar, layout, feature, section, icon, tab | *"Easy to navigate but white colors on notification bar barely visible"* |
| | Search | search, screen, find, select, change | *"It won't let me search"* |
| | Premium version | buy, trial, purchase, version, premium | *"Pro version still needs an ad free version"* |
| | Playlist | order, track, genre, playlist, album, artist | *"I love the way how playlist work keep it up"* |
| | Offline feature | internet, wifi, connection, offline, data | *"Wish i had wifi i really would give it five stars if i did not need wifi"* |
| | Sound quality | bass, speaker, headphone, sound, high | *"Works just fine needs more sound output"* |
| **Cooking Recipe** | Record keeping | track, maintain, manage, monitor, organize | *"Tracking is so easy with frequent foods favorites and build a recipe"* |
| | Online reliability | problem, error, internet, server, connection | *"Network error freezes all the time"* |
| | User experience | thumbnail, item, category, section, search | *"Easy to search and find foods"* |
| | App practicality | convenient, handy, easy, quickly, helpful | *"My Guardian So handy to keep me on track wherever I am"* |
| | Bar code scanner | store, product, code, shop, qr | *"Needs to be easier to open the bar code scanner"* |
| | Recipe diversity | recipe, ingredient, search, menu, choice | *"Endless selection of recipes you can think of and more"* |
| | Version update | version, update, long, recent, upgrade | *"After the update it's even worse"* |
| | Fitness tracker compatibility | fitness, sync, gear, fit, tracker | *"I love that it sync with my Fitbit"* |
| | Diet plan | journey, program, goal, loss, eat | *"Love this app so far it has been instrumental in my weight loss journey"* |
| | Authentication | sign, log, account, login, email | *"Won't let me log in and will not let me create another account"* |
| **Free Call** | Call quality | call, outgoing, audible, echo, voice | *"I can t hear the other side"* |
| | International calls | korea, japan, abroad, overseas, internationally | *"Very useful instant chat and video call with anyone internationally"* |
| | Privacy | hide, block, remove, privacy, unwanted | *"Privacy issue no option to hide last seen"* |
| | Bug reports | exit, shut, freeze, hang, close | *"Keeps on crashing keeps on crashing while calling"* |
| | Other | cool, fun, fast, nice, interesting | *"Useful app i really like this app but sometime very annoying"* |
| | User experience | ux, usability, designed, complex, unintuitive | *"It s easy to use than anything else simplicity is whats admirable"* |
| | User interface | toolbar, section, header, column, icon | *"Not happy with the new UI. Some may like it and it's fine for default"* |
| | Version update | update, yesterday, upgrade, re-download, patch | *"Always need update but not improvement"* |
| | Account authentication | admin, login, signup, registered, signin | *"It spammed my contact list with a link from my account dangerous app"* |
| | Credit services | token, dollar, diamond, lottery, coin | *"It is really useful especially when u have no credit"* |

FeatCompare identifies 28 high-level features and generates a comparative table, as described in Section 3. Among the 28 features, 14 are learned from the local cycle and another 14 from the global cycle. To demonstrate the usage of FeatCompare for app comparison, we consider two specific scenarios: 1) identifying the most commented features in a set of competing apps and 2) comparing feature-wise user opinions among competing apps. For the first scenario, we rank 28 identified features for each app group by calculating the number of reviews associated with each high-level feature within the group. For the second scenario, we explore insights that could be mined from the comparative tables created by FeatCompare.

For the qualitative survey, Table 9 shows the survey questions. In general, the survey consists of three parts:

1. **Background questions**: General questions asking about the participant's age, years of work experience in the mobile development field, job role, and the number of developed mobile apps.
2. **Development activities questions**: Development-related questions including whether the participant has conducted any comparative analysis before, the sources used to conduct competitor analysis, opinion on using the overall app store ratings as the only source to conduct comparative analysis.
3. **Validating FeatCompare**: Questions asking the opinion of the participant on the comparative table (i.e., Figure 3) created by FeatCompare.

To ensure that the participants of a specific organization do not bias our survey, we approach participants through multiple communication channels as follows.

– **Surveying developers of apps on F-Droid.** We retrieve a list of 922 open-source Android apps from F-Droid[6]. For each app, we obtain the developers' contact email addresses from the Google Play Store. Then, we send our survey to the 922 email addresses and receive 20 responses (2.2% response rate).
We also contact the maintainers and the contributors of the 922 F-Droid apps as follows. First, we identify 598 apps providing their corresponding Git repositories on F-droid. Next, we collect 10,146 developers and their email addresses from the git commit history of the identified 598 git repositories. We then rank all developers based on the number of commits they made in the repositories and select the top 3,000 developers as a target. These 3,000 developers contribute 2,135 valid email addresses. In this end, we send our survey to these 2,135 email addresses and obtain 40 responses (1.9% response rate).
In total, we obtain 60 responses from the developers of F-Droid apps.
– **Surveying participants through the popular apps at the Google Play Store.** We retrieve 2,000 contact email addresses from the Google

[6]https://f-droid.org/en/

Play Store pages of the 2,000 popular apps collected in RQ1. Then, we send our survey to these 2,000 email addresses and obtain only 5 responses (0.3%).

- **Surveying the development teams of multinational companies.** We send our survey to the technical leaders in five multinational companies and ask them to distribute the survey within their teams. In the end, We obtain 22 responses from the development teams of different multinational companies.
- **Surveying participants using the development chat platforms.** We post our survey to the most popular development chat platforms. In particular, we post our survey to reddit Android development groups[7,8,9], Facebook developer circle Beirut group[10], and Facebook mobile development pages and groups[11,12,13,14]. We obtain 20 responses from participants on development chat platforms.

In total, we survey 107 participants.

