Classifying Mobile Applications Using Word Embeddings

FAHIMEH EBRAHIMI, MIROSLAV TUSHEV, and ANAS MAHMOUD*

Modern application stores enable developers to classify their apps by choosing from a set of generic categories, or genres, such as health, games, and music. These categories are typically static—new categories do not necessarily emerge over time to reflect innovations in the mobile software landscape. With thousands of apps classified under each category, locating apps that match a specific consumer interest can be a challenging task. To overcome this challenge, in this paper, we propose an automated approach for classifying mobile apps into more focused categories of functionally-related application domains. Our aim is to enhance apps visibility and discoverability. Specifically, we employ word embeddings to generate numeric semantic representations of app descriptions. These representations are then classified to generate more cohesive categories of apps. Our empirical investigation is conducted using a dataset of 600 apps, sampled from the Education, Health&Fitness, and Medical categories of the Apple App Store. The results show that, our classification algorithms achieve their best performance when app descriptions are vectorized using GloVe, a count-based model of word embeddings. Our findings are further validated using a dataset of Sharing Economy apps and the results are evaluated by 12 human subjects. The results show that GloVe combined with Support Vector Machines can produce app classifications that are aligned to a large extent with human-generated classifications.

CCS Concepts: • **Software and its engineering** → *Documentation.*

Additional Key Words and Phrases: App classification, word embeddings, app store, GloVe, Word2Vec, fastText

ACM Reference Format:
Fahimeh Ebrahimi, Miroslav Tushev, and Anas Mahmoud. 2021. Classifying Mobile Applications Using Word Embeddings. *ACM Trans. Softw. Eng. Methodol.* 37, 4, Article 111 (August 2021), 30 pages. https://doi.org/10.1145/3474827

1 INTRODUCTION

Over the past decade, mobile application (app) stores, such as Google Play and the Apple App Store, have expanded in size to host millions of apps, offering app users virtually unlimited options to choose from. These apps are typically classified under several categories (e.g., Gaming) and subcategories (e.g., Sport, Board, and Card) that are intended to help consumers discover apps more effectively. For instance, the Apple App Store, which currently hosts close to 1.8 million apps, classifies apps under 23 distinct categories, while Google Play, which currently hosts close to 2.87 million apps, offers 35 distinct categories of apps [1].

With thousands of apps classified under each category, locating apps that match a specific consumer interest can be a challenging task [89]. Furthermore, categorizing apps under broad categories of loosely related functionalities can severely impact their discoverability, thus their download rates and chances of survival. These challenges have encouraged experts, across a broad range of application domains, to propose more accessible classification schemes of apps in their fields [11, 37, 94]. For instance, apps under the Health&Fitness category are often classified by

Authors' address: Fahimeh Ebrahimi, febrah1@lsu.edu; Miroslav Tushev, mtushe1@lsu.edu; Anas Mahmoud, amahmo4@lsu.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Association for Computing Machinery.
1049-331X/2021/8-ART111 $15.00
https://doi.org/10.1145/3474827

healthcare professionals into more specific categories (e.g., health interventions, consulting, and patient management, etc.) to increase their visibility to doctors and patients [37, 94]. However, these classifications are often static, relying on a manual synthesis of app descriptions or their usage scenarios. Therefore, they can hardly adapt to the rapid pace of innovation in the app market or the large number of new apps approved daily by popular app stores.

Automated approaches that are proposed to solve this problem often employ standard classification techniques to categorize apps based on their publicly available app store descriptions [4, 7, 54, 61]. Other information, such as download rates of apps, their usage scenarios, ratings, and source code have also been used to generate more accurate classification models [80, 97]. However, these techniques are often limited by the restricted syntactic nature of app descriptions (e.g., text sparsity and vocabulary mismatch) [65], the complexity associated with collecting certain types of app information (e.g., source code and usage scenarios), and the general lack of expert-generated ground-truths to assess the performance of generated models.

To overcome these limitations, in this paper, we propose an automated approach for classifying mobile apps using word embeddings. Word embeddings produce semantic vector representations of words in a text collection [68]. Such numeric representations can be used to accurately estimate the semantic similarity between important words in mobile app descriptions, thus, help to overcome the syntactic limitations of these descriptions [20]. Our empirical analysis is conducted using a dataset of 600 apps, sampled from the Education, Health&Fitness, and Medical app categories of the Apple App Store. Existing expert-generated classifications of apps are then used to assess the accuracy of our classifiers and compare their performance to several existing classification and text modeling methods [11, 94]. Our approach is then validated using a dataset of Sharing Economy apps, sampled from a broad range of application domains, such as ride-sharing (e.g. Uber and Lyft), lodging (e.g., Airbnb and Couchsurfing), and freelancing (e.g., TaskRabbit and UpWork). We further conduct a study with 12 participants (judges) to assess the quality of our generated classifications.

The remainder of this paper is organized as follows. Section 2 reviews related work and motivates our research. Section 3 introduces word embeddings. Section 4 describes our research oracle. Section 5 presents our experimental setup and analysis. Section 6 describes our human study. Section 7 discusses our main findings. Section 8 describes the potential limitations of our study. Finally, Section 9 concludes the paper and presents our directions of future work.

2 RELATED WORK AND MOTIVATION

In this section, we review existing work related to mobile app classification, discuss its limitations, and motivate our approach.

2.1 Related work

Motivated by the vast growth of mobile app stores, the research on mobile apps classification has gained considerable momentum over the past few years. For instance, Zhu et al. [96, 97] proposed an automated approach for classifying mobile apps in the Nokia Store. The proposed approach leveraged knowledge available about the apps on search engines (e.g., Google) and their contextual features (usage patterns), extracted from the device logs of app users. These features were then combined using a Maximum Entropy model for training an app classifier. A dataset of 680 apps containing device logs of 443 users was used to evaluate the proposed approach. The results showed that the proposed classifier outperformed other approaches based on topic modeling and word vector analysis.

Berardi et al. [7] proposed a technique for classifying mobile apps into 50 customer-defined classes. The authors crawled Google Play and the Apple App Store to extract apps meta-data, including their descriptions, categories, names, ratings, and size. A Support Vector Machines

(SVMs) classifier was then trained based on the extracted features. All the selected features were weighted by using BM25 as a weighting function [76]. Evaluating the proposed approach over a dataset of 5,792 apps resulted in an F_1 score of 89%.

Sanz et al. [80] proposed a machine learning approach for categorizing apps in the Android app store. The objective was to organize the Android market and detect malicious apps. The proposed approach utilized features extracted from the source code of apps, their requested permissions, and meta-data, including their ratings, size, and advertised permissions. Multiple classification algorithms were then used to classify a dataset of 820 apps sampled from 7 different categories of Google Play. The results showed that Bayesian networks outperformed other algorithms with an Area Under the Curve (AUC) of 93%.

Lulu and Kuflik [54] used unsupervised machine learning to cluster apps based on their functionalities. Specifically, app features were extracted from their app store descriptions and then enriched by content from professional blogs. App features were then represented using TF.IDF weighted vectors of words. Synonymy relations were resolved using WordNet. The authors then used hierarchical clustering to generate hierarchies of functionally-related apps. The effectiveness of the proposed approach was demonstrated on a dataset of 120 apps sampled from Google Play.

Mokarizadeh et al. [61] employed Latent Dirichlet Allocation (LDA) to categorize Android mobile apps. Specifically, LDA was used to model app descriptions (features) [9] and K-means was then used to group similar apps together based on their topic models. The proposed approach was applied to two datasets of Android apps. The results revealed that the default categorization in Google Play did not group apps with similar topics together. Similar to this work, Vakulenko et al. [89] also used topic modeling to group similar apps in the Apple App Store. The authors used LDA to identify recurrent topics in app descriptions. Apps were then classified based on their topic models into 66 categories adapted from the categories and subcategories of the Apple App Store. The results showed that extracted topics extended the original App Store categories and provided in-depth insights into the content of different categories.

Nayebi et al. [64] also leveraged LDA to extract topics from the descriptions of mobile apps. The authors further considered the number of downloads, the number of reviews, and the average ratings as app classification features. DBSCAN was then used to cluster apps into different categories. The proposed technique was evaluated using a dataset of 940 open source apps, sampled from F-Droid. The results showed that DBSCAN performed better in producing homogeneous clusters of apps when using the market attributes of apps (e.g., ratings, downloads, and file size) rather than the topics extracted from their descriptions.

Al-Subaihin et al. [4] proposed a novel approach for app clustering based on their textual features. App features were extracted from their app store descriptions using information retrieval augmented with ontological analysis. Specifically, NLTK's N-gram Collocation Finder was used to extract lists of bi- or tri-grams of commonly collocating words, or *featurelets*, such as <*view*, *image*> or <*send*, *message*>. Agglomerative Hierarchical Clustering (AHC) was then used to cluster apps based on their extracted featurelets. The similarity of feature words was estimated using WordNet. The proposed approach was evaluated using 17,877 apps mined from the BlackBerry app store and Google Play. The cohesiveness of generated clusters was then assessed by human judges. The results showed that the proposed technique improved the default categorizations available in modern app stores.

In a more recent work, Al-Subaihin et al. [3] conducted an empirical comparison of text-based app clustering techniques, including topic modeling (LDA) and keyword feature extraction methods [18]. The analysis was conducted using a dataset of 12,664 mobile app descriptions extracted from Google Play. The results showed that, in terms of quantitative cluster quality, LDA-based solutions performed the best.

Gorla et al. [25] proposed CHABADA, a technique for identifying inconsistencies between the advertised behavior of Android apps and their implemented behavior. The authors leveraged LDA to extract topics from the descriptions of mobile apps. The extracted topics were fed into a K-means algorithm to form distinct clusters of apps. Within each cluster, sensitive APIs governed by user permissions were identified. Outliers with respect to API usage were detected using SVMs. These outliers were considered potentially malicious activities. CHABADA was tested on a dataset of 22,500+ Android apps. The prototype was able to detect several anomalies and flag 56% of novel malware.

2.2 Motivation

The problem of app classification will persist as long as the number of apps in app stores continues to grow. The search engines of popular app stores used to provide adequate accessibility to apps [52]. However, after the explosive growth in the mobile app market in recent years as well as the constant changes in app store ranking policies, relying on a general keyword search can no longer guarantee equal access to apps [24, 51]. This can have catastrophic impacts on the discoverability of apps, and thus, their survivability. Dynamic app classification engines can mitigate this problem by providing a basis for building new independent app search frameworks that can help different user populations (e.g., health professionals, educators, and businessmen) to find apps that fit their specific needs. This can be particularly important in domains such as Health&Fitness, where recent evidence has shown that having access to the right app can help the quality of health among the general public and help health professionals to communicate better with their patients [14, 86].

Rigorous classification techniques can also provide researchers, across a broad range of disciplines (e.g., business, education, and gaming, etc.), with a framework to automatically zoom-in into their specific domains of interest and get unique in-depth insights into the evolution of apps in such domains in terms of features and user goals. Furthermore, app developers can use such techniques early in the process to explore their ecosystem of competition, or any apps that share their specific set of features. Understanding the domain of competition is crucial for app success and survival [48].

2.3 Limitations of Existing Solutions

Our review of related work has exposed several limitations affecting existing app classification solutions. These limitations can be described as follows:

- **Classification features:** A plethora of classification features are used to classify apps. These features are often extracted from the textual descriptions of apps [3, 4, 54, 61, 64], their available meta-data (e.g., ratings, price, etc.) [7, 80], their source code [80], the APIs they use [25], and in some cases, their usage data [97]. In general, going beyond publicly available data can generate unnecessary complexities. For instance, meta-data of apps provide little to no information about their features, and their source code is not always available, and sometimes, obfuscated. In addition, collecting app usage information can raise major privacy concerns, especially if such data is being collected at a large scale.
- **Classification models:** Existing research showed that, due to vocabulary mismatch problems, relying solely on the syntactic attributes (words) of app descriptions, using techniques such as VSM, can generate suboptimal models [3]. Therefore, semantically enabled techniques, such as topic modeling, are commonly used to generate semantic representations of app descriptions. However, such techniques suffer from high operational complexity. For instance, the topic modeling technique LDA requires tuning multiple hyper-parameters in order to generate cohesive topics. These parameters are determined based on heuristics or using the default values provided by tools such as Gensim [74]. Furthermore, such techniques

often suffer when dealing with smaller text snippets such as apps descriptions. Due to these limitations, in most cases, the probabilistic distributions of generated topics are not reflective of actual feature topics. Similar problems can be detected in clustering techniques, such as AHC and DBSCAN, where the number of clusters has to be optimized based on subjective measures of cohesiveness.

