Domain-Specific Analysis of Mobile App Reviews Using Keyword-Assisted Topic Models

Anonymous Author(s)

ABSTRACT

Mobile application (app) reviews contain valuable information for app developers. A plethora of supervised and unsupervised techniques have been proposed in the literature to synthesize useful user feedback from app reviews. However, traditional supervised classification algorithms require extensive manual effort to label ground truth data, while unsupervised text mining techniques, such as topic models, often produce suboptimal results due to the sparsity of useful information in the reviews. To overcome these limitations, in this paper, we propose a fully automatic and unsupervised approach for extracting useful information from mobile app reviews. The proposed approach is based on keyATM, a keyword-assisted approach for topic modeling. keyATM overcomes the problem of data sparsity by using seeding keywords extracted directly from the review corpus. These keywords are then used to generate meaningful domain-specific topics. Our approach is evaluated over two datasets of mobile app reviews sampled from the domains of Investing and Food Delivery apps. The results show that our approach significantly outperforms traditional topic modeling techniques by producing more coherent topics.

ACM Reference Format:
Anonymous Author(s). 2021. Domain-Specific Analysis of Mobile App Reviews Using Keyword-Assisted Topic Models. In *Proceedings of The 44th International Conference on Software Engineering (ICSE 2022).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The explosive growth and widespread of mobile technology in the past decade has changed the way software is produced and consumed. More users now rely on mobile software than ever before. According to App Annie - the mobile market data and analytics platform, an average user spends around 4.2 hours a day using apps [41]. In response to this unprecedented demand, mobile app marketplaces, such as Google Play and the Apple App Store has grown dramatically in size, offering users virtually unlimited choices of apps. For instance, as of 2020, more than four million apps were available to download on the Apple App Store alone [72].

Popular app stores enable users to share their experience with app developers via ratings and textual reviews. This unique channel of user feedback created an opportunity for app developers to monitor their end users' reactions to the different releases of their app. Recently, analyzing mobile app reviews has attracted a considerable attention from the research community [21]. Researchers have utilized supervised and unsupervised machine learning algorithms to extract informative feedback from user reviews, including feature requests and bug reports as well as user goals and their rationale [14, 24, 42, 45, 51, 62, 71].

In general, review mining techniques achieve adequate levels of accuracy, however, they suffer from several limitations. For instance, supervised classification techniques rely on the presence of ground-truth datasets which typically require significant manual effort to generate [24, 42, 62, 70]. Furthermore, these techniques are constrained to a single rubric of predefined categories and, as a result, require additional data and model tweaking to generalize over domain-specific feedback [82]. For example, users of the Ridesharing app Uber might complain about wait times and rates, while users of the Investing app Robinhood might raise concerns about the app's requests for their social security or bank information. These categories of user feedback can be easily missed in the ground truth data. Consequently, a *one-size-fits-all* approach may not be suitable for domain-specific user feedback [21, 67, 82].

To avoid the drawbacks of supervised techniques, unsupervised topic modeling techniques, such as Latent Dirichlet Allocation (LDA) [9], have been applied to extract useful information from app store reviews [22, 25, 26, 31, 59, 64]. However, LDA does not perform well when dealing with small and unstructured text [6, 28, 83]. Short text artifacts, such as user reviews [79], do not typically contain enough information for statistical *bag-of-words* models to build semantic connections between words [1]. Therefore, generated topics can be hard to interpret and rationalize and often require an extensive calibration of hyperparameters to avoid misclassification [12, 28, 85].

To overcome these limitations, in this paper we propose a new approach for extracting useful user feedback from app store reviews. The proposed approach is based on the keyword-assisted topic model keyATM [33]. keyATM relies on a set of representative *seed* words to model the topics of a large document collection by finding evidence on the underrepresented topics. These seeding words can be extracted from the document corpus automatically by applying automated text summarization techniques. Our proposed approach is evaluated using two datasets of user reviews sampled from the domains of Investing and Food Delivery apps. The quality of generated topics is assessed using a set of intrinsic and extrinsic measures of topic coherence [8].

The rest of this paper is organized as follows. Section 2 formally describes LDA and its extension, keyATM. Section 3 introduces our approach. Section 4 evaluates our approach over two datasets of mobile app reviews. Section 5 discusses our main findings and their potential implications. Section 6 describes related work. Section 7 addresses the limitations of our study. Finally, Section 8 concludes our paper and discusses future work.

ICSE 2022, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND

Topic models are statistical techniques that are commonly used for discovering latent topics in text collections. In topic modeling, a topic can be described as a collection of words which represent a thematic concept in a corpus, and documents in the corpus are represented as probabilistic distributions over these topics. In what follows, we introduce the most commonly used topic modeling approach, LDA, and its extension - the keyword-assisted topic model (keyATM).

2.1 Latent Dirichlet Allocation (LDA)

Introduced by Blei et. al [9], LDA is an unsupervised technique for modeling topics in a collection of documents. LDA utilities word co-occurrence information in order to group related words into a single topic. To infer topics from a corpus of documents, LDA represents documents as random mixtures over latent topics. Formally, LDA calculates two Dirichlet distributions: the word-topic distribution ϕ_k for topic k and the document-topic distribution θ_d for document d. The hyperparameters α and β are typically used as priors for ϕ and θ. For each word i in the dataset, a topic z_i is drawn from θ_d and the word w_i is drawn from ϕ_{z_i} distributions.

LDA's usecases range from traditional topic extraction for long texts [10, 43, 58, 74] to tag recommendation for search engines [40], to software systems categorization [77], and bug localization [49]. Despite its advantages, LDA suffers from several limitations when it comes to processing online user-generated text. For instance, in the context of app feedback analysis, mobile app reviews are often short, personal, and contain colloquial terms. Thus, they are too restricted semantically for complex distributional approaches such as LDA to operate, leading LDA to generate random topics or even overfit the data [3, 56, 60, 75, 85]. Furthermore, LDA, by design, tends to generalize over larger topics in order to better model frequently occurring words. Consequently, more specific, nuanced topics are often left ignored [33]. This limitation is critical for user review analysis as useful information in user reviews is often domain-specific [21, 67, 78, 82].

2.2 Keyword-Assisted Topic Modeling

Keyword-assisted topic modeling (keyATM) is a novel technique that has been proposed to improve upon traditional topic models, such as LDA [20, 33]. The key idea behind keyATM is that it incorporates user-defined *seed words* for topic-word distributions. Each potential topic can be supplemented by specific keywords that are believed to describe a theme. Formally, keyATM modifies the traditional LDA algorithm in two ways:

(1) The word-topic distribution ϕ_k is replaced with a "mixture" of two distributions: a seed-topic distribution ϕ^s and a regular-topic distribution ϕ^r. The seed-topic distribution can only select words from the initial seed set, while the regular-topic distribution may select any words in the corpus, including the seed words. The parameter π_k controls the probability of drawing a word from either ϕ^s or ϕ^r.

(2) To draw the document-topic distribution θ_d, for each document d, a binary vector $\vec{b}$ of the length S (number of seeded topics) is generated. $\vec{b}$ takes the values of 1 if d contains any keyword from a respective seed set and 0 otherwise. Next, a document-group distribution ζ^d is sampled from $\vec{b}$ with a hyperparameter τ from which a group variable g is drawn. Each group represents a seed set selected from the corpus. Finally, the group-topic distribution ψ_g is used as prior to draw θ_d.

Algorithm 1 shows the complete keyword-assisted topic model's generative process [33].

Algorithm 1 keyATM's topic generative process.