**Results: FeatCompare can spot the most frequently mentioned features in a group of competing apps.** Tables 6 and 7 show the top ten features with the highest number of reviews associated with five considered app groups. We only present the top-10 representative words and one sample review for each feature due to space limitations. The extracted features show the capability of FeatCompare in automatically finding the most popular (frequently mentioned in reviews) features among competitors. For example, using FeatCompare, we find that "*Scam identification*" is the most frequently discussed feature in the "*SMS*" group. We can also observe that providing detailed weather information is the most popular feature in the "*Weather*" group.

**FeatCompare can spot potential opportunities for improving the app.** Figure 3 presents the comparative table created by FeatCompare for the top three popular weather apps. For each considered app, the comparative table contains the 5-star rating distribution among the reviews associated with each feature and the total number of reviews mentioning the feature. This breakdown helps in identifying how the users perceive every feature of the app. For example, the feature "*Location-aware services*" is mentioned in a similar number of reviews in the *"Weather by WeatherBug"* (aka WeatherBug) app and the *"Weather radar and live maps - The Weather Channel"* (aka WeatherChannel) app. By comparing the star ratings of this feature in two apps,

[7] https://www.reddit.com/r/mAndroidDev/
[8] https://www.reddit.com/r/appdev/
[9] https://www.reddit.com/r/androiddev/
[10] https://www.facebook.com/groups/DevCBeirut
[11] https://www.facebook.com/groups/1549592438605145/
[12] https://www.facebook.com/groups/260880814006061/
[13] https://www.facebook.com/groups/cs464/
[14] https://www.facebook.com/groups/cs464/?ref=contextual_unjoined_mall_chaining


Fig. 3: Comparison of the top ten features of the three most popular apps of the "*Weather*" group.

we can tell that WeatherBug users are more satisfied with its location-aware services than the users of WeatherChannel. FeatCompare also provides the average feature rating per app group to support app comparison. For instance, from Figure 3, we can discover that regarding the location-aware services, WeatherBug significantly outperforms the average of all competing apps in the same group. We also can spot that the app "*AccuWeather: Live weather forecast & storm raider*" may need to improve its weather alert services as it receives a significantly lower rating on the weather alert services compared to the other apps.

**Competitor analysis is a common practice in mobile app development.** Figure 4 and Figure 5 show the job role and the experience of the surveyed participants. As shown in Figure 5, 76% of the participants have at least 2 years of mobile development experience, and 91% of them have been enrolled in development tasks. Note that participants are allowed to choose multiple job roles. As shown in Figure 6, 94% of the surveyed participants compare their apps to similar ones, while only 6% do not perform apps comparison.

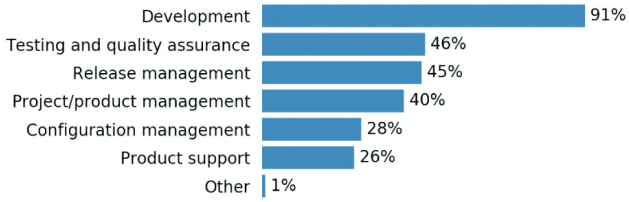


Fig. 4: The distribution of the job roles of the surveyed participants.

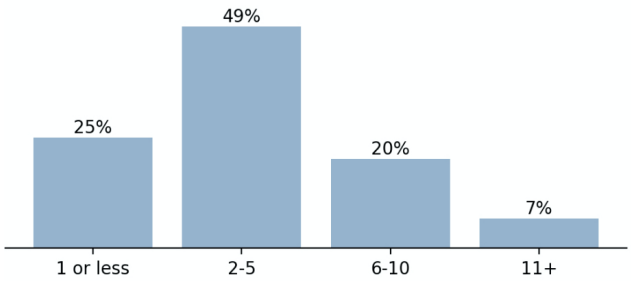


Fig. 5: The distribution of the years of experience of the surveyed participants.

We further investigate how participants conduct comparative analysis. Figure 7 shows that 68% of the participants treat user reviews as a comparison

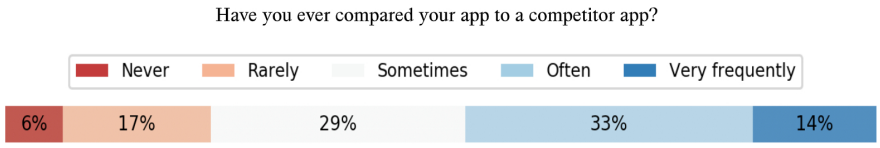


Fig. 6: Participants' answers about competitor analysis.

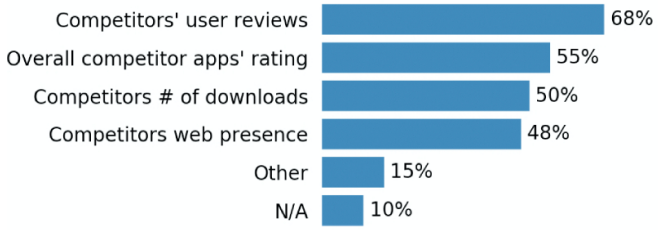


Fig. 7: Participants' answers about the source used for competitor analysis.

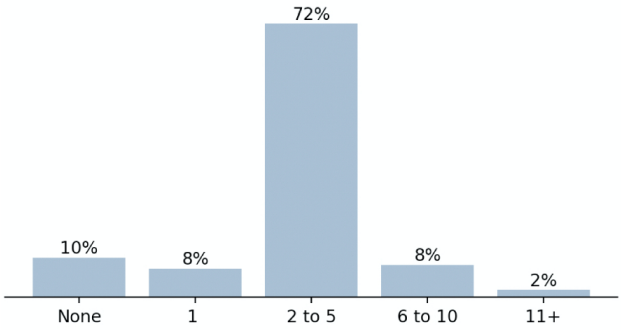


Fig. 8: Participants' answers about the number of competing apps.

source to rely on. We also observe that 82% of the participants compare their apps with at least two competing apps, as shown in Figure 8. These survey results imply that comparative analysis is common among app developers, and user reviews are valuable for performing competitor analysis. Hence, providing an automated approach for similar apps comparison based on user reviews can help app developers perform competitor analysis.