- **Classification labels:** In the majority of existing work, alternative *ad-hoc* categorizations, or classification labels, are proposed by researchers to be used as *ground-truth* to assess the performance of classification algorithms [4, 97]. However, such labels are often subjective, and in many cases ignore aspects of apps that domain experts, such as doctors, educators, and gamers, might find important for their target user population.

To overcome these limitations, in this paper we propose a new approach for classifying mobile apps. Our approach uses word embeddings as an underlying technique to generate semantic representations of app descriptions. For our classification categories, or ground-truth, we utilize expert-generated classifications of apps. These classifications are independently produced by experts with the intention of making apps more accessible to their target users in their domains of operation.

3 WORD EMBEDDINGS

Word embeddings are a type of semantically-aware word representation that allows words with similar meanings to have similar representations. This unique interpretation of text builds upon the distributional hypothesis of Harris, which states that semantically-related words should occur in similar contexts [31]. Formally, a word embedding is a word vectorization technique which represents individual words in a corpus using multi-dimensional vectors of numeric values that are derived from the intrinsic statistical properties of the corpus. Words that have similar meanings should have similar vectors (closer in the vector space). These dense representations proved to be effective for calculating similarities between words using vector geometry, allowing basic computations on these words (low-dimensional matrices) to yield meaningful results (e.g., *India - Delhi* $\approx$ *France - Paris*, both of these vector subtractions encode the concept of *Capital*) and facilitating more effective automated solutions (e.g., deep learning) for challenging natural language processing (NLP) problems, including document classification [32, 33, 49], sentiment analysis [6, 35, 44], and text summarization [2, 43]. Word2Vec [58], GloVe [68], and fastText [10] are among the most commonly used models of word embeddings [28, 90]. In what follows, we describe these models in greater detail.

3.1 Word2Vec

Introduced by Mikolov et al. [58], Word2Vec is a two-layer neural network that utilizes one of two models to produce word embeddings, a Continuous Bag-Of-Words model (CBOW) and a Skip-gram model. The CBOW model, depicted in Fig. 1-a, predicts a word given its surrounding context, while the Skip-gram model, shown in Fig. 1-b, uses a word's information to predict its surrounding context. The context of a word w_i is defined by its neighbor words, composed of k words to the left of w_i and k words to its right. k is a hyperparameter of the model, known as the *window size*. Word2Vec predicts the probability that the word w_i is in the context of w_j with the following softmax equation:

$$p(w_i|w_j) = \frac{exp({V'_{wi}}^T V_{wj})}{\sum_{l=1}^{V} exp({V'_{wl}}^T V_{wj})} \tag{1}$$

where V'_{wi} is the output vector representation of the target word w_i, V_{wj} is the input vector representation of the word w_j, and V is the vocabulary size. The exp and T stand for exponential

and transpose, receptively. The most commonly used pre-trained Word2Vec is learned on more than 100 billion words from the Google News dataset. This model includes 300-dimensional vectors of 3 million words and phrases[1].

3.2 Global Vectors (GloVe)

Introduced by Pennington et al. [68], GloVe uses the similarities between words as an invariant to generate their vector representations, assuming that words that occur in similar contexts are more likely to have similar meanings. Similar to Word2Vec, GloVe generates a numeric vector representation of words to preserve their contextual information. However, unlike the predictive-based model of Word2Vec, GloVe is a count-based model. Specifically, GloVe initially constructs a high dimensional matrix of words co-occurrence. Dimensionality reduction is then applied to the co-occurrence count matrix of the corpus. In this matrix, each row shows how often a word co-occurs with other words in a predefined context window in a large corpus. By applying a matrix factorization method on the count matrix, a lower dimension matrix is produced, where each row is the vector representation of a word. The dimension reduction approach aims to minimize the *"reconstruction loss"*, thus yield the best lower-dimension matrix that can explain most of the variances in the original matrix and capture the statistics of the entire corpus in its model.

The most commonly used pre-trained GloVe is trained over a six billion token corpus. This corpus was constructed using a combination of *Wikipedia 2014*, which had 1.6 billion tokens, and *Gigaword 5*, which had 4.3 billion tokens. The context window size is set to 10. The vocabulary dictionary of this dataset contains 400,000 most frequent words. This pre-trained model represents word vectors in four dimensions: 50, 100, 200, and 300[2]. Several studies showed that GloVe outperformed Word2Vec and other dimensionality reduction baselines, such as Singular Value Decomposition, over many tasks, including estimating word similarity and Named Entity Recognition [28].

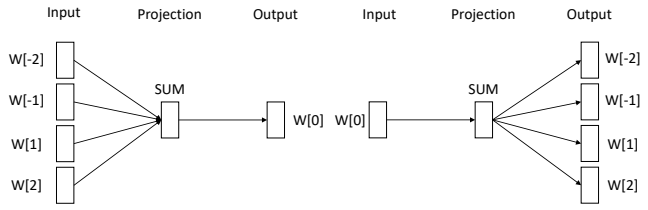
3.3 fastText

fastText [10] is another word-embedding model that was developed by Facebook AI in 2016. This model is based on Word2Vec Skip-Gram model. One of the main advantages of fastText is the fact that it considers the internal structures of words to generate their embeddings. In particular, unlike Word2Vec, which takes individual words as input, fastText breaks words into character n-grams. The vector representation of a single word is generated by averaging the vectors of its n-grams. For instance, the word vector of *"diet"* is a sum of the numerical representations of the n-grams: *"di"*, *"die"*, *"diet"*, *"ie"*, *"iet"*, and *"et"*. Using these n-grams, fastText can generate embeddings for words that do not exist in the original corpus. Multiple pre-trained models of fastText are available. In our analysis, we used the Wiki-news[3] vector representation model which generates one million word vectors learned on *Wikipedia 2017*, *UMBC corpus*, and *statmt.org*. This model contains 16 billion tokens, each represented as a numerical vector of dimension 300.

4 DATA AND ORACLE

In the context of supervised data mining, the term oracle refers to *"any mechanism, manual or automated, for determining the ground truth associated with inputs to be classified"* [30]. In this section, we describe our research oracle, including our data collection process, expert-categorizations, and ground-truth generation.

[1]https://code.google.com/archive/p/word2vec/
[2]https://nlp.stanford.edu/projects/glove/
[3]https://fasttext.cc/docs/en/english-vectors.html



(a) The CBOW Model (b) The Skip-Gram Model

Fig. 1. The two architectures of Word2Vec [58].

4.1 App Data Collection

To collect our data, we developed a Python crawler to automatically scrape app descriptions from the Apple App Store. The dataset used in our analysis was collected in July of 2019. To extract app information, we crawled the web interface of iTunes. The Apple App Store lists all apps classified under each category in alphabetical order, indexed by English letters. The HTML pages associated with each letter were scraped to extract listed app's URL and iTunes ID. In total, 1,893,256 apps' IDs were scraped. This process is shown in Fig. 2.

In the next step, we developed another crawler to extract apps' meta-data, including name, description, category, price, and rating average. As shown in Fig. 3, for each of the scraped IDs, the crawler requested each app's web page. The crawler then extracted each app's meta-data by parsing its web page. We used a language detection library to detect the app's description language[4] and exclude non-English apps. In total, the meta-data of 1,479,203 English apps were collected.

4.2 Expert Categorization

One of the main limitations of existing work on app classification is the lack of an expert-verified oracle (e.g., alternative categorization) of the apps being classified. In general, oracles in existing research are either generated by researchers [4, 7] or based on the default categorizations of app stores [80, 82]. To overcome these limitations, in our analysis, we used two existing expert-generated categorizations of apps. The first categorization was introduced by Cherner et al. [11]. In their work, Cherner et al. utilized qualitative research methods to classify Education apps into several categories and subcategories. These categories can be described as follows:

- **Skill-based**: This category includes educational apps that use rote memorization to help students build specific skills, such as literacy, numeracy, science, subject area, reading, and test preparation. Examples of popular apps under this category include, *Vocabulary Builder*, *Khan Academy*, and *LearnEnglish Grammar*.
- **Content-based**: Apps under this category provide access to educational data. These apps are further divided into two groups: subject_area and reference. Subject_area apps contain static pre-programmed educational content. Reference apps, on the other hand, allow users to search and explore a variety of topics. Examples of popular apps under this category include, *Wikipedia*, *Google Earth*, and *Dictionary*.
- **Function-based**: Apps under this category help transforming the learned content into usable formats. These apps are used for note-taking, presentation, organizing graphics, following

[4]https://github.com/shuyo/language-detection

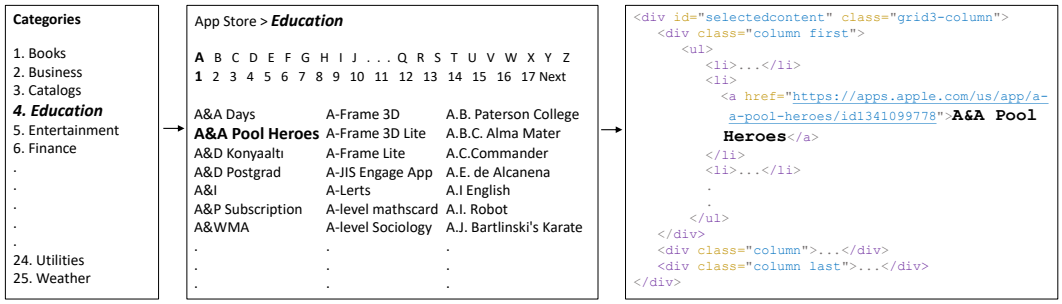


Fig. 2. Scraping individual app IDs from the HTML pages of each app category.

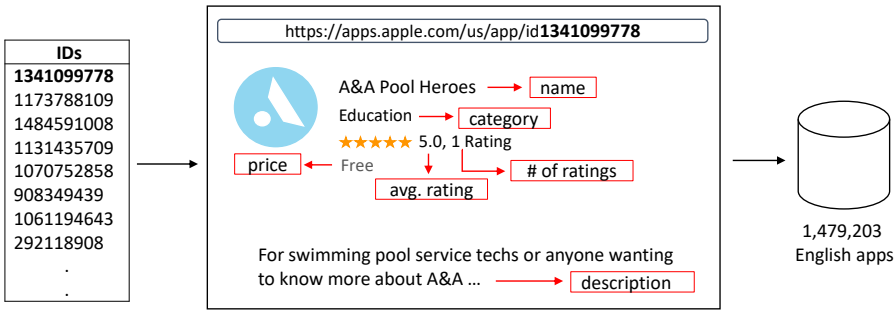


Fig. 3. Scraping app meta-data from each app's HTML page.

school news, and many other functional learning activities. Examples of popular apps under this category include, *Edmodo*, *Inkflow*, and *Remind: School Communication*.

- **Games**: The category of games includes any app that provides some sort of educational content in the form of a game. The Apple App Store labels these apps under both Games and Education categories. Puzzles, trivia, and brain training games are examples of such games. Examples of popular apps under this category include, *Toddler puzzle games for kids*, *MentalUP*, and *Math Ninja*.
- **Misfits**: This category includes apps that do not fit in any of the above categories. These apps are listed under more than one category of the Apple App Store and often have limited educational merit. Examples of popular apps under this category include, *Charades*, *IQ Test: The Intelligence Quiz*, and *Cat sounds effects*.

For our second oracle, we used the framework proposed by Yasini and Marchand [94] to classify health-related apps. In their framework, the use-cases of apps were extracted by a team of IT professionals and medical doctors. The authors then introduced 31 different use-cases which were then grouped into six major usage categories. These categories can be described as follows:

- **Consulting medical information references:** This category includes apps that provide guidelines, scientific popularization, health news, medical textbooks, and access to medical databases. Examples of apps under this category include, *Human Anatomy Atlas, Headspace,* and *Tasty*.