1: **for** topic $k = 1 \ldots T$ **do**
2: choose regular topic distribution $\phi_k^r \sim \text{Dir}(\beta_r)$
3: choose seeded topic distribution $\phi_k^s \sim \text{Dir}(\beta_s)$
4: choose parameter π_k ▷ prob. of drawing from seeded topic
5: **end for**
6: **for** seed set $s = 1 \ldots S$ **do**
7: choose group-topic distribution $\psi_s \sim \text{Dir}(\alpha)$ ▷ of length T
8: **end for**
9: **for** document $d = 1 \ldots D$ **do**
10: choose a binary vector $\vec{b}$ ▷ of length S
11: choose a document-group distribution $\zeta^d \sim \text{Dir}(\tau\vec{b})$
12: choose a group variable $g \sim \text{Mult}(\zeta^d)$
13: choose $\theta_d \sim \text{Dir}(\psi_g)$ ▷ of length T
14: **for** word $i = 1 \ldots N_d$ **do**
15: choose a topic $z_{i,d} \sim \text{Mult}(\theta_d)$
16: choose $x_i \sim \text{Bern}(\pi_{z_i})$ ▷ choose which topic distr. to draw from
17: **if** x_i is 0 **then**
18: select a word $w_i \sim \text{Mult}(\phi_{z_i}^r)$ ▷ from regular
19: **else**
20: select a word $w_i \sim \text{Mult}(\phi_{z_i}^s)$ ▷ from seeded
21: **end if**
22: **end for**
23: **end for**

Our main assumption in this paper is that keyATM can overcome the limitations of LDA when dealing with mobile app reviews. In particular, to address the domain-specificity problem, keyATM utilizes a binary vector $\vec{b}$ which elevates the less-common topics for the provided seed words. These seed words can be extracted in advance based on expert opinion in order to supplement keyATM with a high-level overview of the user review corpus. The main advantage of keyATM is that once the initial seeds are provided, keyATM can collect additional semantically-related keywords from regular topics (line 2, line 16). By combining seeded and regular distributions, keyATM generates more cohesive and focused topics, overcoming the main drawback of using LDA when it comes to modeling semantically-restricted user reviews.

3 APPROACH

The proposed approach is depicted in Fig. 1. In general, this approach can be divided into three main steps. In the first step, we apply several heuristics to extract informative user reviews from a specific application domain. In the second step, we preprocess and summarize extracted reviews in order to generate a representative set of important keywords, or seeds for the corpus. In the third step, seeds are selected and fed into keyATM to generate a topic distribution over the extracted reviews. In what follows, we describe each of these phases along with illustrative examples.

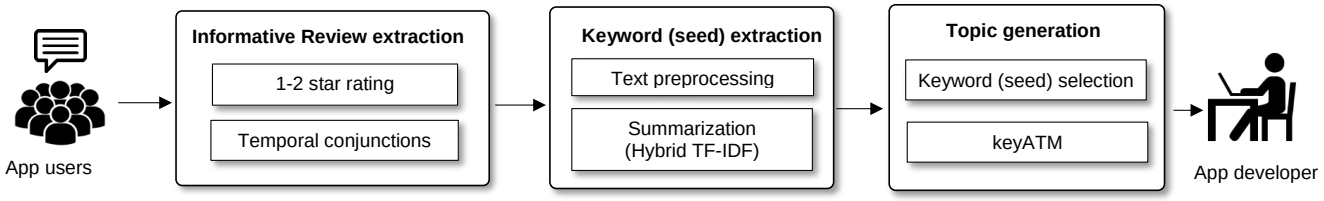

Figure 1: Our proposed approach for app review topic modeling

3.1 Informative Review Extraction

Mobile app reviews vary in quality. Previous research has shown that app reviews do not follow a well-defined structure and often contain spelling fluctuations, colloquial terms, and spam [37, 80]. Therefore, a large proportion of app store reviews is simply uninformative [37]. As topic models are particularly susceptible to generating uninterpretable topics from semantically-poor documents [18], the first step of our approach is to improve the quality of our review corpus by filtering out uninformative feedback.

To detect informative reviews, we adopt Guo and Singh's approach for synthesizing potentially meaningful user stories from mobile app reviews [23]. A *user story* can be defined as a relationship between an action that a user took and a problem that an app produced in response to that action. A user story describes users' experience and outcome when interacting with their apps' specific functionality. For example, a user might complain that their navigation app loses GPS signal when in drive mode. Such a story contains potentially useful information for app developers as it outlines the condition (*in drive mode*) under which a problem (*GPS lost signal*) occurred. These stories are commonly present in low-star app reviews (one and two stars) given that app problems are often accompanied by low user ratings [29, 36, 79]. Our expectation is that topics modeled after reviews with user stories will be more coherent, and thus, more interpretable.

To identify reviews that might contain user stories, common temporal conjunctions are used, including words such as *after*, *as soon as*, *before*, *every time*, *then*, *until*, *when*, *whenever*, *while*, and *during*. Temporal conjunctions indicate temporal and causal ordering of events that was found to be particularly helpful for mitigating the problems of text sparseness [53]. For example, consider the four app reviews in Example 1. $\mathbf{R}_1$, $\mathbf{R}_2$, and $\mathbf{R}_3$ are informative reviews that contain user stories - action-problem pairs (temporal conjunctions are underlined). $\mathbf{R}_1$ describes an action of *scroll through the pages* and the problem - a *crash*. Such a review can help our topic model to build a semantic association between *crash* and *scrolling*. $\mathbf{R}_4$ is a false positive.

Example 1

- $\mathbf{R}_1$: This app crashes <u>when</u> I scroll through the pages.
- $\mathbf{R}_2$: They want your SSN <u>before</u> you can even look. It's definitely a scam.
- $\mathbf{R}_3$: To verify identity it requires u take a picture but <u>then</u> immediately crashes.
- $\mathbf{R}_4$: Still waiting, <u>after</u> a month, to be approved.

3.2 Seed Generation

Under this step of our approach, we seek to generate sets of representative seeds (keywords) from extracted user reviews. To correctly model the underlying latent topic structure, these keywords have to be representative of as many themes in the review corpus as possible [33]. Therefore, the keyword generation process requires *a priori* knowledge of the domain of interest which might not be readily available to the researcher [4, 5]. For example, to model the representative topics of app reviews in the domain of Investing apps, the researcher has to know the specific themes that users discuss in the reviews and the corresponding keywords to generalize over these themes. Extracting these keywords involves manually classifying user reviews into representative topics, which nullifies the advantage of unsupervised techniques. To address these limitations, in our adaptation of keyATM, instead of determining seeds manually, we use extractive summarization.

A summary can be described as a short and concise description that encompasses the main theme of a collection of documents related to a similar topic [35, 46]. In extractive summarization, text artifacts (reviews, comments, tweets) which contain the most important (representative) keywords in the corpus are extracted as potential summaries of the entire corpus. In a sense, each generated summary represents a potential latent theme in a collection of documents, thus can then be used to provide representative keywords (seeds) for keyATM. Common extractive summarization techniques, such as Hybrid-TF.IDF [32] and SumBasic [57], have been applied to summarize unstructured online user feedback (e.g., tweets, YouTube comments, and user reviews) [34, 65, 81, 82] and have been shown to achieve very high levels of agreement with human-generated summaries. Based on these observations, in our analysis, we utilize such techniques to extract the initial set of seeding keywords from the corpus.

To generate summaries from mobile app user reviews, *Hybrid* TF.IDF [32] is often utilized. TF.IDF consists of two components: 1) TF - Term Frequency, or how many times a term appears in a document and 2) IDF - Inverse Document Frequency, or how much information a term provides. TF.IDF-based methods have shown acceptable accuracy levels across a variety of text summarization tasks [2, 16, 30, 38, 54]. However, short texts, such as user reviews, pose a unique challenge to TF.IDF. In particular, because short texts contain only a handful of words, the probability of individual terms occurring multiple times in a single document is low. Therefore, the majority of words get assigned the same TF value. Hybrid TF.IDF addresses this issue by calculating the term frequency as the total frequency of term t across all documents, divided by the

total number of terms. Formally, Hybrid TF.IDF weight for a term t can be computed as:

$$\text{Hybrid TF.IDF}(t,d) = \frac{f_{t,D}}{\sum f_{t,d}} \times \log \frac{|D|}{|d \in D : t \in d|} \quad (1)$$

where $f_{t,D}$ is the count of term t in all documents, $\sum f_{t,d}$ is the total count of all terms in the corpus, $|D|$ is the number of documents in the corpus, and $|d \in D : t \in d|$ is the number of documents that contain t. The total weight of a document d is calculated by summing up all terms' weights. However, in the current form, Hybrid TF.IDF would be biased toward longer documents as they contain more terms. To work around this problem, a normalization factor nf is introduced. The modified Hybrid TF.IDF formula for a document d can be defined as follows:

$$\text{Hybrid TF.IDF}(d) = \frac{1}{\max(nf, |d|)} \times \sum_{i=1}^{|d|} \text{Hybrid TF.IDF}(t_i, d) \quad (2)$$

The normalization factor is typically defined as the upper-bound of the required summary length (number of words) and can be determined experimentally. The actual summarization is then performed by ranking the documents by their total weight. To avoid a situation where summaries with similar words are ranked together, a similarity threshold is used, calculated as the cosine of the angle between the vectorized representations of each two summaries. An optimal similarity threshold can be set to a small positive number in advance, depending on the desired uniqueness of summaries. To illustrate our summarization step, consider the top-4 summaries in Example 2 generated for a dataset of reviews sampled from the domain of Investing apps with the threshold of 0.1 (no two summaries should have a cosine similarity greater than 0.1). Each summary encompasses a separate topic, such as taking money from user's account ($\mathbf{S}_1$), app crashing ($\mathbf{S}_2$), problems with selling a stock ($\mathbf{S}_3$), and issues with customer support ($\mathbf{S}_4$).