**The overall app rating alone is not sufficient for the competitor analysis of mobile apps.** As shown in Figure 9, only 10% of the participants believe that the overall app rating is ample alone to compare among similar apps, i.e., other sources are crucial for the competitor analysis of mobile apps. We also find among the 22% of the survey participants who express a neutral

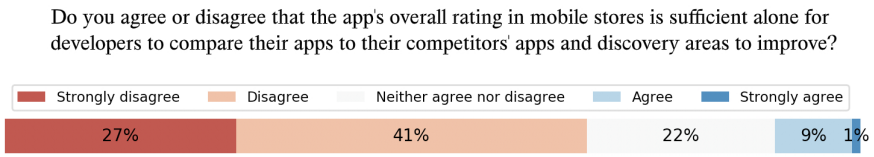


Fig. 9: Participants' opinion about overall app rating.

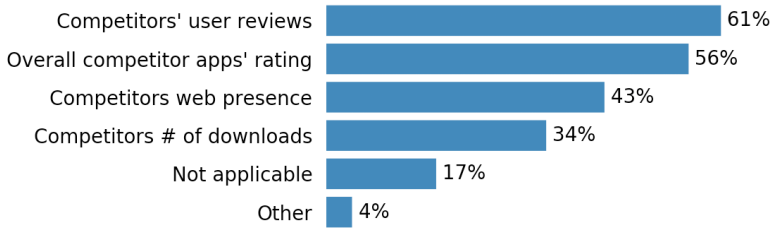


Fig. 10: Source of competitor analysis for participants having a neutral opinion about the overall general rating of an app being enough to compare similar apps.

opinion on the use of the overall app rating, 61% of them indicate that they use the reviews of similar apps in competitor analysis (Ref. Figure 10).

As shown in Figure 10, among the 22% of participants with neutral opinion, only 56% rely on the overall ratings of competitors' apps to conduct competitor analysis, while the percentage of the participants who depend on the competitors user reviews (61%) in the competitive analysis remains higher. Thus, the above results further emphasize the important role of user reviews in competitor analysis of mobile apps.

**Our approach is an asset to mobile developers to conduct a high-level feature analysis of competing apps.** We find that 73% of the participants agree that the comparative table created by FeatCompare will be of great benefit (54% agree + 19% strongly agree) for comparing their apps to competitors' apps, as shown in Figure 11. Then, we take a closer look at the statistics of the responses and find that the participants who appreciate FeatCompare belong to diverse backgrounds. 74% of the participants have a minimum of 2 years of work experience in the mobile field, as shown in Figure 12. Only 5% of them have never performed any competitor analysis before, whereas the rest 95% have, as shown in Figure 13. Figure 14 shows that 86% of the participants compare their apps to at least two competitors.

Our findings aligned with existing mobile development related surveys [2, 40] by demonstrating that competitor analysis is a point of interest to developers and that user feedback is the main source considered when performing the competitive analysis. The qualitative study by Nayebi et al. [40] gathers information about the market's impact on the mobile apps release planning

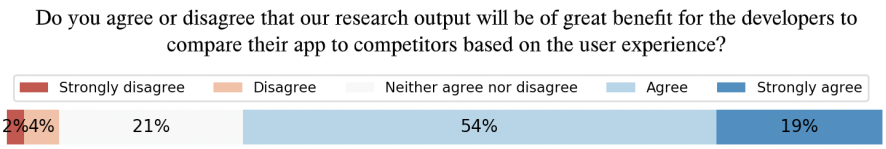


Fig. 11: Participants' opinion about our research work.

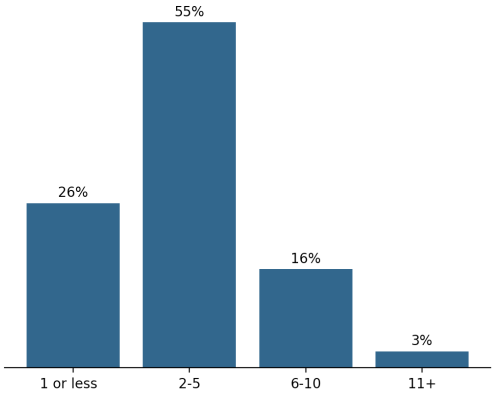


Fig. 12: Years of mobile experience of the participants who agree on the benefit of our work.

by interviewing 22 participants. In particular, 20 out of the 22 surveyed participants vote that the customer's feedback is the most important factor for evaluating the success or the failure of mobile apps. Also, A. Al-Subaihin et al. [2] survey 186 participants with mobile development background. The survey results reveal that more than half of the participants rely on competitor analysis to gather requirements. Specifically, 81% of the respondents that conduct competitor analysis select "user feedback" as the main point of interest when looking at similar apps. Similarly, for app enhancements, more than half of the developers examine similar apps in the app store, and 37% rely on similar apps users' feedback. In addition, our survey expanded the knowledge about practitioners' behaviors related to competitor analysis including opinions about the frequency of performing competitive analysis, ways of identifying competitors, and the number of competing apps considered by developers.

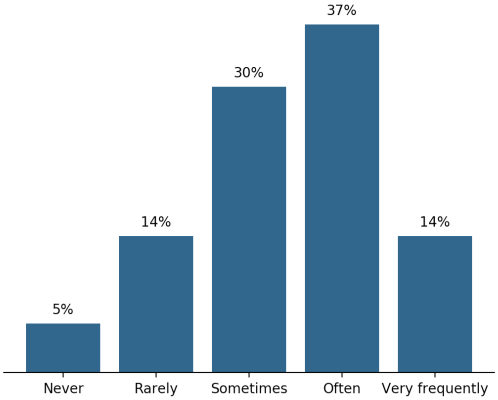


Fig. 13: The frequency of analyzing the competing apps of the participants who agree on the benefit of our work.