- **Educational tools:** This category includes apps that provide sample educational questions, serious gaming, and access to clinical cases. Examples of popular apps under this category include, *TEAS Mastery, Curiscope,* and *Fig. 1 - Medical Cases.*
- **Health related management:** These apps are used for locating health services, managing drug stocks, and interacting with health-related institutions (e.g., scheduling an appointment, ordering a drug, or connecting to an insurance account). Examples of popular apps under this category include, *GoodRx, FollowMyHealth,* and *myHP.*
- **Fulfilling a contextual need:** This category includes apps that collect and interpret medical and health data, check patient records, help diagnosing illnesses, and provide health-decision support. Examples of popular apps under this category include, *iThermonitor, MyFitnessPal,* and *Nike Run Club.*
- **Communicating and/or sharing information:** These apps provide communications platforms for patients, health professionals, and institutions. Examples of popular apps under this category include, *GAIN Trainer, Coach's Eye,* and *TigerText.*
- **Managing professional activities:** These apps help health professionals to calculate expenses and fees, manage their schedules, and search for jobs. Examples of popular apps under this category include, *Kareo, Medscape,* and *Amion.*

4.3 Ground-Truth

Our crawled dataset of apps data contains 138,095 apps from the Education category and a total of 81,944 health-related apps from the Health&Fitness and Medical categories of the Apple App Store. To create our ground-truth, we randomly sampled 300 educational apps and 300 health-related apps. Using the entire content of the app store as our population helped to mitigate the popular app sampling problem [57]. The sample of 300 apps for both datasets is representative at a 95% confidence level with 5.6512% confidence interval.

The descriptions of our sampled apps were then manually examined by three judges and then classified into the different categories identified in our oracle. This process can be described as follows:

- An initial meeting was held to discuss the task of the judges.
- Three judges, including two Ph.D. students and a Master's student in software engineering independently classified the apps.
- The manual classification process was carried out over three sessions, each session lasted around six hours, divided into two periods of three hours each to avoid any fatigue issues and to ensure the integrity of the data [93].
- The results were compiled and majority voting was then used to determine the final app categories.
- Conflicts (~5%) were resolved by referring to the original description of the different expert-generated categories as well as the app descriptions. In some cases, apps were installed to get a better sense of their actual functionalities.
- The final ground-truth was verified by a fourth judge, a professor of software engineering.

The classification process took place prior to conducting the research. Two of the first three judges were not aware of the purpose of the study. On average, the judges had an average of four years of experience in mobile app design and development. Tables 1 and Table 2 show the number of apps classified under each category and subcategory in each of our domains[5].

[5]A replication package is available at http://seel.cse.lsu.edu/data/tosem21.zip

Table 1. The number of apps classified under each category and subcategory of educational apps.

Category	Subcategories	Total
Skill-based	Literacy (38), Numeracy (22), Test-preparation (12), Subject-area (11)	83
Content-based	Subject-area (48), Reference (31)	79
Function-based	Learning-community (58), Others (8)	66
Games	Subject-area (19), Puzzle (14), Brain-training (12)	45
Misfit		27

Table 2. The number of apps classified under each category of health-related apps.

Category	Total
Consulting medical information references	72
Educational tools	23
Health-related management	80
Fulfilling a contextual need	93
Communicating and/or sharing information	26
Managing professional activities	6

5 APPROACH AND ANALYSIS

Our proposed approach (Fig. 4) can be broken down into three main steps: data pre-processing, vectorization, and classification. In what follows, we describe each of these steps in greater detail.

5.1 Pre-processing

Combinations of text pre-processing strategies are often used in text classification tasks to remove potential noise and to enhance the prediction capabilities of the classifier [41]. In our analysis, app descriptions were first converted into lower case tokens. Tokens that contained non-ASCII characters, digits, and URLs, were removed. English stop-words (e.g., *the, in, will*) were also removed based on the list of stop-words provided in NLTK [53]. The remaining words were then lemmatized. We selected lemmatization over stemming to preserve the naturalness of words. In particular, stemmers (e.g., Porter stemmer [71]) tend to be prone to over-stemming which happens when too much of the word is removed that the outcome is not a valid natural word (e.g., *general* and *generous* are stemmed to *gener*). This can be a key factor in the performance of methods that use English corpora for similarity calculations.

5.2 Vectorization

Under this step, we converted the list of pre-processed tokens in each app's description into a vector of word embeddings using the pre-trained models of Word2Vec, GloVe, and fastText. We then used the generated word embeddings to represent the whole description. Word collection (e.g., phrase, sentence, or paragraph) embeddings can be computed using operations on word vectors, such as their unweighted averaging [59], Smooth Inverse Frequency (SIF) [5], Doc2Vec [47, 79] and Recursive Neural Networks (RNNs) [84]. In our analysis, we used the simple unweighted averaging method to obtain an embedding for each app description. Averaging word vectors has been proven to be a strong baseline for paragraph representation especially in cases when the order of words in the text is unimportant [5, 42, 78]. Formally, the vector representation (V_D) of the app D, can be computed as:

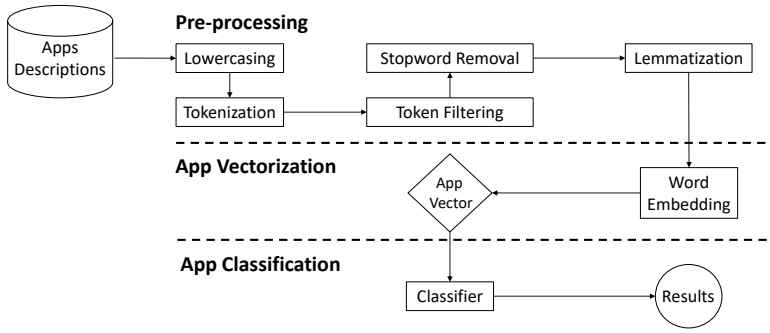


Fig. 4. The main steps of the proposed approach.

Table 3. An example of the text pre-processing steps used in our analysis.

Original sentence	*"dictionary® series is designed to make it easier to learn and use the technical jargon, and abbreviations"*	
Good tokens	[*dictionary, series, is, designed, to, make, it, easier, to, learn, and, use, the, technical, jargon, and, abbreviations"*]	
Stop-word removal	[*dictionary, series, designed, make, easier, learn, use, technical, jargon, abbreviations*]	
Porter stemmer	[*dictionari, seri, design, make, easi, learn, use, technic, jargon, abbrevi*]	
WordNet lemmatizer	[*dictionary, series, design, make, easy, learn, use, technical, jargon, abbreviation*]	
Word embedding	dictionary	[-0.84041, -0.11159, 0.49872, 0.30307, . . . , -0.08523, 0.80811, -0.12826, 0.088422]
	series	[2.2805e-01, 5.6135e-01, 1.5447e-01, . . ., 2.8141e-01, 2.7452e-01, 2.0327e-0]
	design	[0.28934, 0.026391, -0.5053, -0.9966, . . ., 0.44844, 0.063997, 0.56751, 0.059809]
	make	[.3547e-01, 1.1550e-01, -2.9983e-01, . . ., -3.9427e-01, -2.3503e-01, 3.0761e-01]
	easy	[6.7032e-02, -1.0813e-01, 4.4981e-01, . . ., -1.5073e-01, -2.5662e-01, 8.0550e-02]
	learn	[-3.3157e-01, -8.8796e-02, -1.6376e-01, . . ., 1.2686e-02, -8.1500e-02, 1.3113e-03]
	use	[-1.0515e-01, 1.3407e-01, 1.3839e-01, . . ., -4.1949e-01, 8.9402e-02, 1.7569e-01]
	technical	[0.32273 , -0.085892, -0.26033, -0.3997, . . ., -0.78574, -0.23319, -0.11036, -0.59291]
	jargon	[-0.54072, -0.27198, 0.34199, 0.17347, . . ., -0.46058, 0.30216, -0.266, 0.19508]
	abbreviation	[-0.1607 , 0.15862 , 0.77783 , -0.16962 , . . ., -0.12764 , -0.75467 , 0.34786 , -0.31688]
Sentence embedding		[-1.12256169e-01, -4.28820204e-04, 1.39360920e-01, . . ., 8.19473062e-03, -1.63306966e-02]

$$V_D = \frac{1}{n}\sum_{i=1}^{n} V_{w_i} \tag{2}$$

where the description is composed of words $w_0, w_1, ..., w_n$. Each word is represented as a vector $V_{w_0}, V_{w_1}, ..., V_{w_n}$ of word embeddings. Table 3 shows the different pre-processing steps and the vectorization step (using Word2Vec, GloVe, and fastText) being applied to the sample app description sentence, *" dictionary® series is designed to make it easier to learn and use the technical jargon, and abbreviations."*

5.3 Experimental Baselines

In addition to our word embedding vectors, we generate three other types of vectors for app descriptions. These vectors will be used as experimental baselines to compare the performance of word embeddings. These representations can be described as follows:

- **Vector Space Model:** VSM is an algebraic model that consists of a single term-document matrix. Each row of the matrix represents a single term found in the corpus and each column represents an individual document. Each entry in the matrix $w_{i,j}$ is the weight of the term j in the document i, indicating the importance of the term to the document's subject matter. While

the raw frequency of the term in the document can be used as a weight, another approach, known as term frequency-inverse document frequency (TF.IDF) is typically used [81]. TF.IDF is calculated as the product of the frequency of the term in the document (TF) and the term's scarcity across all the documents (IDF). Formally, TF.IDF can be computed as:

$$TF.IDF = f(w,d) \times \log \frac{|D|}{df(w)} \tag{3}$$

where $f(w,d)$ is the term frequency of the word w in document d, D is the total number of documents in the corpus, and $df(w)$ is the number of documents in the corpus D that contain the word w. In this paper, we vectorize each app's description as the TF.IDF of its words. In particular, each app is represented as a vector of size $|V|$, which is the number of words in the description (i.e. $V = \{w_0, w_1, ..., w_m\}$). The i^{th} entry of this vector is set to $TF.IDF(w_i)$ if the description contains the word w_i, and 0 otherwise.

- **Latent Dirichlet Allocation:** LDA is an unsupervised probabilistic approach for estimating a topic distribution over a text corpus [9]. A topic consists of a group of words that collectively represents a potential thematic concept [9, 36]. Formally, LDA assumes that words within documents are the observed data. The known parameters of the model include the number of topics k, and the Dirichlet priors on the topic-word and document-topic distributions β and α. Each topic t_i in the latent topic space ($t_i \in T$) is modeled as a multi-dimensional probability distribution, sampled from a Dirichlet distribution β, over the set of unique words ($w_i \in W$) in the corpus D, such that $\phi_{w|t} \sim Dirichlet(\beta)$. Similarly, each document from the collection ($d_i \in D$), is modeled as a probability distribution, sampled from a Dirichlet distribution α over the set of topics, such that $\theta_{t|d} \sim Dirichlet(\alpha)$. $\theta_{t|d}$ and $\phi_{w|t}$ are inferred using approximate inference techniques such as Gibbs sampling [26]. Gibbs sampling creates an initial, naturally weak, full assignment of words and documents to topics. The sampling process then iterates through each word in each document until word and topic assignments converge to an acceptable (stable) estimation [9].
  We use Gensim, a Python-based open-source toolkit for vector space modeling and topic modeling, to extract topics from our dataset of apps descriptions [74]. LDA's hyper-parameters α and β are calibrated based on the heuristics that are commonly used to calibrate topic modeling in text analysis [39, 62]. In particular, α is set to be automatically learned from the corpus and β is set to 1/(number of topics). The number of iterations for the sampling process is set to 1000 to ensure the stability of generated topics [26]. The number of topics to be found by LDA, k, is set to the number of classification labels. When applied to descriptions, LDA represents each description as a vector of size k. The i^{th} entry of this vector is set to the probability of the topic i to be present in the description.
- **BM25:** BM25 is a text scoring method that was introduced in 1994 as a robust variant of the TF.IDF method [76]. BM25 is calculated as the product of modifications of TF and IDF. Formally, BM25 can be computed as:

$$BM25(w_i) = \frac{f(w,d) \times (k+1)}{f(w,d) + k \times (1 - b + b \times \frac{|d|}{avgD})} \times log(1 + \frac{|D| - df(w) + 0.5}{df(w) + 0.5}) \tag{4}$$

where $f(w,d)$ is the term frequency of the word w in document d, D is the total number of documents in the corpus, and $df(w)$ is the number of documents in the corpus D that contain the word w, $avgD$ is the average length of the documents, b is a parameter to tune the impact of the length of the document on the score, and k is a tuning parameter to modify the impact of term frequency. BM25 has been used in the literature to vectorize app descriptions [7]. In particular, each app description is represented as a vector of size $|V|$, where $|V|$ is the number

of words in the description (i.e. $V = \{w_0, w_1, ..., w_m\}$). The i^{th} entry of this vector is set to $BM25(w_i)$ if the description contains the word w_i and 0 otherwise.