Example 2

- $\mathbf{S}_1$: This takes out money even after you close your account...
- $\mathbf{S}_2$: This app now crashes 100% of the time, every time I open it
- $\mathbf{S}_3$: Allowed me to buy stock, but when i tried to sell my stock they didn't sell it
- $\mathbf{S}_4$: Great until you need customer support, once you need support you're on your own

To improve the accuracy of summarization, text preprocessing strategies are often used. Before generating summaries, extracted app reviews are first converted to lowercase and tokenized into individual words, with punctuation, URLs, and other special symbols removed. Additional splitting strategies, such as splitting digits and alpha-characters are performed (e.g. *2hrs* becomes *2 hrs*). English stop-words, such as *will, this, it*, are removed based on the list provided in NLTK package [7]. Additional cohort-specific stopwords are manually identified and added to the list. These words include app names (*robinhood, acorn*), frequent words (*yeah, well*), and short 1-2 letter words that do not contain any semantic information. Finally, lemmatization is applied to the resulting list of words. Lemmatization is a normalization technique which reduces the number of distinct entries in the data. More specifically, lemmatization converts a word into its dictionary form. This process is applied to improve the performance of clustering algorithms by collapsing different forms of the same word into a single entity [39]. Example 3 presents the preprocessed summaries from Example 2.

Example 3

- $\mathbf{S}_1$: take, money, even, close, account
- $\mathbf{S}_2$: app, crash, time, every, time, open
- $\mathbf{S}_3$: allow, buy, stock, try, sell, stock, sell
- $\mathbf{S}_4$: great, need, customer, support, need, support

3.3 Topic generation

Since each summary succinctly describes a separate theme, seeds are generated by obtaining distinct terms from the preprocessed summaries. For example, the terms *great*, *need*, *customer*, and *support* are extracted from $\mathbf{S}_4$ to describe a theme discussing issues with customer support. These seeds are then supplied to keyATM for the topic modeling step. Our main expectation is that these terms should provide enough semantic information for keyATM to be able to generalize over the whole dataset of extracted user reviews. In what follows, we empirically evaluate our assumption using two datasets of mobile app reviews.

4 EVALUATION

In this section we illustrate the operation of our proposed topic generation process over two datasets of user reviews sampled form the domains of Investing and Food Delivery apps. We further evaluate our generated topics by comparing them to the topics generated by LDA. Our main research question is: ***How well does our approach perform in comparison to LDA?***

4.1 Data Collection

To demonstrate the operation of our approach, we apply it on two datasets of mobile app reviews sampled from the domains of Investing and Food Delivery apps. Investing apps have become increasingly popular in recent years due to the increasing interest in cryptocurrency trading. Zero-commission trading fees and continuous media coverage have multiplied the popularity effect of these apps by bringing in millions of new first-time traders. For example, Robinhood, a simplified Investing app, reported that more than 6 million new users joined the platform in 2021 right after the WallStreetBets *subreddit* controversy [66]. Similarly, the domain of Food Delivery has experienced an unprecedented growth during the COVID-19 pandemic as the demand for Food Delivery services has significantly increased. For example, the four major Food Delivery apps - DoorDash, UberEats, GrubHub, and Postmates reported a significant increase in revenue during the stay-at-home order of 2020 [73]. In fact, the market segment of food delivery apps, currently estimated at $126.91 billion, is expected to grow to $192.16 billion by 2025 [68].

Table 1: The number of user reviews extracted for each app in our dataset.

Investing		Food delivery	
App	**Reviews**	**App**	**Reviews**
Robinhood	7872	Uber Eats	58933
Acorn	4342	DoorDash	34917
Stash	2445	Grubhub	17784
E*TRADE	1605	Postmates	17610
Fidelity	1496	Seamless	1432
TD Ameritrade	1403		
Schwab	1079		
Personal Capital	509		

To collect user reviews for our analysis, we selected the most popular apps from both domains. To identify these apps, the top-100 apps in the category Finance (Investing) and Food&Drink (Food Delivery) on Google Play and the Apple App Store were examined. Apps, which met the following criteria were included in our analysis:

(1) For an app to be included in our analysis, we only considered apps with 10,000 reviews or more. This number of reviews is necessary in order to include only popular and well-established apps in our analysis.
(2) For the Investing domain, banking "all-in-one" apps were excluded as the majority of these apps did not provide Investing services. For Food Delivery apps, specific restaurants' delivery apps, such as *Papa John's Pizza & Delivery* official app, were also excluded.

After examining the top-100 apps, eight Investing and five Food Delivery apps were included. For each of these apps, we collected all textual reviews and star ratings on the Apple App Store and Google Play using Python web scrappers[1][2]. Overall, 370,820 app reviews were collected for our set of Investing apps and 266,544 reviews were collected for the set of Food Delivery apps. Out of these reviews, only 1-2 star rating reviews which included user stories (See Section 3.1) were considered in our analysis, a total of 20,760 reviews for the domain of Investing apps and 130,676 reviews for the Food Delivery domain. The distribution of extracted reviews over our apps is shown in Table 1.

4.2 Evaluation measures

Evaluating topic models can be a challenging task. Due to the fact that there is typically no ground-truth document-topic distribution that exists for every corpus, there is not a single objective metric to evaluate the quality of generated topics. To address this challenge, several topic evaluation techniques have been proposed in the literature. From among these techniques, Normalized Pointwise Mutual Information have been found to be closely correlated with human judgment of topic quality [15, 44, 69].

Introduced by Bouma [11], Normalized Pointwise Mutual Information (NPMI) is a information-theoretic measure of information overlap between words. NPMI can be measured by counting how many times two words appear in the same document versus how many times they appear separately. Formally, for two words w_i and w_j NPMI can be calculated as:

$$\text{NPMI}(w_i, w_j) = \frac{\log \frac{p(w_i, w_j)}{p(w_i)p(w_j)}}{-\log p(w_i, w_j)} \quad (3)$$

where $p(w_i, w_j)$ is the number of documents in which w_i and w_j appear together, and $p(w_i)p(w_j)$ is the the number of documents containing w_i and w_j respectively. The numerator of the NPMI formula is then normalized by dividing it by the negative log-transformed count of w_i and w_j appearing together. This results in a value between -1 when w_i and w_j never occur together and 1 when w_i and w_j only occur together.

The underlying assumption behind using NPMI for evaluating topic quality is that words of cohesive topics should be well connected, or have relatively high average pairwise NPMI. For example, Fig. 2 shows a sample NPMI graph of a word set sampled from the review corpus of the Food Delivery domain. The graph shows that the words *cold*, *driver*, *food*, *lost*, *late*, and *hot* form a dense-set of well-connected nodes (words). This is expected given that these words frequently appear in the same reviews, for example *"drivers are always either late or lost I always get my food cold and my drink hot."* The word *discount*, while connected to *food*, stands at a further semantic distance from other words as it does not appear as frequently with them in the same reviews.

There are two main strategies to compute NPMI: intrinsic and extrinsic [8]. Intrinsic NPMI is calculated based on the co-occurrence of topic words within the corpus. In contrast, the extrinsic strategy uses external datasets of human-produced textual knowledge, such as Wikipedia, to compute the co-occurrence, and thus semantic relatedness, of words. Intrinsic NPMI scores computed over the corpus can show how well the model learned the underlying data, or the extent that topic models accurately represent the content of a corpus. Extrinsic NPMI, on the other hand, shows how common generated topics are in daily language, which can be analogous to how a human examining the quality of topics would decide whether they are coherent or not [69].