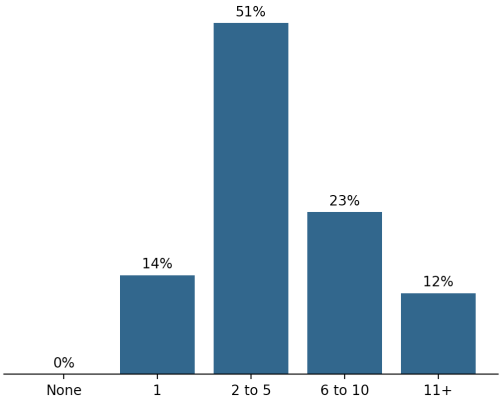


Fig. 14: The number of competing apps of the participants who agree on the benefit of our work.

**Summary of RQ 2**

FeatCompare can help app developers improve their apps by discovering the frequently commented features in competing apps, understanding an app's relative performance on each feature, and spotting the potential areas of improvement. The obtained 107 participants' responses show that competitor analysis is a common practice in mobile app development. 68% of the surveyed participants affirm that the overall app store ratings are not sufficient to conduct a competitor analysis of mobile apps. The survey results show that FeatCompare is an asset to mobile developers and supports high-level feature comparison among competing apps in an automated manner with minimal human effort.

## 6 Discussion

To validate the performance of FeatCompare in extracting high-level features, we implement a Vector Space Model (VSM) baseline to compare against it. For the Vector Space Model (VSM) baseline, the features of each app group are extracted in two steps. First, we create the vector representation of each review from an app belonging to the group using the vector space model (VSM). Specifically, each word appearing in the review becomes one feature, and the weight of the feature is determined by the Tf-Idf (term frequency–inverse document frequency) scheme using TfidfVectorizer function [15]) in the sklearn library to convert the collection of reviews to a matrix of TF-IDF features. Second, we cluster the review vectors using the k-means unsupervised clustering algorithm. The number of clusters, i.e, k, is set to 14, the same as the one used in FeatCompare.

We notice that VSM model produces many non-coherent clusters of reviews, i.e., it is challenging to identify a high-level feature representing the whole cluster. In order to fairly evaluate the coherence of the clusters created by the VSM-based approach, we design an evaluation procedure following Chen et al. [7]. First, for each of the review clusters generated by VSM-based approach, we rank the words appearing in the reviews based on their frequency and select the top-50 words as the representatives of the review cluster (i.e., "high-level feature" identified by the approach). Next, we collect the top-50 representative words for each of the top-10 high-level features identified by FeatCompare and the VSM-based approach respectively. The 20 word groups are then mixed and passed to three judges, who have more than six years' working experiences in IT companies and have developed mobile apps before. We ask judges to rate each of the 20 groups using a 5 point rating scale (i.e., "Strongly agree", "Agree", "Neither agree nor disagree", "Disagree", and "Strongly disagree" ), specifying if they consider most of the words in the group representing a high-level feature of a mobile app. After collecting ratings from three judges on six app groups, i.e., 120 word groups, we calculate the number of coherent word groups identified by FeatCompare and the VSM-based approach. We consider a word group being coherent if at least two judges "Strongly agree" or "Agree" that most words in the group represent a high-level app feature. table 8 summarizes the evaluation results, i.e., FeatCompare can identify much more coherent word groups representing high-level features than the VSM model. As the baseline approach can not identify high-level features from user reviews, we do consider it in RQ1.

## 7 Threats to Validity

**External Threats:** External threats are concerned with our ability to generalize our results. In our study, we analyze the top 196 popular apps that belong

[15] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html/

Table 8: Number of coherent aspects. K (number of aspects) = 10 for all approaches.

| App Group | Coherent aspects | |
|---|---|---|
| | VSM | FeatCompare |
| Weather | 3 | 8 |
| Sports news | 2 | 8 |
| Bible | 2 | 7 |
| SMS | 5 | 10 |
| Music player | 3 | 8 |
| Recipe cooking | 4 | 7 |

to 20 different app groups. The total number of mobile apps in the Google Play Store reached 2.57 million apps by the fourth quarter of 2019 [11]. Hence, our results can be limited to the studied apps. To eliminate the impact of the app selection process on our results, we chose 196 apps distributed across 17 different categories at the Google Play Store. Although the Google Play Store contains more than 17 apps categories, we believe that our apps cover a wide variety of apps in the store. Moreover, our proposed approach can be easily extended to more categories as it does not require manually annotated resources. Similarly, we have only targeted apps in the Google Play Store. However, our approach can be applied to different apps in other stores (e.g., the Apple App Store and the Amazon App Store) as long as the reviews can be extracted.

It is relevant to note that the similarity between apps was defined by identifying keywords reflecting the main functionality across closely related apps. The intuition behind this selection method is that mobile apps that are similar in their descriptions or titles would behave similarly and present common functionality. Therefore, two of the authors chose the groups keywords in a manner to represent as much diversity in the functionalities and business domains as possible. Unlike other related work [28] that only covered fewer than ten distinct app groups, our study includes 20 groups. Moreover, since our method of identifying competing app groups might be biased as in practice, developers may have specific criteria for selecting their competitors, we have addressed this concern by conducting a manual verification. The first two authors independently validated that each app of a group lists the rest of other apps belonging to the same group under the *"similar"* app list recommended by the Google Play Store.

**Internal Threats:** One threat to internal validity is mainly concerned with the manual assignment of features. We manually analyzed the automatically assigned features to user reviews on a statistically representative sample with a confidence level of 95% and a confidence interval of 10% (i.e., 96 user reviews per every group). To mitigate the error of manual identification of features, the first three authors of this paper independently identify features form the user reviews. The Fleiss's Kappa agreement score on labeled reviews is 0.86, which shows a strong agreement between the annotators. However, we are not the

owners of the studied apps, thus our analysis can be limited by our knowledge about the studied apps.