5.4 Classification and Evaluation

Under the third step, we classify apps in our dataset using the different vectorization techniques presented earlier. Our classification configurations can be described as follows:

- **Classification algorithms**: To classify our data, we experiment with multiple classification algorithms: Naive Bayes (NB) [46], Support Vector Machines (SVM) [41], Random Forests (RF) [50], Decision Trees (DT) [60], AdaBoost [21], and Logistic Regression [45]. These algorithms are commonly used to classify crowd feedback in the mobile app market [29, 55, 92]. Their success can be attributed to their ability to deal with short texts (e.g., tweets, user reviews, YouTube comments, etc.) [34, 70, 91]. Our analysis is performed using Scikit-learn, a Python library which integrates a wide range of state-of-the-art machine learning algorithms for supervised and unsupervised classification problems [67].
  We use a one-vs-one strategy for our multi-class classification. This classification strategy splits a multi-class classification into one binary classification problem per pair of classes. The class that receives the majority of votes is selected as the predicted class. Hyperparameter tuning is used to ensure that each classifier achieves its best possible prediction given the data [23]. Our specific list of hyperparameters is shown in Table 4. We use RandomizedSearchCV, a methodology which uses cross-validation to optimize the hyperparameters of the classifier. A detailed explanation of each parameter can be found on Scikit-learn's webpage[6].
- **Classification features**: We extract app classification features from their descriptions. Each app's description is vectorized using the vectorization techniques described in Section 5.2. For each app description, word embedding methods generate a d-dimensional feature vector. The size of the vector for Word2Vec and fastText is set by default to ($d = 300$). GloVe, can generate different size vectors, where ($d \in \{50, 100, 200, 300\}$), VSM generates vectors of size $|V|$, where $|V|$ is the number of words in the description (i.e. $V = \{w_0, w_1, ..., w_m\}$). Using LDA, each app is vectorized into a feature vector of size k representing the probabilistic distribution of the description over the set of k LDA topics. We further analyze the impact of adding existing meta-data features (i.e. the number of ratings, average rating, app size, category, and price) on the classification accuracy. Therefore, for each vectorization technique, we append the extracted meta-data to the vectorized representation of each app.
- **Training settings**: To train and test our classifiers, we use *10*-fold cross-validation. This approach creates 10 partitions of the dataset. In each partition, 90% of the instances are considered as the training set and 10% as the test set. 10-fold cross-validation is selected over other techniques, such as the holdout method (e.g. train/test split), to decrease the variance of the results.
- **Validation metrics**: The standard measures of precision (P), recall (R), and F-measure (F_β) are used to evaluate the performance of our classification algorithms. These measures are computed independently for each classification label and averaged over all the labels. Precision is calculated as the ratio of the number of correctly classified instances under a specific label (t_p) to the total number of classified instances under the same label ($t_p + f_p$). Recall is calculated as the ratio of t_p to the total number of instances belonging to that label ($t_p + f_n$). The F-measure represents the weighted harmonic mean of precision and recall. β is used to emphasize precision or recall. A $\beta = 2$ is commonly used in related literature to slightly emphasize recall over precision. Formally, f_β can be calculated as $(\beta^2 + 1)PR/(\beta^2 P + R)$.

[6]https://scikit-learn.org/stable/modules/grid_search.html

Table 4. Hyperparameter configuration for each classifier.

Classifier	Parameter set
Naive Bayes	Gaussian NB: *'var_smoothing'* ∈ $\{10^{-10}, 10^{1}\}$
	Multinomial NB: *'alpha'* ∈ {0, 1}
Adaboost	*'n_estimators'* ∈ {10, 50, 100, 200}
Random Forest	*'max_depth'* ∈ {10, 30, 50, 100, None}
	'max_features' ∈ {'auto', 'sqrt'}
	'min_samples_split' ∈ {2, 5, 10}
	'n_estimators' ∈ {10, 50, 100, 200}
KNN	*'k'* ∈ {2, 5, 7, 10}
SVM	*'kernel'* ∈ {'linear', 'rbf'}
	'C' ∈ {0.1, 1, 10, 100}
	'gamma' ∈ {1/(n_features * X.var()), 1/n_features}
Decision Trees	*'max_depth'* ∈ {10, 30, 50, 100, None}
	'max_features' ∈ {'auto', 'sqrt'}
	'min_samples_split' ∈ {2, 5, 10}
	'criterion' ∈ {'qini', 'entropy'}
Logistic Regression	*'penalty '* ∈ {'l1', 'l2'}
	'C' ∈ {0.1, 1, 10, 100}

5.5 Results and Analysis

The results of classifying our sets of Education and Health apps are shown in Table 5. On average, our classification algorithms achieved their best performance when app descriptions were vectorized using GloVe_{300}. In particular, SVM (linear kernel) was able to achieve the best results in separating the general categories of education apps, achieving an F_2 of 0.84. Logistic Regression achieved a comparable performance (F_2 = 0.8) . However, the accuracy went down when we classified education apps at a subcategory level. This was actually expected given that it becomes harder to separate categories at such a granularity level.

In the health dataset, SVM was also able to achieve the best results (F_2 = 0.8). Logistic Regression (F_2 = 0.78) and KNN (F_2 = 0.78) were able to achieve comparable performance. However, the accuracy was on average lower than the accuracy achieved on the education dataset. A comparison of the performance based on the different size vectors generated by GloVe is shown in Fig. 5. In general, for both datasets, GloVe achieved its best results at vector size 300. Increasing the size of vectors resulted in more expressive vectors that capture all word relations.

Our results also showed that adding app meta-data to the set of classification features did not improve the performance. As Fig. 6 shows, apps meta-data failed to enhance the predictive capabilities of our classifiers. To get a better sense of our results, we compared our findings with Berardi et al. [7]. In their work, the authors considered app descriptions as well as apps' meta-data (rating, size, category, and price) as classification features. App descriptions were preprocessed using tokenization, stop-word removal, and stemming and then vectorized using BM25 [75]. We replicated this type of analysis on our dataset. Following Berardi et al. [7], a mutual information-based feature selection method was also applied to select the most informative set of app features [19]. The results showed that adding meta-data as classification features did not improve the results. This was actually expected given that, unlike descriptions, meta-data attributes of apps (apps' names, ratings, downloads, etc.) hardly convey any functionality-related information.

Table 5. The performance (Precision (P), Recall (R), F-measure (F_2)) of the different classification algorithms using the different proposed app description vectorization techniques.

Approach	Classifier	Education categories			Education sub_categories			Health apps		
		P	**R**	F_2	**P**	**R**	F_2	**P**	**R**	F_2
VSM	NB	0.53	0.56	0.55	0.28	0.35	0.33	0.5	0.57	0.55
	AdaBoost	0.37	0.39	0.38	0.28	0.25	0.25	0.33	0.42	0.39
	Random Forest	0.67	0.65	0.65	0.46	0.45	0.45	0.55	0.59	0.58
	KNN	0.65	0.62	0.62	0.56	0.51	0.51	0.64	0.6	0.6
	SVM	0.71	0.69	0.69	0.57	0.5	0.51	0.64	0.66	0.65
	Decision Trees	0.56	0.51	0.51	0.44	0.38	0.39	0.52	0.51	0.51
	Logistic Regression	0.68	0.67	0.67	0.44	0.45	0.44	0.56	0.61	0.59
	Average	0.59	0.58	0.58	0.43	0.41	0.41	0.53	0.56	0.55
LDA	NB	0.27	0.32	0.3	0.15	0.12	0.12	0.41	0.27	0.28
	AdaBoost	0.42	0.35	0.36	0.14	0.2	0.18	0.32	0.29	0.29
	Random Forest	0.49	0.41	0.42	0.29	0.28	0.28	0.33	0.31	0.31
	KNN	0.38	0.35	0.35	0.26	0.25	0.25	0.34	0.28	0.29
	SVM	0.35	0.41	0.39	0.2	0.29	0.26	0.33	0.38	0.36
	Decision Trees	0.44	0.38	0.39	0.29	0.25	0.25	0.3	0.27	0.27
	Logistic Regression	0.35	0.4	0.38	0.23	0.31	0.28	0.33	0.39	0.37
	Average	0.38	0.37	0.37	0.22	0.24	0.23	0.33	0.31	0.31
GloVe 300	NB	0.7	0.68	0.68	**0.67**	**0.63**	**0.63**	0.72	0.66	0.67
	AdaBoost	0.69	0.67	0.67	0.4	0.38	0.38	0.54	0.55	0.54
	Random Forest	0.73	0.71	0.71	0.56	0.53	0.53	0.7	0.71	0.7
	KNN	0.74	0.72	0.72	0.66	0.58	0.59	0.81	0.78	0.78
	SVM	**0.85**	**0.84**	**0.84**	0.6	0.57	0.57	**0.83**	**0.8**	**0.8**
	Decision Trees	0.63	0.59	0.59	0.46	0.44	0.44	0.6	0.58	0.58
	Logistic Regression	0.82	0.8	0.8	0.57	0.54	0.54	0.79	0.78	0.78
	Average	0.73	0.71	0.71	0.56	0.52	0.53	0.71	0.69	0.69
Word2Vec	NB	0.74	0.69	0.69	0.62	0.52	0.53	0.68	0.63	0.63
	AdaBoost	0.55	0.51	0.51	0.21	0.22	0.21	0.4	0.37	0.37
	Random Forest	0.67	0.68	0.67	0.52	0.48	0.48	0.62	0.65	0.64
	KNN	0.64	0.64	0.64	0.53	0.48	0.48	0.64	0.58	0.59
	SVM	0.67	0.68	0.67	0.43	0.45	0.44	0.59	0.64	0.62
	Decision Trees	0.47	0.45	0.45	0.34	0.28	0.29	0.5	0.48	0.48
	Logistic Regression	0.66	0.66	0.66	0.41	0.45	0.44	0.54	0.62	0.6
	Average	0.62	0.61	0.61	0.43	0.41	0.41	0.56	0.56	0.56
fastText	NB	0.68	0.64	0.64	0.6	0.6	0.6	0.72	0.65	0.66
	AdaBoost	0.63	0.57	0.58	0.16	0.21	0.19	0.52	0.6	0.58
	Random Forest	0.69	0.7	0.69	0.57	0.53	0.53	0.66	0.69	0.68
	KNN	0.71	0.68	0.68	0.62	0.57	0.57	0.83	0.76	0.77
	SVM	0.79	0.78	0.78	0.59	0.53	0.54	0.8	0.79	0.79
	Decision Trees	0.61	0.57	0.57	0.45	0.41	0.41	0.5	0.47	0.47
	Logistic Regression	0.78	0.77	0.77	0.49	0.48	0.48	0.65	0.7	0.68
	Average	0.69	0.67	0.67	0.49	0.47	0.47	0.66	0.66	0.66
BM25	NB	0.48	0.33	0.35	0.4	0.32	0.33	0.54	0.39	0.41
	AdaBoost	0.42	0.37	0.37	0.26	0.25	0.25	0.37	0.37	0.37
	Random Forest	0.58	0.57	0.57	0.45	0.39	0.4	0.49	0.56	0.54
	KNN	0.55	0.46	0.47	0.48	0.39	0.4	0.54	0.45	0.46
	SVM	0.53	0.47	0.48	0.51	0.42	0.43	0.52	0.48	0.48
	Decision Trees	0.51	0.47	0.47	0.38	0.33	0.33	0.42	0.41	0.41
	Logistic Regression	0.58	0.58	0.58	0.55	0.45	0.46	0.54	0.55	0.54
	Average	0.52	0.46	0.47	0.43	0.36	0.37	0.48	0.45	0.46

5.6 Statistical Analysis

We use statistical testing to measure the difference in performance between our proposed approach and other experimental baselines. We first used the Shapiro-Wilk test to test the normality of the

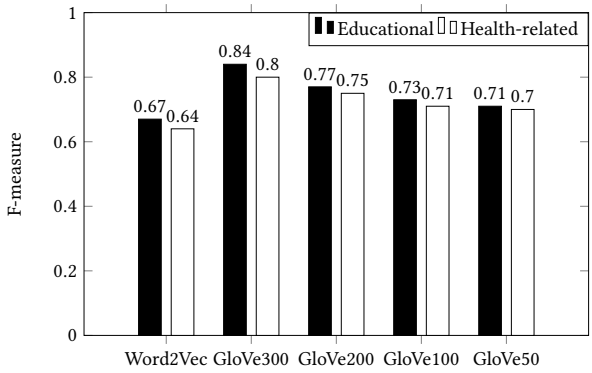


Fig. 5. SVM classification results using different GloVe size vectors.

variable (f-measure) being compared in our analysis [83]. The results showed that the normality assumption did not hold for the majority of our comparisons. Therefore, for our type of data, we used the non-parametric tests Wilcoxon signed-rank and Friedman to measure statistical significance (p-$value$ = 0.05).