In our analysis, we employ both strategies for computing topic coherence. For extrinsic evaluation, we download the entire English Wikipedia dump[3] of October 2017. The dump includes 5 million articles, 133 word-length per article on average, packed into a 16 GB JSON file. Each article was tokenized and preprocessed by converting into lowercase and removing special non-ASCII symbols. To calculate the coherence of a given topic is calculated as the average NPMI between its 10 most probable words in a topic (a total of 45 unique word-pairs). Formally, our topic coherence measure is calculated as:

$$coherence(t) = \frac{1}{45} \times \sum_{i=1}^{9} \sum_{j=i+1}^{10} \text{NPMI}(w_i, w_j) \quad (4)$$

[1]https://pypi.org/project/app-store-scraper/
[2]https://pypi.org/project/google-play-scraper/
[3]https://dumps.wikimedia.org/enwiki/

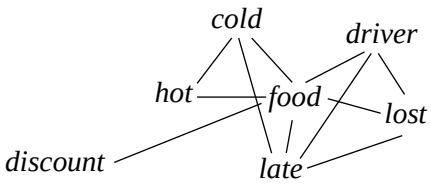

Figure 2: A connected NPMI graph of words extracted from the food delivery review corpus.

4.3 Model Configuration

To compare the performance of our approach to the baseline (LDA), we perform hyperparameter tuning in order to achieve the maximum coherence score possible over both datasets. In addition to the number of topics (K), LDA has two hyperparameters, α and β. We use the implementation of LDA from the Gensim Python package[4], where α is inferred from the corpus automatically and β is set to $1/K$. As a standard practice of topic models evaluation, we train LDA model for $K = \{10, 20, ..., 100\}$. These bounds of K were selected based on the coherence score, where a sharp decline indicates that a model no longer can generate coherent topics.

One of the main parameters that determine the performance of our approach is the quality of the seeding words. Smaller number of seeds might not convey enough semantic information for keyATM to capture meaningful topics, while larger number of keywords might be too general to form cohesive topics. To optimize of set of seeds, we calculated the pairwise NPMI scores for each seeds in each individual summary review. We then attempt to optimize the set of seeds by removing the bottom n-th percentile of the seeds (seeding words which share the lowest average pairwise similarity with other seeds). The main assumption is that removing these potentially unrelated words we can produce a more focused seeds, and thus, better topics. For example, Table 2 shows the average NPMI score for a group of words sampled from the summary reviews of the Food Delivery corpus. Words such as *refuse*, *first*, and *add* can be removed due to their low average pairwise similarity to other words in the group. To test how many seeds to consider, we include four model configurations in our analysis with $n = \{0, 5, 15, 25\}$, where keyATM_n refers to keyATM being trained after the n% of seeds at the lower end of NMPI score are removed. Each keyATM configuration is then trained for different value of K to determine the optimal $\langle n, K \rangle$ configuration for the review corpus.

4.4 Evaluation Results

A replication package of our analysis, including our datasets, is publicly available[5]. To answer our research question, we trained an LDA model and each configuration of keyATM over our two domains of app reviews for various K values. The coherence scores for the trained models are shown in Fig. 3. For each K, we compared the topic coherence scores by performing an independent two-tailed t-test between LDA and each configuration of keyATM. The results of this test are presented in Tables 3 and 4.

[4]https://pypi.org/project/gensim/
[5]https://github.com/icse2022submission/submission1

Table 2: Pairwise NPMI scores (intrinsic) over an example keyword set in the Food Delivery domain, sorted by average score. "refuse" is removed at bottom 5th, "first" is removed at bottom 15th, and "add" is removed at bottom 25th respectively.

	service	charge	fee	large	make	flat	use	place	add	first	refuse
service	1	0.10	0.14	0.07	0.08	0.06	0.13	0.05	0.01	0.06	0.10
charge	0.10	1	0.32	0.08	0.08	0.09	0.05	0.07	0.10	0.05	0.07
fee	0.14	0.32	1	0.13	0.09	0.15	0.06	0.04	0.24	0.04	0.00
large	0.07	0.08	0.13	1	0.11	0.20	0.06	0.10	0.10	0.07	0.07
make	0.08	0.08	0.09	0.11	1	0.06	0.07	0.08	0.08	0.08	0.06
flat	0.06	0.09	0.15	0.20	0.06	1	0.01	0.02	0.09	0.05	0.09
use	0.13	0.05	0.06	0.06	0.07	0.01	1	0.05	0.03	0.12	0.05
place	0.05	0.07	0.04	0.10	0.08	0.02	0.05	1	0.06	0.13	0.03
add	0.01	0.10	0.24	0.10	0.08	0.09	0.03	0.06	1	0.03	-0.05
first	0.06	0.05	0.04	0.07	0.08	0.05	0.12	0.13	0.03	1	0.02
refuse	0.10	0.07	0.00	0.07	0.06	0.09	0.05	0.03	-0.05	0.02	1

The results show that our approach outperformed LDA in terms of extrinsic and intrinsic coherence scores for both domains over all values of K, with minor exceptions. For the Investing domain, LDA's topic quality started to decline sharply after 20 topics, while our approach maintained a relatively flat coherence curve over the whole range of K. In terms of significance of the obtained results, we found that our approach tends to perform significantly better with a higher number of topics ($K = 50$ and $K \geq 70$). For $K = 10$, we found that our approach significantly outperforms LDA in terms of intrinsic coherence, which suggests that our approach infers the topics from the underlying Investing dataset better. For the Food Delivery domain, we observed a similar trend, with LDA's topic coherence dropping more sharply after the $K = 30$ mark, thus, every keyATM configuration significantly outperformed LDA for $K \geq 50$. Furthermore, some configurations, such as keyATM-15 and keyATM-25 significantly outperformed LDA across all $K \geq 20$ in terms of extrinsic coherence, which suggests that the produced topics even for smaller K values are more interpretable by humans.

We further observed that some keyATM configurations outperform each other for different K values. For example, in the Investing domain, keyATM-25 performs the best when $K = \{10, 40, 80, 100\}$, but produces slightly worse results for the rest of the Ks. To compare the performance of various keyATM configurations, we used an independent t-test for every pair of configurations. The general trend suggests that removing some percentile of unrelated keywords somewhat helps the model to produce more coherent topics for certain K values. However, not all improvements were statistically significant. In what follows, we discuss these trends along with their potential implications in greater detail.

5 DISCUSSION AND IMPACT

Based on our results, the main implication of our study suggests that it is possible to avoid a costly tagging process of a ground truth dataset and still outperform a traditional topic modeling technique over app review data. With only a handful of hyperparameters required to consider, our study outlines a *first-of-its-kind* approach for inferring meaningful latent topics from a semantically-restricted corpus of user reviews.

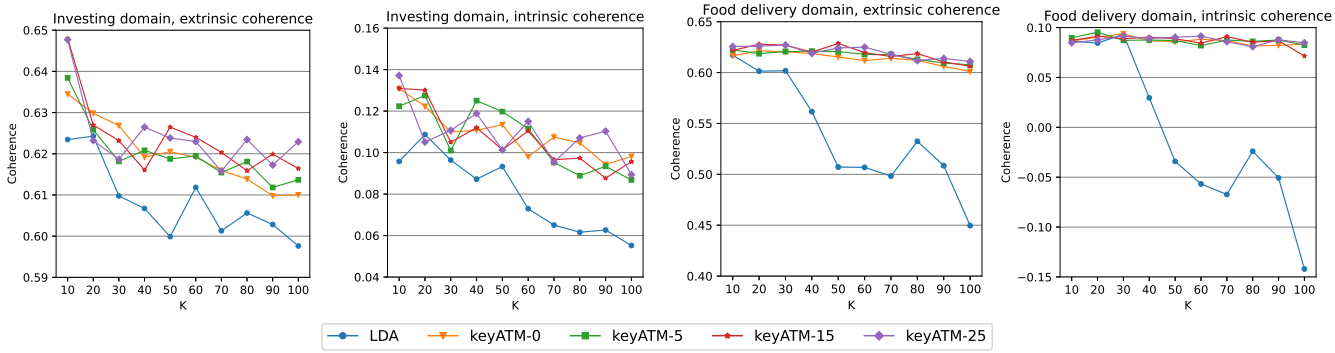

Figure 3: Intrinsic and extrinsic coherence scores for two domains of app reviews across all tested model configurations and K. For example, keyATM-5 is a keyATM model with the bottom 5th percentile of keywords removed from each summary.