**Construct Threats:** User reviews may contain multiple features. Our approach assigns a single high-level feature per review. The assumption that every review represents one feature only can impact the accuracy of the feature extraction approach and the average feature ratings reported for each app. In ection 5.2, we generate the features level average rating per app for the weather group. Although we find that the average percentage of multi-label reviews across the 672 manually labeled reviews is 8.6%, the results in RQ2 may be impacted by the fact that a single feature is extracted from every review. To mitigate this problem and validate whether the features' average rating will be significantly impacted if multi-labelling was assigned to reviews, we conduct an experiment to generate the results of RQ2 taking into consideration multi-labelling. In the manually labeled statistical sample of user reviews of the weather apps, we find that 9% of the reviews are multi-label. We followed the same approach as in section 5.2 to calculate the average feature rating, except that we assign multiple labels per review before calculating the number of reviews associated with each high-level feature. First, we randomly assigned an additional label (different from the label already assigned by FeatCompare to the review) to 9% of the user reviews of the weather apps. Second, we calculate the number of reviews associated with each high-level feature within the group. Finally, we generate the features' average rating. We compare the overall rating of the top-10 features in the cases of single label assignment and multi-label assignment. We observe an average of 0.03 rating difference on the top-10 high-level features extracted from 10 apps in the weather group. We perform the Wilcoxon signed-rank test [30] using the wilcox.test function in R [16]. We observed a p-value of 0.3459, indicating no significant difference between the average rating on high-level features before and after considering multi-label reviews.

## 8 Related Work

In this section, we first introduce the existing mobile app review analysis approaches for competing apps. We then briefly mention approaches for extracting fine-grained features from app reviews and other studies on mining mobile app reviews.

### 8.1 Mining User Reviews for Mobile App Comparison

While automated tools are increasingly being proposed to analyze the reviews of a specific mobile app [15, 16, 20, 23, 52], few researchers have centered their focus on extracting useful knowledge from reviews of competing

[16] https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/wilcox.test

apps [8, 28, 50, 53]. For example, Shah et al. [53] propose the task of mining mobile app reviews for competitor analysis as a tool named REVSUM [53]. REVSUM takes as input reviews from a set of competing apps and compares users' sentiment on each fine-grained feature among competing apps. First, REVSUM identifies reviews that contain feature evaluation, bug reports, or feature requests. Next, it extracts fine-grained features from selected reviews using an approach named SAFE [23]. In the end, REVSUM applies the sentiment score prediction function offered in the Stanford CoreNLP library on sentences that mention fine-grained feature(s) reviews and compares the average sentiment score of each feature across competing apps. Shah et al. did not evaluate the accuracy of the sentiment score prediction component and the fine-grained feature extraction component in REVSUM. Thus, it is unclear how REVSUM performs in practice.

Dalpiaz and Parente design a similar tool named RE-SWOT that can extract fine-grained feature requirements from user reviews of competing apps and generate a Strength-Weakness-Opportunity-Threat (SWOT) matrix supporting competitor analysis [8]. RE-SWOT takes as input all reviews of competing apps and identifies fine-grained features by finding word pairs that co-occur frequently in reviews. Several hand-craft rules are applied to filter out meaningless co-occur word pairs. To further reduce the number of fine-grained features, RE-SWOT merges semantically similar features by invoking a closed NLP service[17]. After grouping similar fine-grained features, each review is then assigned to identified features based on words appearing in the review. Different from REVSUM, RE-SWOT does not apply any sentiment analysis tool on reviews. It aggregates ratings of reviews associated with each feature and creates a SWOT table for each app based on the average rating per feature compared to the average rating per competing group. RE-SWOT suffers from three main issues: 1) the identified fine-grained features have not been evaluated; 2) only a small-scale interview was conducted to validate the usefulness of the generated SWOT tables; 3) it does not filter out non-informative reviews.

Different from RE-SWOT and REVSUM, which first identify fine-grained features from user reviews and then summarize users' opinions on each feature among competing apps, Li et al. [28] propose a tool that can compare features of competing apps via identifying comparative reviews. Comparative reviews are reviews such as *"Slower page loading than chrome"* for Firefox mobile app, which directly compares Firefox with Chrome. Their approach is good at identifying explicit app comparison provided by users. However, it fails to compare other features that are not mentioned in user reviews in a comparative fashion. Besides, there might be a limited number of comparative reviews available among a target set of competing apps.

Shah et al. [8] introduce an approach that compares two apps' features pairwise. The authors use a dataset of 25 apps among which many apps belong to the same AppStore category. First, the authors extract fine-grained features by relying on the two words collocations in the user reviews of all

[17] https://www.cortical.io/

the apps. Second, the authors compute the average sentiment score (positive and negative) for a feature using SentiStrength tool [18]. After the developer chooses a base app, the tool identifies the list of competing apps based on the common fine-grained features shared among the set of 25 apps and selected by the developer. The competitor analysis is conducted by comparing the sentiments of the features of the base app and the selected competitor. While the approach proposed by Shah et al. [8] supports only competitive analysis of two competing apps, while FeatCompare supports multiple competing apps comparison.

Our paper aims to reduce the limitations of the above work and complement them by proposing FeatCompare, an approach that mines user opinions on high-level features among competing apps automatically. FeatCompare filters out non-informative reviews and utilizes not only reviews from competing apps of a specific group, but also reviews of apps across multiple groups. Moreover, besides a qualitative case study on the resultant comparative tables, we perform a quantitative evaluation of the feature extraction component in FeatCompare on 480 annotated reviews. To validate the idea of FeatCompare, we also conduct a survey with more than 100 mobile app developers.

### 8.2 Extracting Features from Mobile App Reviews

Extracting features from user reviews for a specific app is a trending and fundamental task in app store mining research [15, 16, 21, 23, 52]. Identified features could be used to summarize user opinions on app features and thus provide actionable insights for developers to improve their apps. Depending on how to identify words describing app features in reviews, these approaches can be categorized into two groups, rule-based approaches, and collocation-based approaches.