To examine the effect of using different vectorization methods (GloVe, Word2Vec, fastText, LDA, TF.IDF, and BM25) on the accuracy of our classifiers, we first applied the Friedman hypothesis test with Bonferroni-Holm correction (control method = GloVe) at p-$value$ = 0.05 [22, 38]. Our null hypothesis H_0 states that there is no difference in the f-measures between different vectorization methods. The alternative hypothesis H_1 is in favor of a significant difference between our different methods. To show the effect size of the difference, we used Kendall's W (coefficient of concordance). Kendall's uses the Cohen's interpretation guidelines of 0.1 (small effect), 0.3 (moderate effect), and above 0.5 as a strong effect [13]. The results showed a p-$value < 0.005$, indicating that we can reject H_0 with at least a medium effect size (Kendall's W = 0.47) between the classification results of GloVe and other vectorization techniques. Given these results, we conclude that GloVe leads to statistically significant improvement in comparison to other techniques, including the baseline using app meta-data as classification features.

We further examine the effect of considering apps meta-data as classification features. In particular, we compare the f-measure values of SVMs+GloVe and SVMs+GloVe+metadata features, applying 10-fold cross validation in both cases. Our null hypothesis H_0 states that adding apps metadata does not have any effect on the f-measure. The alternative hypothesis H_1 is in favor of the effect of adding apps metadata. Since in this test we have two groups of data, we use Wilcoxon signed-rank at p-$value < 0.05$ to measure statistical significance. To show the effect size of the difference between applied methods, we calculate Cliff's Delta (d), a non-parametric effect size method. We interpret the effect size values as *small* for $0.147 < d < 0.33$, *medium* for $0.33 < d < 0.474$, and *large* for $d > 0.474$ [27, 85]. Our results show that there is a statistically significant difference (p-$value < 0.05$) with at least a medium (d = 0.07) effect size when considering apps metadata as classification features, indicating that adding apps metadata can significantly degrade the accuracy of classification.

6 VALIDATION AND HUMAN EXPERIMENT

In the first phase of our analysis, we showed that word embeddings of mobile app descriptions can be used to classify apps into more focused categories of application domains. To further validate our findings, in this section, we apply our classification procedure to a third dataset of mobile

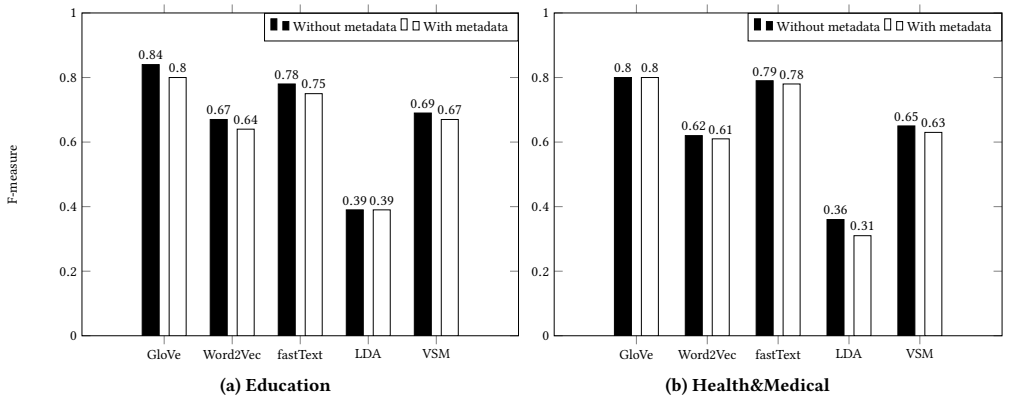


Fig. 6. The results of classifying educational health-related apps using SVMs with/without adding apps metadata (i.e. the number of ratings, the average rating, the app size, the category, and the price.

apps, sampled from the domain of Sharing Economy (SE). The Sharing (also known as *Shared* or *Gig*) Economy refers to a sustainable form of Peer-to-Peer (P2P) business exchange that is built around sharing assets and resources [56]. Over the past decade, Sharing Economy apps, such as Uber, TaskRabbit, and Airbnb, have caused major disturbances in established classical markets, enabling people to exchange and monetize their underused (or idle) assets at an unprecedented scale [15, 16, 73, 77]. As of today, there are thousands of active Sharing Economy apps, operating in a market sector that is projected to grow to close to 335 billion U.S. dollars by 2025 [72].

The domain of Sharing Economy presents a prime example of application domains where a more precise categorization of apps is highly needed. In particular, popular app stores, such as the Apple App Store and Google Play, do not provide a separate category for Sharing Economy apps, instead, these apps are scattered over a broad range of categories that hardly capture their core functionalities. For example, both Uber (ride-sharing) and Airbnb (lodging) are categorized under the Travel category in the App Store and Gigwalk, a freelancing app, is classified under the Lifestyle category. This makes it very challenging for service providers and receivers to navigate the landscape of SE apps and make optimized economic decisions in one of the fastest growing software ecosystems in the world.

In this section, we use our classification approach to classify a dataset of popular Sharing Economy apps based on their core functionalities. We then conduct a human experiment, using 12 study participants, to validate our classifier from an end-user point of view.

6.1 Dataset and Expert Classification

To conduct our analysis, we sample a dataset of Sharing Economy apps that are currently active in the market. We enforce the following criteria on apps to be included in our sample:

(1) The app must facilitate some sort of a P2P connection and include the sharing of some sort of a resource, such as an asset (e.g., an apartment, car, electric drill, etc.) or a skill (e.g., plumbing, hair styling, coding, etc.).
(2) The app must be available on Google Play or the Apple App Store so that we can extract its description.
(3) The app must be located and/or have a substantial presence in the U.S. By focusing on the U.S. market, we ensure that app descriptions are available in English and that the app supports a service that is familiar to the casual U.S. user.

With these criteria in place, we searched for apps to be included in our dataset. We conducted a Google search using the query: `(sharing OR shared OR gig) AND economy AND (platforms OR apps OR systems)`. We examined the first 10 pages of the search results and added 72 new platforms that matched our inclusion criteria. We then used the *similar* feature on Google Play and the Apple App Store to locate any apps we missed through Google search. Specifically, we examined the list of similar apps resulting from searching app stores for each of our 72 apps. Lightweight snowballing was then used to add any major apps that we might have missed. Apps were iteratively added until no more new apps that satisfied our inclusion criteria were located. In total, 108 unique apps were included in our dataset. Descriptive statistics of our dataset are provided in Table 6.

To generate our expert-based categories, we went through each app in our sample and independently examined their Apple App Store descriptions. We used memoing to keep track of the reasoning behind each suggested category. Axial coding was then used to consolidate individual categories into more abstract categories. For example, the categories of *food delivery* and *grocery delivery* were merged into a single *delivery* category. Generated categories were then iteratively revised until no more categories were found. By the end of our classification process, six main categories of Sharing Economy apps, shown in Fig. 7, had emerged. These categories can be described as follows:

- **Skill-based:** These apps facilitate the sharing of personal skills (hiring labor). Specific examples include the baby sitting apps *Sittercity* and *Urbansitter*, the tutoring apps *Verbling*, *Codementor*, and *Classgap*, and the freelancing apps *Fiverr* and *Upwork*.
- **Delivery:** Under this category, we include apps which enable users to utilize their vehicles to deliver goods to other users. Examples of apps in this category include *UberEats*, *Grubhub*, and *Shipt* for grocery and food delivery and *DriveMatch*, *uShip*, and *Dolly* for hiring delivery drivers.
- **Ride-sharing:** This category includes apps which allow their users to share rides, such as carpooling and driver/rider connections. Examples of apps in this category include traditional ride-sharing services, such *Uber*, *Lyft*, and *Via*, as well as more specialized platforms, such as *HopSkipDriver* for children transportation, *Veyo* for medical transportation, and *Wingz* for hiring a driver.
- **Asset-sharing:** Under this category, we include any app which enables users to lend and borrow assets. This category is different from other categories in the sense that the resource being shared is the asset itself (e.g., a vehicle or an electric drill), not the person (e.g., a driver or electrician). Examples of apps under this category include the boat sharing apps *Get-MyBoat* and *Boatsetter*, the bike sharing app *Spinlister*, and the RV sharing apps *RVezy* and *Outdoorsy*.
- **Lodging:** This category contains renting and short-term accommodation services such as *Airbnb*, *Vrbo*, and *Misterbnb* as well as space-sharing for storage (*Neighbor*), events and work (*Splacer* and *LiquidSpace*), and even parking (*ParqEx*).
- **Other:** Although our objective was to classify all apps into the main general categories, two apps in our dataset were too niche-oriented to warrant the creation of a separate category. These apps are *Prosper* for lending and borrowing money and *Kickstarter*, a platform for crowdfunding various projects.

6.2 Automated Classification

To classify our apps, we extracted their descriptions from the Apple App Store. Descriptions were then processed by applying tokenization, removing non-ASCII characters and URLs, stop-word

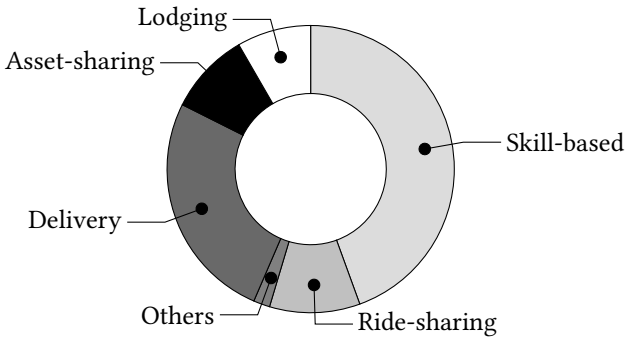


Fig. 7. Distribution of our apps over application domains.

Table 6. Descriptive statistics for the 108 apps in our dataset.

Metric	Mean	Median	Min	Max
App Store Rating	4.23	4.60	1.60	4.90
Google Play Rating	3.86	3.90	2.00	4.90
App Store # of Reviews	201K	2.4K	2	8.9M
Google Play # of Reviews	134K	1.3K	7	7.91M
Google Play # of Installs	6.9M	100K	1K	500M

removal, and lemmatization (Sec. 5.1). We then vectorized the description words using the pre-trained model of GloVe (vector size = 300). An SVMs classifier (linear kernel) was then trained on the data using 10-fold cross-validation. The results showed that our proposed model (SVMs+GloVe) achieved an F_2 of 0.8 (Precision = 0.84 and Recall = 0.79). Overall, these results came out consistent with our previous results on the education and health app datasets.

A closer look at our results revealed that our automated classifier failed to correctly label apps that describe their features using words that are common in more than one category. For example, `ParqEx` is a parking sharing app that was misclassified as a delivery app. `ParqEx` description contains words such as *space*, *parking*, and *book* which are common in both delivery and lodging apps. Another observation is that two of the skill-based apps were misclassified as lodging apps. A possible explanation is that our dataset included only seven lodging apps, which were not enough data for the classifier to generate a separate category for these apps. Our expectation is that such errors will be minimized as more data becomes available for our classifier to work with.