Table 3: Independent t-test results (t-values) for the difference between extrinsic (Ext) and intrinsic (Int) coherence of the baseline (LDA) and the various keyATM model configurations for the Investing domain. Significant values are in bold. p-values are indicated:*p<0.05; †p<0.01; ‡p<0.001.

	keyATM-0		keyATM-5		keyATM-15		keyATM-25	
# of topics	Ext	Int	Ext	Int	Ext	Int	Ext	Int
10	0.794	**2.599***	0.974	1.905	1.428	**2.861***	1.532	**3.220†**
20	0.473	0.742	0.123	0.876	0.214	1.620	-0.079	-0.101
30	1.841	0.677	0.765	0.199	1.310	0.380	0.858	0.652
40	1.430	1.636	1.629	**3.536‡**	0.983	1.777	**2.226***	1.768
50	**2.418***	1.536	**2.126***	**2.129***	**3.022†**	0.454	**2.644†**	0.470
60	1.007	1.911	1.012	**3.213†**	1.543	**2.934†**	1.495	**3.616‡**
70	**2.094***	**3.495‡**	1.905	**2.182***	**2.882†**	**2.234***	**2.013***	1.809
80	1.398	**4.328‡**	**2.235***	1.924	1.627	**2.720†**	**3.025†**	**3.915‡**
90	0.984	**2.360***	1.284	**2.170***	**2.446***	1.557	1.955	**3.480‡**
100	**2.467***	**4.577‡**	**3.048†**	**2.705†**	**3.174†**	**3.245†**	**4.451‡**	**2.489***

Table 4: Independent t-test results (t-values) for the difference between extrinsic (Ext) and intrinsic (Int) coherence of the baseline (LDA) and the various keyATM model configurations for the Food Delivery domain. Significant values are in bold. p-values are indicated:*p<0.05; †p<0.01; ‡p<0.001.

	keyATM-0		keyATM-5		keyATM-15		keyATM-25	
# of topics	Ext	Int	Ext	Int	Ext	Int	Ext	Int
10	-0.061	0.002	0.507	0.345	0.434	0.086	0.749	-0.188
20	1.746	0.620	1.580	1.083	**2.319***	0.676	**2.148***	0.323
30	1.657	0.066	1.744	-0.727	**2.280***	-0.566	**2.382***	-0.256
40	**2.370***	1.687	**2.484***	1.719	**2.425***	1.798	**2.391***	1.757
50	**3.580‡**	**2.689†**	**3.763‡**	**2.717†**	**4.027‡**	**2.765†**	**3.868‡**	**2.798†**
60	**3.607‡**	**3.404‡**	**3.831‡**	**3.224†**	**3.878‡**	**3.305†**	**4.075‡**	**3.483‡**
70	**4.221‡**	**3.846‡**	**4.360‡**	**3.860‡**	**4.287‡**	**3.934‡**	**4.370‡**	**3.801‡**
80	**3.762‡**	**3.419‡**	**3.802‡**	**3.573‡**	**4.084‡**	**3.563‡**	**3.745‡**	**3.342†**
90	**4.265‡**	**3.946‡**	**4.441‡**	**4.112‡**	**4.498‡**	**4.104‡**	**4.638‡**	**4.040‡**
100	**5.755‡**	**5.835‡**	**6.061‡**	**5.815‡**	**5.991‡**	**5.447‡**	**6.174‡**	**5.884‡**

Our results provide evidence that it is possible to significantly improve the quality of the inferred topics by supplementing a topic model with a small set of representative keywords. In fact, even when the supplied keyword set is incomplete, the model is still able to correctly infer latent topics and provide interpretable results. For example, Table 5 presents the top-5 topics in terms of coherence generated for the reviews in our datasets. Topic 3 of the Investing app reviews is about losing money due to price fluctuations on a trading day. However, only a handful of words hinting toward that specific topic were provided to the model: ***sell***, ***stock***, and

lose. Nevertheless, the model correctly identified the main idea of the topic and discovered new related words. Another example is Topic 1 in the Food Delivery review corpus, which seems to be discussing a problem with customer service. Interestingly, the keywords ***customer*** and ***service*** were never supplied to this topic. Nevertheless, the model was still able to create a topic with a high coherence score. An interesting case is Topic 2 from the Investing domain - none of the keywords appeared in the top-10 words of the topic, however, the topic's theme is interpretable, pointing out to user interface (UI) problems. In fact, this topic which was obtained with $K = 10$ does not appear in LDA's topics at all for the similar values of $K = \{10, 20, 30\}$. This shows that our approach is able to extract underrepresented topics from a semantically-restricted dataset, where LDA typically fails.

As expected, smaller sets of focused keywords (seeds) produced the highest-quality topics. A large number of general keywords still managed to produce coherent topics, although such keywords are less likely to appear in the top-10 list of the topic. This suggests that the quality of the resulting model is influenced by the quality of the supplied keywords, therefore the main effort should be focused on the process of improving the seeding words for the topics. This process, in turn, is dependent on the underlying dataset. For example, using different summarization parameters can produce different seeds. As such, these parameters should be tuned along with the rest of the hyperparameters, such as K, α, and β.

In terms of specific model configurations, our results indicate a general trend toward higher coherence score produced by the configurations with a higher percentile removed keywords (n). The implications of this finding suggest that there is no *one-size-fits-all* approach when it comes to deriving an optimal model: each document corpus requires a comprehensive hyperparameter tuning strategy to create the best-performing model. For instance, in terms of raw coherence score, we found that keyATM-15 performed the best when K was set to 10 for Investing and $K = 50$ for Food Delivery domains. This difference in K can be explained by the difference in dataset size; the Food Delivery dataset has about 6.5 times more reviews than the Investing dataset.

In terms of impact, our expectation is that our approach would advance the state of the art by enabling further empirical investigations related to using topic modeling in software engineering tasks. In particular, topic modeling, especially LDA, has long been used to provide support for basic Software Engineering activities, such as requirements traceability [27], bug localization [76], code retrieval [48], and most recently mobile app review analysis. However, LDA's performance has always been hindered by the limited semantic and syntactic nature of software artifacts, whether source code, bug reports, requirements specifications, or software user reviews [17, 27, 52, 82]. Our proposed approach aims to address these challenges by assessing LDA to produce more cohesive topics that can be pointed out by few important words extracted from the corpus. Such words can be automatically determined using basic text summarization techniques that have been shown to achieve high levels of agreements with human generated summaries over collections of software user feedback [34, 65, 81]. This relatively simple methodology can overcome the limitations often associated with other expensive solutions, such as using machine learning to tune LDA's parameters [61], thus, enabling the development of more practical software engineering tools.

In terms of practical impact, our results highlight the need for a more nuanced, domain-aware approach for information extraction from user reviews. Similar to existing topic modeling and text classification algorithms, our approach can facilitate a transition from domain knowledge to requirements specifications. However, the key advantage of our approach over the existing techniques is that it can provide more fine-grained requirements information and simultaneously avoid the need to manually label a subset of reviews. The information obtained from keyATM can then be effectively used by app developers. For instance, developers of Investing apps may observe that UI (Topic 2) is a major concern of Investing app users, thus, concentrate their development efforts to improve user experience by extracting reviews where the UI topic has a high probability and analyze them, and may be even derive a more nuanced set of keywords (e.g., screen, color, graphs, etc.) to split the topic into any level of detail required. After the release of their apps, developers can further monitor user feedback and reallocate their resources more efficiently to quickly address the emerging user concerns.