Rule-based approaches such as MARA [20, 21] and SAFE [23] mine fine-grained features based on manual defined linguistic rules. However, these linguistic rules are often created by investigating a limited number of reviews. Thus, they suffer from a potential loss of features due to the bias in sampled reviews. A recent study [52] shows that the most advanced rule-based feature extraction approach SAFE is sensitive to the density of the annotated app reviews in a review dataset and may lead to poor performance in practice.

To catch more fine-grained features and reduce manual work in rule-based approaches, researchers have proposed another line of tools that can automatically identify fine-grained features from reviews without linguistic patterns [15, 16]. These approaches are collocation-based, i.e., mining word pairs such as *"picture view"* that co-occur unusually often in reviews. Guzman and Maalej propose the first collocation-based feature extraction algorithm [16]. Their approach removes words that are not nouns, verbs, or adjectives from reviews. Next, they calculate the co-occurrence of all word pairs in the preprocessed reviews and treat those that appear in at least three reviews and

[18] http://sentistrength.wlv.ac.uk

that have less than three words distance between them as fine-grained features. A sentiment analysis tool is then applied to each sentence that contains at least one fine-grained feature, and sentiment scores of sentences are aggregated to indicate user opinions on each feature. Gu and Kim propose an approach named SUR-Mine, which aims to answer "what parts are loved by users" for app developers [15]. Unlike previous work, SUR-Mine extracts feature and opinion word pairs such as *"prediction, accuracy"* together and then uses co-occur frequency as the criteria for identifying fine-grained features and their associated sentiment.

Similar to the above approaches, FeatCompare mines features from user reviews. Different from them, FeatCompare targets high-level features and aims for comparing features shared cross competing apps. FeatCompare does not require hand-craft linguistic patterns. Instead, it utilizes an unsupervised neural model to automatically identify the high-level feature that is most semantically relevant to each review. word2vec and GloVe[44] are among the widely used algorithms to convert words into vectors. In GLFE, we follow ABAE and use word2vec to initialize word embeddings. Existing studies [26, 46] in the NLP domain show that word2vec outperforms GloVe in various tasks. Rezaeinia et al.[46] compare word2vec to GloVe using four different benchmark sentiment datasets, including Amazon product reviews and the Stanford sentiment treebank. Their experimental results of the sentiment classification task show that word2vec performs better than GloVe in sentiment analysis for all datasets in terms of accuracy and F-score. Moreover, Levy et al. [26] demonstrate that word2vec outperforms GloVe in several linguistic tasks (e.g., word similarity and analogy detection) applied on eight datasets. The authors also show that word2vec is computationally efficient; it is faster to train than GloVe and requires less disk space and memory.

### 8.3 Other Studies on Mining Mobile App Reviews

**Summarizing reviews:** Fu et al. [12] assume that negative reviews (associated with 1-star or 2-star ratings) are most interesting to developers. They apply Latent Dirichlet Allocation [4] (LDA) on negative reviews and identify the major reasons why users dislike an app and learn how users' complaints changed over time. Vu et al. [57] believe that a set of keywords could capture developers' interest. Thus, they propose a framework that takes input a set of keywords from developers and then ranks all reviews based on their relevance to the specified interest and group most relevant reviews to summarize user opinions.

**Categorizing reviews:** There are plenty of taxonomy methods and corresponding classifiers proposed to automatically categorize reviews based on user intent or software engineering topics. Hassan et al. [18] propose a way to identify reviews that are likely to get response by app developers. So store owners can spot such reviews for app developers. Panichella et al. [43] use natural language processing (NLP) and sentiment analysis techniques to automatically

classify user reviews into four types: information seeking, information giving, feature request, and problem discovery. Villarroel et al. [55] propose a tool named CLAP that mines app reviews for the release planning of mobile apps. CLAP features a supervised learning algorithm that can categorize reviews into three categories: bug report, the suggestion for a new feature, and others. Consequently, CLAP clusters similar reviews and provides the developers with suggestions for future releases. Mcilroy et al. [36] introduce a fine-grained categorization of reviews. They identified 14 types of issues in reviews and found that up to 30% of the reviews raise various types of issues in a single review. Hassan et al. [17] study the frequency of each issue type in triggering bad updates (i.e., updates with high percentages of negative reviews or user complaints). SURF [9] considers a combined categorization of user intent and SE topics. Man et al. [33] define seven types of issues appearing in reviews cross multiple app stores and propose a corresponding review classifier. Lu et al. [31] propose a review classifier that can categorize reviews into three main types, including non-functional requests (related to reliability, usability, portability, and performance), functional feature requirements, and others.

**Prioritizing and filtering non-informative reviews:** Keertipati et al. [24] rank feature requests extracted from reviews based on four feature attributes, including frequency, rating, negative emotions, and deontics. Recently, Gao et al. [13] propose a tool named IDEA to identify emerging issues from user reviews. They define the emerging issue as an issue in a time slice that rarely appears in the previous slice but is mentioned by a significant proportion of user reviews in the current slice.

Despite the potential usage of user reviews for improving mobile apps, many user reviews contain less-valuable information, such as pure user emotional expression, questions, etc. To solve this issue, Chen et al. [6] proposed an app review analyzing framework named AR-Miner. AR-Miner first identifies non-informative reviews by training a semi-supervised algorithm on a small scale of labeled reviews along with a mass of unlabeled reviews. Next, it groups the informative reviews using LDA, and further prioritizes the informative reviews by an effective review ranking scheme. FeatCompare adopts the review filtering component AR-Miner in the data preprocessing step to filter non-informative sentences from reviews.

## 9 Conclusion

With the tremendous number of the daily submitted user reviews, the manual analysis of user reviews becomes an impractical task. Since app reviews enclose valuable information for app developers, including users' opinions about the apps feature, an automated approach to extracting them from the feedback is essential to conduct app competitor analysis.

In this paper, we introduce FeatCompare, a novel approach that helps developers perform competitor analysis on high-level features based on user reviews with the minimal human intervention. FeatCompare contains three

main components: 1) a component to preprocess raw reviews and filter non-informative user reviews; 2) GLFE, an unsupervised model that can identify global and local high-level features from app reviews and select the final feature for each review via a thresholding mechanism, and 3) an aggregator that generates a table of comparison among apps for summarizing feature-level user opinions.