6.3 Study Participants and Procedure

To further evaluate the effectiveness of our automated classifier, we conducted a human experiment with 12 study participants. Our participants were recruited through convenience sampling. Our sample has four females and eight males with an average age of 36 (min = 21, max = 48). All participants reported using one or more Sharing Economy apps, either as service providers (e.g., driving for Uber, Lyft, DoorDash and Instacart) or receivers (e.g., renting an Airbnb, ordering groceries through Instacart, and hiring workers through TaskRabbit). Choosing subjects who are average end-users of apps, rather than expert software developers, provides an evidence of the value of our approach to the casual users. Our experimental procedure can be described as follow:

- We randomly sampled 20 apps from our dataset using stratified sampling. We excluded popular apps such as Uber, Lyft, Airbnb, TaskRabbit, and UpWork as classifying these apps can be trivial.
- We prepared an assignment for our study participants. The assignment included a description of our expert-generated categories of Sharing Economy apps as well as the descriptions of the 20 apps in our sample.
- Each of our 12 study participants was presented with the assignment and asked to classify each app by picking a category from the expert-based classification. If they thought that none of our suggested categories was a good fit for the app, they were advised to either classify the app as *others*, or add their own category. Apps were listed in a random order in each assignment. Randomization helped to control any effect that might result from the order of the treatment, such as, our participants getting bored or tired and not spending as much time on classifying apps that appear later in the list.
- The human generated classifications are then compared with our automatically generated classifications and the results are then collected and analyzed.

6.4 Results

The results of our study are shown in Table 7. The table includes the list of apps used in our study, their ground truth classifications, their classifications by our approach, their current categories in the Apple App Store, and our 12 study participants classification. The results show that, our study participants achieved an average 79% agreement with our automated classification results. Cases of disagreement were detected over apps which were misclassified by our approach. For instance, our study participants had a hard time finding the right category for *Carvertize*, an app which pays drivers by placing advertisements on their cars. This app should be classified as an asset-sharing app, however, it was incorrectly labeled as skill-based by our classifier. This can be attributed to the fact that this particular app describes its features using words such as *media*, *student*, *college*, and *specialize*, missing common words in the asset-sharing category, such as *car* and *lend*. This vague terminology confused our classifier as well as most of our participants.

To evaluate the agreement between study participants, we used Cohen's kappa [12]. Cohen's kappa coefficient is commonly used in the literature to measure the inter-rater agreement for categorical data [17, 69]. This method is known as a more robust method than simple percent agreement since it takes into account the agreements occurring by chance [66]. Formally, Cohen's kappa (k) can be calculated as:

$$k = \frac{p_o - p_c}{1 - p_c} \tag{5}$$

where p_o is the proportion of items for which the raters agreed on, and p_c is the proportion of items that agreement was expected by chance. Our results of human assessment suggest a Cohen's kappa of **0.83**, which indicates an almost perfect agreement between our study participants. Cohen's kappa values less than zero are interpreted as no agreement, 0.01 – 0.20 as none to slight, 0.21 – 0.40 as fair, 0.41 – 0.60 as moderate, 0.61 – 0.80 as substantial, and 0.81 – 1.00 an as almost perfect agreement between raters [12]. Overall, our human assessment shows that our automated classification approach can generate accurate classifications that correlate with human generated classifications to a large extent.

7 DISCUSSION

In this section, we discuss our main analysis results and we provide further analysis on some of our findings in this paper.

Table 7. The results of our human study. We use the following code for the different categories: 1 = Skill-based, 2 = Delivery, 3 = Ride-sharing, 4 = Asset-sharing, 5 = Lodging, and 6 = Other. G.T. is our ground-truth classification, Auto is the automated classification, $\{S_1, \ldots, S_{12}\}$ is the set of subjects.

App	Category	G.T.	Auto.	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
Rvezy	Travel	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Carvertise	Productivity	4	1	2	3	3	4	4	4	3	3	3	3	4	4
SparkDriver	Business	2	2	2	2	2	2	2	2	2	2	2	2	2	2
BiteSquad	Food&Drink	2	2	2	2	2	2	2	2	2	2	2	2	2	2
PointPickup	Utilities	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Shipt	Business	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Pickup	Lifestyle	2	1	2	2	1	1	4	2	2	2	2	2	2	1
ParqEx	Navigation	5	2	5	5	5	5	5	5	4	5	5	4	5	4
Couchsurfing	Travel	5	5	5	5	5	5	5	5	5	4	5	4	5	5
Wingz	Travel	3	3	3	3	3	3	3	1	3	3	3	3	3	3
zTrip	Travel	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Eatwith	Travel	1	5	1	5	5	1	1	1	1	1	1	1	1	1
Withlocals	Travel	1	5	1	1	1	1	1	1	1	1	1	1	1	1
Gigwalk	Lifestyle	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Handy	Lifestyle	1	1	1	1	1	1	1	1	1	1	1	1	1	1
GigSmart	Business	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Bambino	Lifestyle	1	1	1	1	1	1	1	1	1	1	1	1	1	1
UrbanSitter	Lifestyle	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Caregiver	Lifestyle	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Fiverr	Business	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Agreement with automated				0.8	0.85	0.8	0.75	0.75	0.75	0.8	0.75	0.8	0.75	0.8	0.9

7.1 Word Embeddings vs. Topic Modeling

We begin our discussion by comparing the performance of our approach with LDA, one of the most commonly used approaches in app classification tasks [61, 64, 89]. Our results show that word embeddings models were more successful in identifying correct app categories than topic models (LDA). This can be explained based on the observation that the topics (Table 8) generated for our data were of poor quality. In other words, they failed to capture any of the expert-generated categories. For example, while the second topic generated for our educational apps included words such as *game* and *fun*, it failed to represent a coherent category due to the presence of important words from other categories, such as *learn* and *child*. Topics generated for the set of Health apps seem to be more cohesive. For example, Topic 5 includes words such as *workout*, *exercise*, *weight*, and *fit* which are indicative of a fitness app. Similarly, Topic 6 includes words such as *day*, *medical*, *track*, and *time* which indicate a patient management app. However, both topics also share words with other less cohesive topics, such as Topic 1 and Topic 3, leading the classifier to make inaccurate predictions. In other words, due to the overlapping nature of the different topic categories, the classes are not separable by LDA.

The poor results of LDA can be partially explained based on the limited length of app descriptions [40]. Prior evidence has shown that LDA does not perform well when the input documents are short in length [8, 39]. Specifically, LDA is a data-intensive technique that requires large quantities of text to generate meaningful topic distributions. However, due to the limited nature of description text, applying standard LDA to such data often produces incoherent and overlapping topics [39]. One instance of LDA misclassification in our dataset is the app of the International Journal of Psychology (IJP). This app keeps track of the research in the field of psychology. The description of this app includes statements such as:

– *Stay current with the latest articles through early view*
– *Receive alerts when new issues are available (opt in)*
– *Save your favorite articles for quick and easy access, including offline*
– *Dynamic references show references in context*
– *Share article abstract and link via email*
– *Access your personal or institutional subscription to IJP on your ipad*

IJP connects users to a database of articles related to psychology. This apps perfectly fits into the content-based app category. IJP's description contains words such as *journal*, *article*, *reference*, *research*, *report*, and *study*. These words are semantically related to the groups of words that convey the concept of a database of information. The word embeddings models used in our analysis correctly classified this app. LDA, in contrast, misclassified this app as a function-based app. This happened due to the presence of words such as *access*, *include*, and *use* in IJP's description. These words led to classifying the app to Topic 5 which is mostly related to function-based apps.

Table 8. Topics generated by LDA for our dataset of Education and Health apps.

Dataset	Topics	Most probable words
Education	Topic 1	learn, word, play, child, help, game, letter, lesson, fun, use
	Topic 2	math, game, level, time, word, kid, child, puzzle, fun, use
	Topic 3	color, kid, book, child, game, school, story, learn, feature, fun
	Topic 4	dictionary, english, use, question, access, period, record, exam, free, time
	Topic 5	school, inform, use, feature, student, english, access, news, include, event
Health&Medical	Topic 1	view, class, schedule, download, today, time, inform, contact, location, exam
	Topic 2	inform, use, injury, patient, provide, help, assess, view, detail, product
	Topic 3	workout, exercise, weight, account, purchase, fit, period, program, hour, renew
	Topic 4	patient, meditation, health, help, inform, day, education, treatment, track, time
	Topic 5	calculate, workout, health, exercise, nutrition, rate, calorie, food, heart, pain
	Topic 6	medicine, patient, doctor, care, medical, prescript, time, help, use, access

Word embedding models tend to be immune to the limitations of topic modeling methods. For instance, Fig. 8-a and b show a 2D projection of the GloVe_{300} embeddings (vectors) of a collection of words sampled from the descriptions of our education and health apps. The projection shows that related words tend to appear in separate clusters in the 2D space. For example, words such as *Italian, Portuguese, Spanish*, and *French* which are indicative of foreign languages appeared in a single well-defined cluster (closer in the space). The same applies to the words *crosswords, sudoku*, and *puzzle*, which are indicative of educational games. Similar patterns of related word clusters can also be observed in the Health category, where words such as *diabetes, hypertension*, and *obesity* appeared near each other in the vector space. This kind of representation provided sufficient information for our classifiers to make accurate predictions.

7.2 Word Embeddings vs. Bag-of-Words

Our results also show that bag-of-words methods such as VSM and BM25 can be heavily disadvantaged by the vocabulary mismatch problem of app descriptions. In particular, both BM25 and VSM vectorize app descriptions based on the *TF.IDF* scores of their individual words. In comparison, word embeddings capture the semantic meaning of words during vectorization. In app description

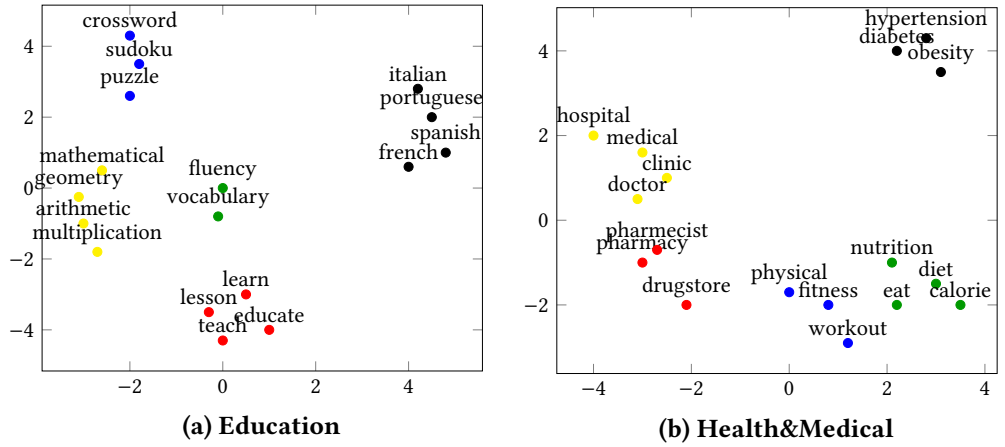


Fig. 8. Example word vectors represented in a 2D space generated by applying PCA on 300-dimensional GloVe word embeddings.

classification tasks, the semantic meaning of description words generates more information than their TF.IDF scores. For instance, apps that connect patients with their doctors use different vocabulary to convey this functionality, such as *"connect patients with doctors"* and *"link patients with physician"*. According to BM25 and VSM, the similarity of these two sentences are 8%, and 20%, respectively. In comparison, GloVe captures the similarity of these two sentences with a confidence level of 86% as the vectors of *connect* and *link*, and *doctor* and *physician* appear very close in the space.

7.3 GloVe vs. Word2Vec and fastText

Our results show that GloVe has outperformed Word2Vec and fastText (Table 5). In general, context-free word embeddings models, such as Glove, Word2Vec, and fastText are known to achieve comparable results. However, their performance can slightly vary depending on the intrinsic complexity of the text corpus [58, 63, 68, 95]. A potential reason for GloVe's better performance is the fact that the vocabulary used to train the model was more comprehensive than the vocabulary used to train the Word2Vec model. In particular, the pre-trained Word2Vec model used in our analysis did not include 1,422 words of the words that appeared in the descriptions of our apps, while 822 words (496 unique words) were missing from the pre-trained GloVe model. Furthermore, the majority of missing vocabulary in GloVe included insignificant words, such as apps names (i.e. *Accelastudy, Fortville, Kidomy, Vuga, etc.*), typos (i.e. *againsttheclock, comapany, jjust, uesd. etc.*), compound names (i.e. *cross-contamination, x-rays, custom-made, cutting-edge*), and unknown English words (i.e. *woao, yorinks, dixio, cassanea*). As for fastText, the embeddings generated for rare words (character n-grams) did not help the classification accuracy as such words were highly uncommon in our dataset.