6 RELATED WORK

The problem of extracting valuable information from app reviews has received significant attention in the literature [14, 24, 25, 34, 42, 45, 47, 51, 55, 70, 71, 78, 80, 82, 84]. A wide variety of supervised and unsupervised techniques have been used to mine such reviews for different categories of feedback. For example, Guo and Singh [23] proposed Caspar - an approach for extracting user stories from informative app reviews. A user story is an "action-problem" pair of events where a problem with an app is triggered by user action. The authors applied dependency parsing to extract temporally-related user stories from app reviews and train a bidirectional LSTM network for classification. Panichella et al. [63] proposed a tool for classifying user reviews into useful software maintenance categories. The approach uses NLP heuristics, such as common linguistic patterns, to formulate features for the classifier. The authors found that the structure of a review and its sentiment can predict the maintenance category with high precision and recall. Williams et al. [82] proposed a methodology to extract domain-specific user feedback from app reviews and tweets. The authors utilized Hybrid TF.IDF to extract important words from app reviews and then used PMI to derive relationships between them to form specific user concerns in a given app domain.

Along the line of our work, active learning was proposed to reduce the manual effort required to classify app reviews. Active learning algorithms learn the data incrementally by selecting a small sample of reviews for manual labeling based on uncertainty metric and predicting the remaining labels. The steps are repeated until the desired accuracy is reached. For example, Dhinakaran et. al. [19] evaluated an active learning pipeline for app review classification. The authors classified reviews into four categories: feature request, bug report, user experience, and rating. Several uncertainty sampling metrics were proposed and evaluated, such as Least Confident Prediction, Smallest Margin, and Highest Entropy. The experiment was conducted on Maalej et al.'s labeled

Table 5: Top-5 generated topics for the Investing and Food Delivery domains by the best performing model configurations. keyATM-15 with K=10 was used for the Investing domain, and keyATM-15 with K=50 was used for Food Delivery. Seeds are highlighted in bold. Example reviews are selected from the top-10 reviews for each topic.

Domain	Topics	Most probable words	Example review
Investing	Topic 1	**trade**, **day**, money, trading, **stock**, time, market, platform, **service**, like	...price could swing $.10-$.30/share easily between the time you've submitted a purchase and when it actually submits. Trading hours open up at 8am and only last until 5pm, the worst trading hours by far
	Topic 2	see, update, change, new, stock, like, screen, look, show, view	Please change the color and font of new Robinhood, it's hard to focus and it hurt my eyes if I look into app for while, not impress with this update.
	Topic 3	money, **sell**, **stock**, **lose**, price, market, time, buy, trade, trading	When I decided to sell my shares my orders were not executed at my price and the stock kept going down. Lost of hundred dollars.
	Topic 4	**account**, money, **email**, customer, bank, time, service, **day**, **say**, **support**	The customer service is HORRIBLE!!!!!!!!... Contacted customer service 2 or 3 times and after 24 hours... They say it takes about 10 days to close and get money transferred back to my bank.
	Topic 5	**money**, **account**, **take**, bank, invest, fee, charge, back, day, transfer	Reason for me giving a low rating is due to the fact that all it seems the app is doing is taking my money. I'm getting over drafts, I STILL don't see where my $700 in roundups went
Food delivery	Topic 1	**help**, **problem**, **would**, service, **could**, customer, app, **great**, **able**, order	Customer service is really bad. 2 days in a row they cancelled my orders
	Topic 2	app, **work**, **address**, order, try, time, use, **update**, even, get	When I type in my address and zip code and then hit Find Restaurants, it says "State is UNSET should be VALID"
	Topic 3	**drive**, driver, food, minute, **around**, house, away, **street**, go, car	Watched the driver drive to the next town over before delivering our food.
	Topic 4	app, order, time, **second**, **crash**, try, **twice**, **open**, use, every	Over the past few days the app has been crashing every single time I open it.
	Topic 5	app, ever, **number**, use, bad, give, **account**, **sign**, **one**, **star**	I can't even log in. It asks for my phone number...

datasets [50]. The authors showed that active learners that employ binary classifiers were more effective in terms of F-score for review classification tasks than passive learners.

Recently, unsupervised techniques have been used to discover latent topics in user reviews. Most of these approaches modify and extend LDA to increase its effectiveness for user generated feedback [12, 13, 56, 59, 67]. For instance, Mehrotra et al. [56] proposed grouping related short user tweets based on hashtags before supplying them to the LDA model. Qiao et al. [67] introduced the Latent Product Defect Mining Model (LPDM) for collecting domain-specific product defects from customer reviews. This approach augments LDA by including latent product components and their descriptions. While these techniques share the same goal of improving topic cohesiveness, to the best of our knowledge, our proposed approach is the first to utilize keyATM as well as text summarization techniques to generate more cohesive topics of user reviews.

7 THREATS TO VALIDITY

The study conducted in this paper suffers from several methodological constraints that might jeopardize the validity of our results. One major external validity threat might stems from the fact that our results might not be generalizable to other application domains of mobile apps. In an attempt to overcome this threat, we performed our analysis on two different domains of apps, Investing and Food Delivery. Furthermore, the datasets have drastically different sizes to ensure that our approach is able to produce coherent topics regardless of the amount of data available.

An internal validity threat might arise from the fact that we removed a large amount of reviews during our informative review extracting step in order to increase the quality of the review corpus. Therefore, some of the latent topics from the removed reviews might have been missed. However, performing review filtering is a standard practice for any supervised or unsupervised machine learning task. These methodologies have been shown to remove high percentage of uninformative reviews in review corpora with high levels of precision [23, 36]. Furthermore, the hyperparameter settings used to tune LDA might affect the internal validity of the study. However, there are no robust methodology available for every parameter for every scenario. Therefore, we applied a standard procedure of training several models with various K parameter values until the best results were obtained.

Another internal validity threat might stem from the apps selected for each domain. While we acknowledge that there are dozens, if not hundreds, of apps available under the Investing and Food Delivery domains, most of these apps do not have a sufficient number of reviews. Including such less popular apps would be problematic as the generated topics would be biased toward apps with more reviews. Nonetheless, we acknowledge the fact that the results of our analysis might not necessarily generalize over other apps in our domains or even over other domains.

Construct validity is the degree to which the various performance measures used in the study accurately capture the concepts they purport to measure. A construct validity threat might be raised about the reliability of the coherence measures used to evaluate the quality of generated topics. To address these threats, we used two types of intrinsic and extrinsic coherence measures that are based on the semantic relatedness of topic words. These measures have been extensively used in the literature and have been shown to achieve high correlation levels with human judgment [69]. Generally speaking, topic models are often utilized as a means to an end, where generated topics are used to help enable other tasks, such as information retrieval, or even used as input for machine learning algorithms, thus, they are better evaluated in that context. Nonetheless, further evaluation using human judges is necessary to paint a full picture of topic quality.

8 CONCLUSIONS

In this paper, we proposed an approach for analyzing mobile app user feedback using keyword-assisted topic models. The proposed approach relies on a set of seeding words (keywords) extracted from the corpus to generate more cohesive topics. In this paper, we showed that these keywords can be automatically extracted from the corpus using general-purpose extractive summarization techniques. The proposed approach was evaluated on two datasets of user reviews, sampled from the domains of Investing and Food Delivery apps. The results showed that our proposed keyword assisted topic modeling approach was able to significantly outperform LDA on both intrinsic and extrinsic measures of topic cohesiveness. Furthermore, our approach was able to model topics that are often overlooked by classical topic modeling techniques. Our findings in this paper are intended to advance the state-of-the-art in mobile app review analysis as well as enable further empirical investigations into topic modeling for software user feedback. To achieve these goals, our work in this paper will be extended across two main directions:

- Automatic tuning: we will continue to evaluate the proposed approach over larger datasets of mobile app reviews and across more application domains. Our objective is to devise automated optimization strategies for tuning the different parameters of our underlying topic modeling approach in different settings.
- Tool support: a working prototype which will implement our findings in this paper will be implemented and made publicly available. Such a prototype will enable us to examine the applicability and usability of our approach as well as its overall effectiveness in practical settings.

9 A PLACEHOLDER FOR REBUTTAL

ACKNOWLEDGMENT

A placeholder for acknowledging the funding agency

REFERENCES

[1] Charu Aggarwal and Chengxiang Zhai. 2012. A survey of text clustering algorithms. In *Mining Text Data*. Springer, 77–128.

[2] Nasser Alsaedi, Pete Burnap, and Omer Rana. 2016. Temporal TF-IDF: A high performance approach for event summarization in twitter. In *International Conference on Web Intelligence*. 515–521.