We apply FeatCompare on ten million user reviews of 196 popular apps on Google play belonging to 20 different functional groups. A quantitative evaluation of GLFE is conducted for five randomly picked groups of competing apps. The evaluation results show that GLFE outperforms the state-of-the-art high-level feature extraction model ABAE by 14.7% on average. We also survey 107 mobile app developers asking how they perform competitor analysis in practice and how they perceive the comparative table created by FeatCompare. 73% of the mobile app developers participants endorse the benefits of FeatCompare in competitor analysis. We aim in the future to include more mobile app categories in our study. Furthermore, we aim to expand our qualitative study to investigate a larger number of mobile app developers. Lastly, we will expand our work and look at how high-level feature ratings evolve over releases among competing mobile apps.

## References

1. Akdeniz. Google Play Crawler. https://github.com/Akdeniz/google-play-crawler, 2013. (Last accessed: March 2020).
2. A. AlSubaihin, F. Sarro, S. Black, L. Capra, and M. Harman. App store effects on software engineering practices. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.
3. AppAnnie. App Annie. https://www.appannie.com/, 2016. (Last accessed March 2020).
4. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
5. L. V. G. Carreño and K. Winbladh. Analysis of user comments: An approach for software requirements evolution. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 582–591, 2013.
6. N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. AR-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE '14, pages 767–778, 2014.
7. Z. Chen, A. Mukherjee, and B. Liu. Aspect extraction with automated prior knowledge learning. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 06 2014.
8. F. Dalpiaz and M. Parente. RE-SWOT: from user feedback to requirements via competitor analysis. In *Proceedings of the 25th International*

*Working Conference on Requirements Engineering: Foundation for Software Quality*, volume 11412 of *REFSQ '19*, pages 55–70, 2019.
9. A. Di Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. Canfora. SURF: summarizer of user reviews feedback. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, pages 55–58. IEEE, 2017.
10. O. El Zarif, D. A. da Costa, S. Hassan, and Y. Zou. On the relationship between user churn and software issues. In S. Kim, G. Gousios, S. Nadi, and J. Hejderup, editors, *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*, pages 339–349. ACM, 2020.
11. eMarketer. Number of apps available in leading app stores as of 4th quarter 2019. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2020. (Last accessed March 2020).
12. B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1276–1284, 2013.
13. C. Gao, J. Zeng, M. R. Lyu, and I. King. Online app review analysis for identifying emerging issues. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 48–58, 2018.
14. A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 1025–1035, New York, NY, USA, 2014. Association for Computing Machinery.
15. X. Gu and S. Kim. "What parts of your apps are loved by users?" (T). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE '15, pages 760–770, 2015.
16. E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Proceedings of the 22nd International Requirements Engineering Conference*, RE '14, pages 153–162, 2014.
17. S. Hassan, C. Bezemer, and A. E. Hassan. Studying bad updates of top free-to-download apps in the Google Play Store. *IEEE Transactions on Software Engineering*, 46(7):773–793, 2020.
18. S. Hassan, C. Tantithamthavorn, C. Bezemer, and A. E. Hassan. Studying the dialogue between users and developers of free apps in the Google Play Store. *Empirical Software Engineering*, 23(3):1275–1312, 2018.
19. R. He, W. S. Lee, H. T. Ng, and D. Dahlmeier. An unsupervised neural attention model for aspect extraction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL '17, pages 388–397, 2017.
20. C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 41–44, 2013.

21. C. Iacob, R. Harrison, and S. Faily. Online reviews as first class artifacts in mobile app development. In *Proceedings of the 5th International Conference on Mobile Computing, Applications, and Services*, MobiCASE '13, pages 47–53, 2013.
22. D. P. K. JLB. Adam: A method for stochastic optimization. In *3rd international conference for learning representations, San Diego*, 2015.
23. T. Johann, C. Stanik, A. M. A. B., and W. Maalej. SAFE: A simple approach for feature extraction from app descriptions and app reviews. In *Proceedings of the 25th International Requirements Engineering Conference*, RE '17, pages 21–30, 2017.
24. S. Keertipati, B. T. R. Savarimuthu, and S. A. Licorish. Approaches for prioritizing feature improvements extracted from app reviews. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, EASE '16, pages 1–6, 2016.
25. S.-M. Kim, P. Pantel, T. Chklovski, and M. Pennacchiotti. Automatically assessing review helpfulness. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, page 423–430, 2006.
26. O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.
27. X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li. Unsupervised deep bug report summarization. In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, page 144–155, New York, NY, USA, 2018. Association for Computing Machinery.
28. Y. Li, B. Jia, Y. Guo, and X. Chen. Mining user reviews for mobile app comparisons. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):75:1–75:15, Sept. 2017.
29. S. L. Lim, P. J. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden. Investigating country differences in mobile app user behavior and challenges for software engineering. *IEEE Transactions on Software Engineering*, 41(1):40–64, 2015.
30. M. Lovric, editor. *International Encyclopedia of Statistical Science*. Springer, 2011.
31. M. Lu and P. Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17, page 344–353, 2017.
32. S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun. Active semi-supervised approach for checking app behavior against its description. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 2, pages 179–184, 2015.
33. Y. Man, C. Gao, M. R. Lyu, and J. Jiang. Experience report: Understanding cross-platform app issues from user reviews. In *Proceedings of the 27th IEEE International Symposium on Software Reliability Engineering*, ISSRE'16, pages 138–149, 2016.