7.4 Pre-trained vs. Locally trained models

In addition to pre-trained word embeddings, we used the Gensim library in Python to train our word embedding models on our corpus of 1,479,203 app descriptions (the entire population of mobile apps collected from the Apple App Store). The corpus was initially preprocessed by removing non-ASCII

characters and URLs. We then trained our different word embeddings models (Word2Vec, fastText, and GloVe) on the corpus. These models were then used to vectorize apps in our dataset.

As Fig. 9 shows, for all three embedding models (Word2Vec, fastText, and GloVe), the pre-trained models outperformed the models trained on our corpus. The poor performance of these models can be related to the relatively small size of the corpus. In order to achieve the most semantically meaningful vector representations of words, word embeddings models need a rich corpus where common English words are significantly more common than rare words. Our corpus of app descriptions contains numerous rare words that are not common English words, thus, resulting in low-quality semantic representations (embeddings) of words. In particular, in our dataset of mobile apps descriptions, 55% of the words appeared less than four times in the corpus.

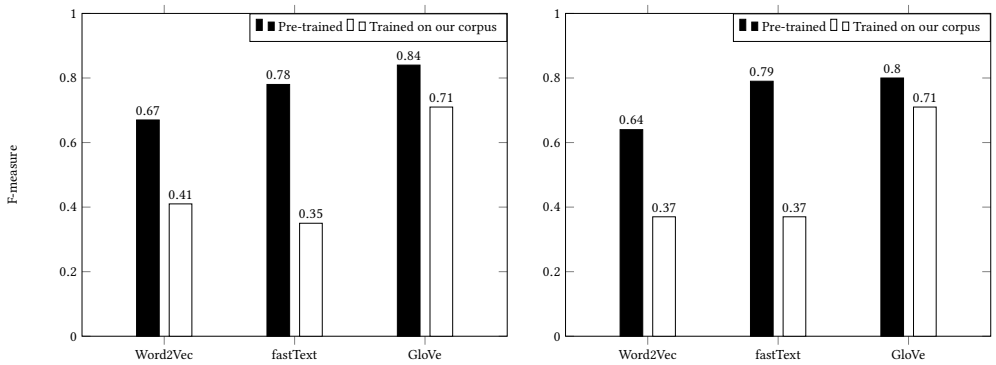


Fig. 9. The results of classifying educational health-related apps using SVMs using two different word embeddings: pre-trained and trained on our own corpus of apps descriptions

7.5 Time Analysis

In terms of running time, word embedding models typically require more time than other techniques such as LDA and VSM in order to classify mobile apps. Table 9 shows the execution time of different vectorization methods when classifying educational apps. Time was measured on an Intel(R) Core(TM) i7-7500U CPU 2.7 GHz, with 12.0GB of RAM. On average, VSM is the fastest method since it only requires the calculation of TF.IDF values of words. BM25 requires slightly more time since feature selection is applied for each training set. Word embeddings are on average slower in generating the classification results since loading a model can be a time-consuming task. However, once the embedding model is loaded, feature extraction and app classification require approximately the same amount of time as other methods to be completed.

Table 9. The average running time (in seconds) of the different vectorization methods.

Approach	Load Model	Extract Features	Classification	Total
Baseline	-	6.92	19.64	26.56
VSM	-	4.41	1.26	5.67
LDA	13.76	0.31	0.42	14.49
Word2Vec	113.27	206.77	1.9	321.94
GloVe	154.39	23.48	1.9	179.77
fastText	111.88	4.65	1.9	118.43

8 THREATS TO VALIDITY

The study presented in this paper has several limitations that could potentially limit the validity of the results. In what follows, we discuss these threats along with our mitigation strategies.

8.1 Internal Validity

Internal validity refers to confounding factors that might affect the causal relations established throughout the experiment [93]. A potential threat to the proposed study's internal validity might stem from the fact that human judgment was used to classify our apps and create our ground-truth. Different judges might classify the data differently, which might impact the results of our automated classifiers. Despite these subjectivity concerns, it is not uncommon in text classification tasks to use humans' judgment to prepare the ground-truth. Therefore, these threats are inevitable; however, they can be partially mitigated. For instance, in our analysis, this threat was mitigated by using multiple judges and majority voting and by utilizing pre-existing oracles that were proposed by domain experts [11, 94]. Similarly, a threat may arise from the fact that the categories for our validation set of Sharing Economy apps were generated by the authors. Nonetheless, the authors have published multiple papers on the Sharing Economy [87, 88, 92] and received multiple federal funding grants to develop accessible Sharing Economy solutions for their local community, thus, they can be considered as domain experts in the field.

Other internal validity issues might arise from the specific word-embedding methods, classification algorithms, and open source tools (Scikit-learn) used in our analysis. For example, we used GloVe, Word2Vec, and fastText to generate our word-embeddings. Other techniques, such as BERT, and other types of classification algorithms, such as Hierarchical Agglomerative Clustering (HAC), might arrive at different results.

8.2 External Validity

Threats to external validity impact the generalizability of the results obtained in the study [93]. In particular, the results of our experiment might not generalize beyond the specific experimental settings used in this paper. External validity concerns might be raised about the fact that only 600 apps sampled from two application domains were considered in our analysis, thus, the results of our empirical investigation might not generalize to other apps or domains. To mitigate this threat, we uniformly sampled our apps from the collection of all the educational and health-related apps in the Apple App Store [57]. This helped us to mitigate sampling problems and increase the confidence in our results. To further enhance the generalizability of the results, we validated our classification model on a third dataset sampled from the domain of Sharing Economy. The results came out aligned with our results on other datasets, providing evidence on the applicability of our approach to other application domains.

8.3 Construct Validity

Construct validity is the degree to which the various performance measures accurately capture the concepts they intend to measure. In our experiment, there were minimal threats to construct validity as the standard performance measures (recall, precision, and the F-measure), which are extensively used in related research, were used to assess the performance of our different investigated methods. We believe that these measures sufficiently captured and quantified the different aspects of performance we were interested in.

8.4 Conclusion Validity

Conclusion validity is concerned with issues that might affect the ability to draw the right conclusion about the relations between the treatment and the outcomes of the experiment [93]. To control for such threats, our data were tested for normality prior to our analysis and appropriate non-parametric statistical tests were then used to measure the difference in performance among the different treatments (classification settings). Overall, we were able to reject our null hypotheses with high statistical power. Furthermore, in our human experiment, our subject sample included 12 participants of female and male subjects (age between 21 - 54) and with various levels of experience in the Sharing Economy as service providers and receivers. We applied randomization whenever possible to minimize any confounding effects.

9 CONCLUSION AND FUTURE WORK

In this paper, we proposed a new approach for classifying mobile apps based on their app store descriptions. Our approach utilized models of word embeddings to generate numeric semantic representations of app descriptions. These vector representations were then classified to produce more cohesive categories of mobile apps. The performance of our approach was evaluated using a dataset of apps sampled from the Education, Health&Fitness, and Medical categories of the Apple App Store. Expert-generated categorizations of these apps were used to produce our ground-truth. Our results showed that word embeddings produced using the pre-trained GloVe_{300} led to higher quality categorization than embeddings generated using Word2Vec and fastText. Our results also showed that word embeddings were able to outperform other vectorization techniques such as bag-of-words (VSM and BM25) and topic modeling (LDA) and other baselines which considered app meta-data attributes as classification features [7]. To further validate our results, we applied our GolVe classification model on a third dataset of Sharing Economy apps. The results showed that our model was able to achieve accuracy levels comparable to the accuracy achieved on the first two datasets. We further ran a study with 12 participants to assess the quality of our classifier. The results showed that our study participants classified our apps with a high degree of agreement with our approach.

Our work in this paper is expected to help users discover apps that match their specific interests more effectively. Developers can also use our approach to identify their direct competition in the app store. In terms of future work, our analysis in this paper can be extended along three main directions:

- **More data:** More analysis, utilizing more expert-generated categorizations of apps across a broad range of application domains will be conducted. Our objective is to determine a global set of configuration settings that can be used to dynamically generate more accessible categorizations of apps.
- **Tool support:** A working prototype will be developed to implement our findings. The prototype will be ideally implemented in a mobile app with a user-friendly interface to aid mobile app users in finding apps that meet their specific needs.
- **Extrinsic evaluation:** Our evaluation in this paper was mainly intrinsic, based on how well the generated categories correlated with existing classifications. While such an evaluation can be sufficient for model assessment, it does not capture the practical significance of the results. Therefore, a main direction of future work will be dedicated to extrinsic evaluation. Extrinsic evaluation is concerned with criteria relating to the system's function, or role, in relation to its purpose (e.g., validation through experience). To conduct such analysis, our prototype will be provided to selected groups of stakeholders, such as health professionals, educators, and app developers to be used as an integral part of their app search and development activities.

Evaluation data will be collected through surveys that will measure the approach's usability, scalability, and overall value to users.

ACKNOWLEDGMENT

This work was supported in part by the U.S. National Science Foundation (Award CNS 1951411) and LSU Economic Development Assistantships awards.

REFERENCES

[1] 2019. Mobile app usage. https://www.statista.com/topics/1002/mobile-app-usage/.
[2] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019. Docbert: Bert for document classification. *arXiv preprint arXiv:1904.08398* (2019).
[3] Afnan Al-Subaihin, Federica Sarro, Sue Black, and Licia Capra. 2019. Empirical comparison of text-based mobile apps similarity measurement techniques. In *Empirical Software Engineering*. 1–26.
[4] Afnan Al-Subaihin, Federica Sarro, Sue Black, Licia Capra, Mark Harman, Yue Jia, and Yuanyuan Zhang. 2016. Clustering mobile apps based on mined textual features. In *International Symposium on Empirical Software Engineering and Measurement*. 1–38.
[5] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2016. A simple but tough-to-beat baseline for sentence embeddings. In *International Conference on Learning Representations*.
[6] Mohaddeseh Bastan, Mahnaz Koupaee, Youngseo Son, Richard Sicoli, and Niranjan Balasubramanian. 2020. Author's Sentiment Prediction. *arXiv preprint arXiv:2011.06128* (2020).
[7] Giacomo Berardi, Andrea Esuli, Tiziano Fagni, and Fabrizio Sebastiani. 2015. Multi-store metadata-based supervised mobile app classification. In *Annual ACM Symposium on Applied Computing*. 585–588.
[8] 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.
[9] David Blei, Andrew Ng, and Michael Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022.
[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.
[11] Todd Cherner, Judy Dix, and Corey Lee. 2014. Cleaning up that mess: A framework for classifying educational apps. *Contemporary Issues in Technology and Teacher Education* 14, 2 (2014), 158–193.
[12] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46.
[13] Anthony Conger. 1980. Integration and generalization of kappas for multiple raters. *Psychological Bulletin* 88, 2 (1980), 322.
[14] Laura Dennison, Leanne Morrison, Gemma Conway, and Lucy Yardley. 2013. Opportunities and Challenges for Smartphone Applications in Supporting Health Behavior Change: Qualitative Study. *Journal of Medical Internet Research* 15, 4 (2013), e86.
[15] Tawanna Dillahunt, Xinyi Wang, Earnest Wheeler, Hao Cheng, Brent Hecht, and Haiyi Zhu. 2017. The Sharing Economy in Computing: A Systematic Literature Review. *Proceedings of the ACM Human Computer Interaction* 1 (2017), 26.
[16] Tarik Dogru, Makarand Mody, and Courtney Suess. 2019. Adding evidence to the debate: Quantifying Airbnb's disruptive impact on ten key hotel markets. *Tourism Management* 72 (2019), 27–39.
[17] Tore Dybå and Torgeir Dingsøyr. 2008. Strength of evidence in systematic reviews in software engineering. In *International Symposium on Empirical Software Engineering and Measurement*. 178–187.
[18] Anthony Finkelstein, Mark Harman, Yue Jia, William Martin, Federica Sarro, and Yuanyuan Zhang. 2017. Investigating the relationship between price, rating, and popularity in the Blackberry World App Store. *Information and Software Technology* 87 (2017), 119–139.
[19] George Forman. 2004. A pitfall and solution in multi-class feature selection for text classification. In *International Conference on Machine Learning*. 38–46.
[20] Maha Fraj, Mohamed Hajkacem, and Nadia Essoussi. 2018. A novel tweets clustering method using word embeddings. In *International Conference on Computer Systems and Applications*. 1–7.
[21] Yoav Freund and Robert Schapire. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*. 23–37.
[22] Milton Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Amer. Statist. Assoc.* 32, 200 (1937), 675–701.