[3] Leticia Anaya. 2011. *Comparing Latent Dirichlet Allocation and Latent Semantic Analysis as Classifiers*. ERIC.

[4] David Andrzejewski and Xiaojin Zhu. 2009. Latent dirichlet allocation with topic-in-set knowledge. In *Workshop on Semi-Supervised Learning for Natural Language Processing*. 43–48.

[5] David Andrzejewski, Xiaojin Zhu, and Mark Craven. 2009. Incorporating domain knowledge into topic modeling via Dirichlet forest priors. In *International Conference on Machine Learning*. 25–32.

[6] Lidong Bing, Wai Lam, and Tak-Lam Wong. 2011. Using query log and social tagging to refine queries based on latent topics. In *International Conference on Information and Knowledge Management*. 583–592.

[7] Steven Bird. 2006. NLTK: the Natural Language Toolkit. In *Interactive Presentation Sessions*. 69–72.

[8] Stuart Blair, Yaxin Bi, and Maurice Mulvenna. 2020. Aggregated topic models for increasing social media topic coherence. *Applied Intelligence* 50, 1 (2020), 138–156.

[9] David Blei, Andrew Ng, and Michael Jordan. 2003. Latent Dirichlet Allocation. *The Journal of Machine Learning research* 3 (2003), 993–1022.

[10] Levent Bolelli, Şeyda Ertekin, and Lee Giles. 2009. Topic and trend detection in text collections using Latent Dirichlet Allocation. In *European Conference on Information Retrieval*. 776–780.

[11] Gerlof Bouma. 2009. Normalized (pointwise) mutual information in collocation extraction. *German Society for Computational Linguistics* 30 (2009), 31–40.

[12] Laura Galvis Carreno and Kristina Winbladh. 2013. Analysis of user comments: An approach for software requirements evolution. In *International Conference on Software Engineering*. 582–591.

[13] Dimple Chehal, Parul Gupta, and Payal Gulati. 2021. Implementation and comparison of topic modeling techniques based on user reviews in e-commerce recommendations. *Journal of Ambient Intelligence and Humanized Computing* 12, 5 (2021), 5055–5070.

[14] Ning Chen, Jialiu Lin, Steven Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: mining informative reviews for developers from mobile app marketplace. In *International Conference on Software Engineering*. 767–778.

[15] Kahyun Choi, Jin Ha Lee, Craig Willis, and Stephen Downie. 2015. Topic Modeling Users' Interpretations of Songs to Inform Subject Access in Music Digital Libraries. In *Joined Conference on Digital Libraries*. 183–186.

[16] Hans Christian, Mikhael Pramodana Agus, and Derwin Suhartono. 2016. Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF). *ComTech: Computer, Mathematics and Engineering Applications* 7, 4 (2016), 285–294.

[17] Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2012. Using IR methods for labeling source code artifacts: Is it worthwhile?. In *International Conference on Program Comprehension*. 193–202.

[18] Stefan Debortoli, Oliver Müller, Iris Junglas, and Jan Brocke. 2016. Text mining for information systems researchers: An annotated topic modeling tutorial. *Communications of the Association for Information Systems* 39, 1 (2016), 7.

[19] Venkatesh Dhinakaran, Raseshwari Pulle, Nirav Ajmeri, and Pradeep Murukannaiah. 2018. App review analysis via active learning: reducing supervision effort without compromising classification accuracy. In *IEEE International Requirements Engineering Conference*. 170–181.

[20] Shusei Eshima, Kosuke Imai, and Tomoya Sasaki. 2020. Keyword assisted topic models. *arXiv preprint arXiv:2004.05964* (2020).

[21] Necmiye Genc-Nayebi and Alain Abran. 2017. A systematic literature review: Opinion mining studies from mobile app store user reviews. *Journal of Systems and Software* 125 (2017), 207–219.

[22] Maria Gomez, Romain Rouvoy, Martin Monperrus, and Lionel Seinturier. 2015. A recommender system of buggy app checkers for app store moderators. In *International Conference on Mobile Software Engineering and Systems*. 1–11.

[23] Hui Guo and Munindar Singh. 2020. Caspar: Extracting and synthesizing user stories of problems from app reviews. In *International Conference on Software Engineering*. 628–640.

[24] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. 2015. Ensemble methods for app review classification: An approach for software evolution (n). In *International Conference on Automated Software Engineering*. 771–776.

[25] Emitza Guzman and Walid Maalej. 2014. How do users like this feature? a fine grained sentiment analysis of app reviews. In *IEEE International Requirements Engineering Conference*. 153–162.

[26] Kazuyuki Higashi, Hiroyuki Nakagawa, and Tatsuhiro Tsuchiya. 2018. Improvement of User Review Classification Using Keyword Expansion (S). In *International Conference on Software Engineering & Knowledge Engineering*. 125–124.

[27] Abram Hindle, Christian Bird, Thomas Zimmermann, and Nachiappan Nagappan. 2012. Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers?. In *IEEE International Conference on Software Maintenance*. 243–252.

[28] Liangjie Hong and Brian Davison. 2010. Empirical study of topic modeling in twitter. In *Workshop on Social Media Analytics*. 80–88.

[29] Leonard Hoon, Rajesh Vasa, Jean-Guy Schneider, and Kon Mouzakis. 2012. A preliminary analysis of vocabulary in mobile app user reviews. In *Computer-Human Interaction Conference*. 245–248.

[30] Eduard Hovy, Chin-Yew Lin, et al. 1999. Automated text summarization in SUMMARIST. *Advances in Automatic Text Summarization* 14 (1999), 81–94.

[31] Claudia Iacob and Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In *Conference on Mining Software Repositories*. 41–44.

[32] David Inouye and Jugal Kalita. 2011. Comparing twitter summarization algorithms for multiple post summaries. In *International Conference on Privacy, Security, Risk and Trust and International Conference on Social Computing*. 298–306.

[33] Jagadeesh Jagarlamudi, Hal Daumé, and Raghavendra Udupa. 2012. Incorporating lexical priors into topic models. In *Conference of the European Chapter of the Association for Computational Linguistics*. 204–213.

[34] Nishant Jha and Anas Mahmoud. 2018. Using frame semantics for classifying and summarizing application store reviews. *Empirical Software Engineering* 23, 6 (2018), 3734–3767.

[35] Elham Khabiri, James Caverlee, and Chiao-Fang Hsu. 2011. Summarizing user-contributed comments. In *International AAAI Conference on Web and Social Media*, Vol. 5.

[36] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E Hassan. 2014. What do mobile app users complain about? *IEEE software* 32, 3 (2014), 70–77.

[37] Mubasher Khalid, Muhammad Asif, and Usman Shehzaib. 2015. Towards improving the quality of mobile app reviews. *International Journal of Information Technology and Computer Science* 7, 10 (2015), 35.

[38] Rahim Khan, Yurong Qian, and Sajid Naeem. 2019. Extractive based Text Summarization Using K-Means and TF-IDF. *International Journal of Information Engineering & Electronic Business* 11, 3 (2019).

[39] Tuomo Korenius, Jorma Laurikkala, Kalervo Järveli, and Martti Juhola. 2004. Stemming and lemmatization in the clustering of finnish text documents. In *International Conference on Information and Knowledge Management*. 625–633.

[40] Ralf Krestel, Peter Fankhauser, and Wolfgang Nejdl. 2009. Latent Dirichlet Allocation for tag recommendation. In *Recommender Systems Conference*. 61–68.

[41] Donny Kristianto. 2021. Winning the Attention War: Consumers in Nine Major Markets Now Spend More than Four Hours a Day in Apps. https://www.appannie.com/en/insights/market-data/q1-2021-market-index/. Accessed: 2021-05-31.

[42] Zijad Kurtanović and Walid Maalej. 2017. Mining user rationale from software reviews. In *IEEE International Requirements Engineering Conference*. 61–70.

[43] Retno Kusumaningrum, Ihsan Aji Wiedjayanto, Satriyo Adhy, et al. 2016. Classification of Indonesian news articles based on Latent Dirichlet Allocation. In *International Conference on Data and Software Engineering*. 1–5.

[44] Jey Han Lau, David Newman, and Timothy Baldwin. 2014. Machine reading tea leaves: Automatically evaluating topic coherence and topic model quality. In *Conference of the European Chapter of the Association for Computational Linguistics*. 530–539.