34. P. Martin. 77% will not download a retail app rated lower than 3 stars. https://blog.testmunk.com/77-will-not-download-a-retail-app-rated-lower-than-3-stars/. (Last accessed: July 2017).
35. M. McHugh. Interrater reliability: The kappa statistic. *Biochemia medica : časopis Hrvatskoga društva medicinskih biokemičara / HDMB*, 22:276–82, 10 2012.
36. S. McIlroy, N. Ali, H. Khalid, and A. E. Hassan. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering*, 21(3):1067–1106, 2016.
37. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *Proceedings of the 1st International Conference on Learning Representations*, ICLR'13, pages 1–12, 2013.
38. A. Mukherjee and B. Liu. Aspect extraction through semi-supervised modeling. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 339–348, 2012.
39. M. Nayebi, B. Adams, and G. Ruhe. Release practices for mobile apps – what do users and developers think? In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 552–562, 2016.
40. M. Nayebi, H. Farahi, and G. Ruhe. Which version should be released to app store? In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 324–333, 2017.
41. E. Noei, D. A. da Costa, and Y. Zou. Winning the app production rally. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '18, pages 283–294, 2018.
42. D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 125–134, 2013.
43. S. Panichella, A. D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, ICSME '15, pages 281–290, 2015.
44. J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.
45. J. Ramos. Using TF-IDF to determine word relevance in document queries. In *Proceedings of the 1st instructional Conference on Machine Learning*, iCML '03, pages 1–4, 2003.
46. S. M. Rezaeinia, R. Rahmani, A. Ghodsi, and H. Veisi. Sentiment analysis based on improved pre-trained word embeddings. *Expert Systems with Applications*, 117:139 – 147, 2019.
47. J. Saldaña. *The coding manual for qualitative researchers*. Sage, 2015.

48. S. Scalabrino, G. Bavota, B. Russo, M. D. Penta, and R. Oliveto. Listening to the crowd for the release planning of mobile apps. *IEEE Transactions on Software Engineering*, 45(1):68–86, 2019.
49. C. B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999.
50. F. A. Shah, Y. Sabanin, and D. Pfahl. Feature-based evaluation of competing apps. In *Proceedings of the ACM International Workshop on App Market Analytics*, WAMA '16, pages 15–21, 2016.
51. F. A. Shah, K. Sirts, and D. Pfahl. The impact of annotation guidelines and annotated data on extracting app features from app reviews. *CoRR*, abs/1810.05187, 2018.
52. F. A. Shah, K. Sirts, and D. Pfahl. Is the SAFE approach too simple for app feature extraction? A replication study. In *Proceedings of the 25th International Working Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ 19, pages 21–36, 2019.
53. F. A. Shah, K. Sirts, and D. Pfahl. Using app reviews for competitive analysis: tool support. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, WAMA '19, pages 40–46, 2019.
54. R. Vasa, L. Hoon, K. Mouzakis, and A. Noguchi. A preliminary analysis of mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12, pages 241–244, 2012.
55. L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. D. Penta. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 14–24, 2016.
56. P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.
57. P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach (T). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE '15, pages 749–759, 2015.
58. X. Zhao, J. Jiang, H. Yan, and X. Li. Jointly modeling aspects and opinions with a MaxEnt-LDA hybrid. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. ACL, 2010.

## The survey questions

Table 9: List of defined questions in the conducted survey.

| ID | Question | The possible answers |
|---|---|---|
| **Background questions:** | | |
| Q1.1 | What is your age? | "<= 25 years old", "26-35 years old", "36-45 years old", "46+ years old", and "Prefer not to answer" |
| Q1.2 | How many years of work experience do you have? | "1 or less", "2-5 years", "6-10 years", "11-20 years", and "21+ years" |
| Q1.3 | How many years of work experience in the field of mobile development do you have? | "1 or less", "2-5 years", "6-10 years", and "11+ years" |
| Q1.4 | What are your roles in the development of mobile apps? Please select all that apply. | "Development", "Testing and quality assurance", "Release management", "Configuration management", "Product support", "Project/product management", "Other" |
| Q1.5 | How many mobile apps have you developed (including the current one) | "None", "1", "2-5", "6-10", and "11+" |
| Q1.6 | Which of the following industries best describe the category of your app? Please select all that apply. | "Weather", "Bible", "Browser", "Navigation", "Free Call", "SMS", "Music player", "News", "Security", "Wallpaper", "Taxi and rideshare", "Dating", "Recipe cooking", "Coloring", "Pregnancy", "Sports news", "Video editor", "Notes", "Mobile banking apps", "Accommodation booking", "Other" |
| **Development activities questions:** | | |
| Q2.1 | Do you agree or disagree with the following statement: app's overall rating in mobile stores is sufficient alone for developers to compare their apps to their competitors' apps and discovery areas to improve. | "Strongly agree", "Agree", "Neither agree nor disagree", "Disagree", and "Strongly disagree" |
| Q2.2 | Have you ever compared your app to a competitor app? | "Very frequently", "Often", "Sometimes", "Rarely", and "Never" |
| Q2.3 | How do you identify your app competitors? Please select all that apply. | "Keyword search", "App stores similar app suggestion", "App stores categories", "Customer feedback", "Social media", "Other" |
| Q2.4 | In case you do perform competitor analysis, how do you perform it? Please select all that apply. | "User reviews of the competitive apps", "Check the overall competitive app ratings", "Check the web presence of the competitors", "Check the competitor app number of downloads", "Other" |

| ID | Question | The possible answers |
|---|---|---|
| **Development activities questions:** | | |
| Q2.5 | If you have not conducted a competitor analysis before, what are in your opinion good sources of comparison? | |
| Q2.6 | How many competitors do you compare your app features to? (If you have developed multiple apps, please select a range that represents the average number of competitors that you used to compare with your app) | "None", "1", "2-5", "6-10", "11+" |
| **Validating FeatCompare:** | | |
| Q3.1 | Do you agree or disagree with the following statement: our research output will be of great benefit for the developers to compare their app to competitors' apps based on the user experience | "Strongly agree", "Agree", "Neither agree nor disagree", "Disagree", and "Strongly disagree" |
| Q3.2 | In the space below, please provide any feedback that you wish to share with us. For example, do you have any recommendations to improve the results of our comparison tool? We would really appreciate your input and feedback. | |