[23] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135–146.
[24] Cuiyun Gao, Jichuan Zeng, Zhiyuan Wen, David Lo, Xin Xia, Irwin King, and Michael Lyu. 2020. Emerging App Issue Identification via Online Joint Sentiment-Topic Tracing. *arXiv preprint arXiv:2008.09976* (2020).
[25] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking app behavior against app descriptions. In *International Conference on Software Engineering*. 1025–1035.
[26] Thomas Griffiths and Mark Steyvers. 2004. Finding scientific topics. *National Academy of Sciences* 101, 1 (2004), 5228–5235.
[27] Robert Grissom and John Kim. 2005. *Effect sizes for research: A broad practical approach.* Lawrence Erlbaum Associates Publishers.
[28] Luis Gutiérrez and Brian Keith. 2018. A Systematic Literature Review on Word Embeddings. In *International Conference on Software Process Improvement*. 132–141.
[29] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. 2015. Ensemble Methods for App Review Classification: An Approach for Software Evolution. In *International Conference on Automated Software Engineering*. 771–776.
[30] Monte Hancock. 2016. *Data Mining: Supervised Learning*. Taylor & Francis Group, 406–421.
[31] Zellig Harris. 1954. Distributional structure. *Word* 10, 2-3 (1954), 146–162.
[32] Maryam Heidari and James Jones. 2020. Using bert to extract topic-independent sentiment features for social media bot detection. In *Annual Ubiquitous Computing, Electronics & Mobile Communication Conference*. 0542–0547.
[33] Maryam Heidari, James Jones, and Ozlem Uzuner. 2020. Deep Contextualized Word Embedding for Text-based Online User Profiling to Detect Social Bots on Twitter. In *International Conference on Data Mining Workshops*.
[34] Maryam Heidari, James Jones, and Ozlem Uzuner. 2021. An Empirical Study of Machine learning Algorithms for Social Media Bot Detection. In *International IOT, Electronics and Mechatronics Conference*. 1–5.
[35] Maryam Heidari and Setareh Rafatirad. 2020. Using Transfer Learning Approach to Implement Convolutional Neural Network to Recommend Airline Tickets by Using Online Reviews. In *International Workshop on Semantic and Social Media Adaptation and Personalization*.
[36] Thomas Hofmann. 2017. Probabilistic latent semantic indexing. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 211–218.
[37] Matthias Höhn, Ute Jan, Theodor Framke, and Urs-Vito Albrecht. 2016. Classification of Health Related Applications. *Studies in health technology and informatics* 266 (2016), 139–142.
[38] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.
[39] Liangjie Hong and Brian Davison. 2010. Empirical Study of Topic Modeling in Twitter. In *Workshop on Social Media Analytics*. 80–88.
[40] He Jiang, Hongjing Ma, Zhilei Ren, Jingxuan Zhang, and Xiaochen Li. 2014. What makes a good app description?. In *Asia-Pacific Symposium on Internetware*. 45–53.
[41] Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning*. 137–142.
[42] Tom Kenter, Alexey Borisov, and Maarten Rijke. 2016. Siamese cbow: Optimizing word embeddings for sentence representations. *arXiv preprint arXiv:1606.04640* (2016).
[43] Moniba Keymanesh, Micha Elsner, and Srinivasan Parthasarathy. 2020. Toward Domain-Guided Controllable Summarization of Privacy Policies. In *Natural Legal Language Processing Workshop at KDD*.
[44] Moniba Keymanesh, Saket Gurukar, Bethany Boettner, Christopher Browning, Catherine Calder, and Srinivasan Parthasarathy. 2020. Twitter Watch: Leveraging Social Media to Monitor and Predict Collective-Efficacy of Neighborhoods. In *Complex Networks*. 197–211.
[45] David Kleinbaum and Mitchel Klein. 2002. *Logistic regression.* Springer.
[46] Pat Langley, Wayne Iba, and Kevin Thompson. 1992. An Analysis of Bayesian Classifiers. In *Artificial Intelligence*. 223–228.
[47] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. 1188–1196.
[48] Gunwood Lee and Raghu Santanam. 2014. Determinants of mobile apps' success: Evidence from the app store market. *Journal of Management Information Systems* 31, 2 (2014), 133–170.
[49] Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Annual Meeting of the Association for Computational Linguistics*. 302–308.
[50] Andy Liaw and Matthew Wiener. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.
[51] Soo Lim and Peter Bentley. 2013. Investigating app store ranking algorithms using a simulation of mobile app ecosystems. In *Congress on Evolutionary Computation*. 2672–2679.

[52] Soo Lim, Peter Bentley, Natalie Kanakam, Fuyuki Ishikawa, and Shinichi Honiden. 2015. Investigating Country Differences in Mobile App User Behavior and Challenges for Software Engineering. *IEEE Transactions on Software Engineering* 41, 1 (2015), 40–64.
[53] Edward Loper and Steven Bird. 2002. NLTK: The natural language toolkit. In *COLING/ACL on Interactive Presentation Sessions*. 69–72.
[54] David Lulu and Tsvi Kuflik. 2013. Functionality-based clustering using short textual description: Helping users to find apps installed on their mobile device. In *International Conference on Intelligent User Interfaces*. 297–306.
[55] Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *Requirements Engineering Conference*. 116–125.
[56] Chris Martin. 2016. The sharing economy: A pathway to sustainability or a nightmarish form of neoliberal capitalism? *Ecological Economics* 121 (2016), 149–159.
[57] William Martin, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang. 2015. The app sampling problem for app store mining. In *Conference on Mining Software Repositories*. 123–133.
[58] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Workshop of 1st International Conference on Learning Representations*.
[59] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.
[60] Tom Mitchell. 1997. *Machine learning*. McGraw-hill New York.
[61] Shahab Mokarizadeh, Mohammad Rahman, and Mihhail Matskin. 2013. Mining and Analysis of Apps in Google Play. In *International Conference on Web Information Systems and Technologies*. 527–535.
[62] Ece Mutlu, Toktam Oghaz, Ege Tutunculer, and Ivan Garibay. 2020. Do Bots Have Moral Judgement? The Difference Between Bots and Humans in Moral Rhetoric. In *International Conference on Advances in Social Networks Analysis and Mining*.
[63] Marwa Naili, Anja Chaibi, and Henda Hajjami. 2017. Comparative study of word embedding methods in topic segmentation. *Procedia Computer Science* 112 (2017), 340–349.
[64] Maleknaz Nayebi, Homayoon Farrahi, Ada Lee, Henry Cho, and Guenther Ruhe. 2016. More insight from being more focused: Analysis of clustered market apps. In *International Workshop on App Market Analytics*. 30–36.
[65] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. 2013. {WHYPER}: Towards automating risk assessment of mobile applications. In {*USENIX*} *Security Symposium*. 527–542.
[66] Chang Park and Hyun Kim. 2015. Measurement of inter-rater reliability in systematic review. *Hanyang Medical Reviews* 35, 1 (2015), 44–49.
[67] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[68] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*. 1532–1543.
[69] Jorge Pérez, Jessica Díaz, Javier Garcia-Martin, and Bernardo Tabuenca. 2020. Systematic literature reviews in software engineering—Enhancement of the study selection process using Cohen's kappa statistic. *Journal of Systems and Software* 168 (2020).
[70] 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.
[71] Martin Porter. 1980. An algorithm for suffix stripping. 14, 3 (1980), 130–137.
[72] PwC. 2015. The Sharing Economy: Consumer Intelligence Series. *PricewaterhouseCoopers LLP* (2015).
[73] Giovanni Quattrone, Davide Proserpio, Daniele Quercia, Licia Capra, and Mirco Musolesi. 2016. Who Benefits from the "Sharing" Economy of Airbnb?. In *International Conference on World Wide Web*. 1385–1394.
[74] Radim Rehurek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. In *Workshop on New Challenges for NLP Frameworks*.
[75] Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation* (2004).
[76] Stephen Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *International Conference on Research and Development in Information Retrieval*. 345–354.
[77] Amin Sabzehzar, Yili Hong, and Raghu Santanam. 2020. People Don't Change, Their Priorities Do: Evidence of Value Homophily for Disaster Relief. (2020).
[78] Amir Sadeghian and Alireza Sharafat. 2015. Bag of words meets bags of popcorn. (2015).
[79] Hani Safadi, Weifeng Li, Pouya Rahmati, Saber Soleymani, Krzysztof Kochut, and Amit Sheth. 2020. Curtailing Fake News Propagation with Psychographics. *SSRN Electronic Journal* (2020).

[80] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, and Pablo Bringas. 2012. On the automatic categorisation of Android applications. In *Consumer Communications and Networking Conference*. 149–153.
[81] Hinrich Schütze, Christopher Manning, and Prabhakar Raghavan. 2008. Introduction to information retrieval. In *International Communication of Association for Computing Machinery Conference*.
[82] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. 2010. Automated static code analysis for classifying Android applications using machine learning. In *International Conference on Computational Intelligence and Security*. 329–333.
[83] Samuel Shapiro and Martin Wilk. 1965. An analysis of variance test for normality (complete samples). *Biometrika* 52, 3 (1965), 591–611.
[84] Richard Socher, Eric Huang, Jeffrey Pennin, Christopher Manning, and Andrew Ng. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*. 801–809.
[85] Yuan Tian, Meiyappan Nagappan, David Lo, and Ahmed Hassan. 2015. What are the characteristics of high-rated apps? A case study on free android applications. In *International Conference on Software Maintenance and Evolution*. 301–310.
[86] John Torous, Jennifer Nicholas, Mark E Larsen, Joseph Firth, and Helen Christensen. 2018. Clinical review of user engagement with mental health smartphone apps: evidence, theory and improvements. *Evidence-Based Mental Health* 21, 3 (2018), 116–119.
[87] Miroslav Tushev, Fahimeh Ebrahimi, and Anas Mahmoud. 2020. Digital Discrimination in Sharing Economy A Requirements Engineering Perspective. In *International Requirements Engineering Conference*. 204–214.
[88] Miroslav Tushev, Fahimeh Ebrahimi, and Anas Mahmoud. 2021. Analysis of Non-Discrimination Policies in Sharing Economy. In *International Conference on Software Maintenance and Evolution*.
[89] Svitlana Vakulenko, Oliver Müller, and Jan Brocke. 2014. Enriching iTunes App Store categories via topic modeling. In *International Conference on Information Systems*. 1–11.
[90] Sahar Voghoei, James Byars, Khaled Rasheed, and Hamid Arabnia. 2021. Decoding the Alphabet Soup of Degrees in the United States Postsecondary Education System Through Hybrid Method: Database and Text Mining. *Advances in Data Science and Information Engineering* (2021).
[91] Sida Wang and Christopher Manning. 2012. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. In *Annual Meeting of the Association for Computational Linguistics*. 90–94.
[92] 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 (2020), 229—-263.
[93] Claes Wohlin, Per Runeson, Martin Höst, Magnus Ohlsson, Björn Regnell, and Anders Wesslèn. 2012. *Experimentation in Software Engineering*. Springer.
[94] Mobin Yasini and Guillaume Marchand. 2015. Toward a use case based classification of mobile health applications. In *Studies in health technology and informatics*. 175–179.
[95] Liang Yu, Jin Wang, Robert Lai, and Xuejie Zhang. 2017. Refining word embeddings for sentiment analysis. In *Conference on Empirical Methods in Natural Language Processing*. 534–539.
[96] Hengshu Zhu, Huanhuan Cao, Enhong Chen, Hui Xiong, and Jilei Tian. 2012. Exploiting enriched contextual information for mobile app classification. In *International Conference on Information and Knowledge Management*. 1617–1621.
[97] Hengshu Zhu, Enhong Chen, Hui Xiong, Huanhuan Cao, and Jilei Tian. 2013. Mobile app classification with enriched contextual information. *IEEE Transactions on Mobile Computing* 13, 7 (2013), 1550–1563.