[45] Xiaozhou Li, Boyang Zhang, Zheying Zhang, and Kostas Stefanidis. 2020. A Sentiment-Statistical Approach for Identifying Problematic Mobile App Updates Based on User Reviews. *Information* 11, 3 (2020), 152.

[46] Clare Llewellyn, Claire Grover, and Jon Oberlander. 2014. Summarizing newspaper comments. In *International AAAI Conference on Web and Social Media*, Vol. 8.

[47] Mengmeng Lu and Peng Liang. 2017. Automatic classification of non-functional requirements from augmented app user reviews. In *International Conference on Evaluation and Assessment in Software Engineering*. 344–353.

[48] Stacy Lukins, Nicholas Kraft, and Letha Etzkorn. 2008. Source Code Retrieval for Bug Localization Using Latent Dirichlet Allocation. In *Reverse Engineering*. 155–164.

[49] Stacy Lukins, Nicholas Kraft, and Letha Etzkorn. 2010. Bug localization using Latent Dirichlet Allocation. *Information and Software Technology* 52, 9 (2010), 972–990.

[50] Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik. 2016. On the automatic classification of app reviews. *Requirements Engineering* 21, 3 (2016), 311–331.

[51] Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *IEEE International Requirements Engineering Conference*. 116–125.

[52] Anas Mahmoud and Gary Bradshaw. 2017. Semantic topic models for source code analysis. *Empirical Software Engineering* 22, 4 (2017), 1956–2000.

[53] Inderjeet Mani, Marc Verhagen, Ben Wellner, Chungmin Lee, and James Pustejovsky. 2006. Machine learning of temporal relations. In *International Conference*

on Computational Linguistics and Meeting of the Association for Computational Linguistics. 753–760.

[54] Usha Manjari, Syed Rousha, Dasi Sumanth, and Sirisha Devi. 2020. Extractive Text Summarization from Web pages using Selenium and TF-IDF algorithm. In *International Conference on Trends in Electronics and Informatics (ICOEI)*. 648–652.

[55] Stuart McIlroy, Nasir Ali, Hammad Khalid, and Ahmed E Hassan. 2016. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering* 21, 3 (2016), 1067–1106.

[56] Rishabh Mehrotra, Scott Sanner, Wray Buntine, and Lexing Xie. 2013. Improving LDA topic models for microblogs via tweet pooling and automatic labeling. In *Conference on Research and Development in Information Retrieval*. 889–892.

[57] Ani Nenkova and Lucy Vanderwende. 2005. The impact of frequency on summarization. *Microsoft Research, Redmond, Washington, Tech. Rep. MSR-TR-2005* 101 (2005).

[58] Xiaochuan Ni, Jian-Tao Sun, Jian Hu, and Zheng Chen. 2009. Mining multilingual topics from Wikipedia. In *International Conference on World Wide Web*. 1155–1156.

[59] Ehsan Noei, Feng Zhang, and Ying Zou. 2019. Too many user-reviews, what should app developers look at first? *Transactions on Software Engineering* (2019).

[60] Jeungmin Oh, Daehoon Kim, Uichin Lee, Jae-Gil Lee, and Junehwa Song. 2013. Facilitating developer-user interactions with mobile app review digests. In *CHI Extended Abstracts on Human Factors in Computing Systems*. 1809–1814.

[61] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimilano Di Penta, Denys Poshynanyk, and Andrea De Lucia. 2013. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 522–531.

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

[63] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Visaggio, Gerardo Canfora, and Harald Gall. 2016. Ardoc: App reviews development oriented classifier. In *International Symposium on Foundations of Software Engineering*. 1023–1027.

[64] Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. 2015. Leveraging user reviews to improve accuracy for mobile app retrieval. In *International Conference on Research and Development in Information Retrieval*. 533–542.

[65] Elizabeth Poché, Nishant Jha, Grant Williams, Jazmine Staten, Miles Vesper, and Anas Mahmoud. 2017. Analyzing user comments on YouTube coding tutorial videos. In *International Conference on Program Comprehension*. 196–206.

[66] PYMNTS. 2021. High-Speed Traders Pay Robinhood $331 Million In Q1 To Execute Trades. https://www.pymnts.com/earnings/2021/high-speed-traders-pay-robinhood-331-million-dollars-q1-execute-trades/. Accessed: 2021-06-24.

[67] Zhilei Qiao, Xuan Zhang, Mi Zhou, Gang Alan Wang, and Weiguo Fan. 2017. A domain oriented LDA model for mining product defects from online customer reviews. (2017).

[68] Research and Markets. 2021. Global Online Food Delivery Services Market Report 2021: Market is Expected to Reach $192.16 Billion in 2025, from $126.91 Billion in 2021 - Long-term Forecast to 2030. https://www.prnewswire.com. Accessed: 2021-07-24.

[69] Michael Röder, Andreas Both, and Alexander Hinneburg. 2015. Exploring the space of topic coherence measures. In *International Conference on Web Search and Data Mining*. 399–408.

[70] Furqan Rustam, Arif Mehmood, Muhammad Ahmad, Saleem Ullah, Dost Muhammad Khan, and Gyu Sang Choi. 2020. Classification of shopify app user reviews using novel multi text features. *IEEE Access* 8 (2020), 30234–30244.

[71] Andrea Di Sorbo, Sebastiano Panichella, Carol Alexandru, Junji Shimagaki, Corrado Visaggio, Gerardo Canfora, and Harald Gall. 2016. What would users change in my app? summarizing app reviews for recommending software changes. In *International Symposium on Foundations of Software Engineering*. 499–510.

[72] Statista. 2021. Number of available apps in the Apple App Store from 2008 to 2020. https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/. Accessed: 2021-05-31.

[73] Levi Sumagaysay. 2020. The pandemic has more than doubled food-delivery apps' business. Now what? https://www.marketwatch.com. Accessed: 2021-07-24.

[74] Shaheen Syed and Marco Spruit. 2017. Full-text or abstract? Examining topic coherence scores using Latent Dirichlet Allocation. In *International Conference on Data Science and Advanced Analytics*. 165–174.

[75] Maria Terzi, Maria-Angela Ferrario, and Jon Whittle. 2011. Free text in user reviews: Their role in recommender systems. In *Workshop on Recommender Systems and the Social Web at International Conference on Recommender Systems*. 45–48.

[76] Stephen Thomas, Meiyappan Nagappan, Dorothea Blostein, and Ahmed Hassan. 2013. The impact of classifier configuration and classifier combination on bug localization. *IEEE Transactions on Software Engineering* 39, 10 (2013), 1427–1443.

[77] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. 2009. Using Latent Dirichlet Allocation for automatic categorization of software. In *International Working Conference on Mining Software Repositories*. 163–166.

[78] Miroslav Tushev, Fahimeh Ebrahimi, and Anas Mahmoud. 2020. Digital Discrimination in Sharing Economy A Requirements Engineering Perspective. In *IEEE International Requirements Engineering Conference*. 204–214.

[79] Rajesh Vasa, Leonard Hoon, Kon Mouzakis, and Akihiro Noguchi. 2012. A preliminary analysis of mobile app user reviews. In *Computer-Human Interaction Conference*. 241–244.

[80] Jianyu Wang, Rui Wen, Chunming Wu, Yu Huang, and Jian Xion. 2019. Fdgars: Fraudster detection via graph convolutional networks in online app review system. In *World Wide Web Conference*. 310–316.

[81] Grant Williams and Anas Mahmoud. 2017. Mining Twitter Feeds for Software User Requirements. In *International Requirements Engineering Conference*. 1–10.

[82] Grant Williams, Miroslav Tushev, Fahimeh Ebrahimi, and Anas Mahmoud. 2020. Modeling user concerns in Sharing Economy: the case of food delivery apps. *Automated Software Engineering* 27, 3 (2020), 229–263.

[83] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2013. A biterm topic model for short texts. In *International Conference on World Wide Web*. 1445–1456.

[84] Hui Yang and Peng Liang. 2015. Identification and Classification of Requirements from App User Reviews.. In *International Conference on Software Engineering & Knowledge Engineering*. 7–12.

[85] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. 2011. Comparing twitter and traditional media using topic models. In *European Conference on Information Retrieval*. 338–349.