Enhancing the Search Tool of the Android Settings through Natural Language Processing

Luiz Ricardo Takeshi Horita, Múcio Donizetti Paixão Júnior, João Batista Pereira Matos Junior
Sidia Institute of Science and Technology
{horita.l, mucio.j, j.matos}@samsung.com

***Abstract*—*Smartphones* have become essential to daily life, providing much more than communication services. Continuously, the device is getting "smarter" and more complex with additional features and sensors. Configuring those features may not be so easy for new users, and making the Settings app easy to use is challenging. With this in mind, a search tool was indexed on its initial screen. However, it is still not efficient enough. While most Android search tools will try to match exactly the queried words, a more intuitive tool capable of finding content related to the meaning of those words would be desirably better. In this paper, we propose a solution based on *word2vec* model to encode the context of each screen in order to get more robust and intuitive search approach on the Android Settings application. Although the search problem has not been entirely solved yet, experiments showed satisfactory results, which include resolving more than 82% of cases that cannot be handled by the search tool embedded to the Android Settings.**

***Index Terms*—search approach, smartphone application, user interest hierarchy, natural language processing**

I. Introduction

It is notable the tremendous increase on the number of *smartphone* adopters all over the world in the last decade. Initially, the term *smartphone* was only a marketing strategy to refer a brand new class of mobile phones that provided integrated services, ranging from several kinds of communication (voice call, messaging, emailing), computing, personal information management applications, and wireless connections [1]. However, in the last few years the meaning of *smart* has been pushed further with advances on artificial intelligence and creation of new features for mobile applications. At the same time devices get more complex, they must be easy to use for greater user experience (UX), which also implies making them more inclusive [2].

The operating system (OS) of a *smartphone* provides a computer-like platform that allows installing applications (commonly named as apps) of any type, like calculator, calendar, web browser, etc. However, it is in the embedded app **Settings** that the user can control and configure the functionality and behavior of the device or any other apps installed on the device. For example, one could configure the permissions of an app to access mobile data or sensors, change the contents and presentation of the user interface (UI), set connectivity or the device's behavior according to sensor inputs, etc. To make it easier to an inexperienced user to configure the device, a search tool has been indexed to the Settings application [3]. Then, instead of exploring the Settings menu to find the desired feature to be configured, one can type keywords that describe the feature directly into the search tool.

Nonetheless, we have noted that this search tool has some limitations due to its simplicity. It is observable that the search tool on the Android Settings works simply as a look-up table. Thus, having a database stored in a local file or in a virtual table, the search tool only checks the existence of the searched terms on this database. Because of this, no contextual information is considered, making it work well only if the right keywords are used.

Aiming to overcome this problem, we propose the use of continuous bag-of-words (CBOW) to provide contextual information during the search. CBOW is one of the *word2vec* flavors [4], that is capable of encoding the contextual information defined by a set of words. So, assuming the Settings structure is organized by features of the device's functionalities, and that each branch of this menu tree has its own context, then we can encode the context of every accessible screens by taking their textual information and applying CBOW. Thus, the search tool will not be a simple string search but a screen context search.

The advantage of searching the context instead of the string is its robustness. Since the context is encoded as a vector $\mathbf{v} \in \mathbb{R}^n$, it becomes independent to the existence of the queried words in the database. This is because words with similar meaning are encoded close to each other on the CBOW space.

Currently, the two most popular *smartphone* OSs are Android and iOS [5]. In this paper, however, we have focused on the Android, which represents today more than 76% of the worldwide mobile OS market share [5]. Experimental results showed that our method can find around 90% of the Android Settings screens, including 82% of those not found by the current embedded search tool.

II. Related Works

The search tool in the Android Settings app closely resembles other search tools in other domains like web search and Desktop search. For instance, web search aims to provide a systematic approach for users to access indexed information from internet, while desktop search allows indexing information from several files of different formats [6]. Similarly, it seems Settings search index information of screens inside the Settings app. Although the application domain may change, search tools are essential to improve user experience in the user-computer-interaction. For this reason, researchers have

been applying great effort on making the search results more personalized, more efficient, and more meaningful to users, as will be discussed below.

Some works propose a heuristic to apply a genetic algorithm to select search engines that are more likely to contain useful information given a search query in a meta-search engine problem, where the search tool has to combine results from various search engines, therefore selecting the most relevant search engines can improve its efficiency [7]. On the other hand, others apply neural networks combined with weighted round robin to find the most relevant search engines [8]. It was shown that the round robin approach can improve the cases where a document frequently occurs in the selecting procedure while the neural network can improve the evaluation of the similarity between a current queries and past queries.

Besides these search approaches, one could also use techniques related to information retrieval and text classification [9], [10]. An important contribution for this and for many other application of natural language processing was the *word2vec*, which was proposed by [4]. It is a neural network language model that can be modeled as a continuous bag-of-words (CBOW) or as a continuous skip-gram. CBOW is faster to train than the continuous skip-gram, and works slightly better on frequent words. On the other hand, continuous skip-gram has advantage over CBOW on small amount of training data, and represents rare words or phrases better.

In [11] information retrieval techniques are combined with *word2vec* to improve the search results for API code examples. In this research, the main problem of existing techniques was found to be lexical mismatch, so machine learning was used to relate API documentation to API code in order to achieve better code retrieval. In this way, it enabled the usage of more abstract or not so intuitive queries. For example, one may query ”insert an element into an array at a given position” expecting the answer to be the example of a code usage for ”List.add”. Beside lexical mismatch, search engines do not have good performance when presented to broad or ambiguous queries especially in the absence of context. In order to overcome this problem, [12] propose an approach based on *word2vec* model to enrich snippet searches by expanding a word with the top-n most semantic similar words. Snippets are short texts and, in most cases, words do not appear more than once in the same snippet. The textual content of each snippet is derived in a set of terms that are later enriched by a vocabulary learned by the *word2vec* model.

III. Android Search Tool

According to the Android Developer Guide [3], Android has a Search framework as one of its core user features. It allows users to find any data that is available to them, once this content is located on a database on the device or on Internet. This framework is used on the Settings search tool, but it is also available to Android developers so they can use it on their application [3]. The search algorithm however, can be implemented in a specific Android activity, which given the search queries and the database, returns the search results.

Although the search algorithm used on Settings is not available in a document or as source code, the graphic interface itself shows how it works. For illustration, some examples were obtained directly from an Android device (Figure 1), where it is possible to observe that the words queried for search are explicitly highlighted in blue letters on the results, confirming that the search algorithm basically looks for strings that contain the queried words. However, some singularities can also be noted. First, the queried words are considered independently during the search, which returns, in certain level, all the results that contain any of those words, even if it is a result without any relation to what has been searched, as shown in Figure 1a. Second, in some cases, a single additional word can prevent the search tool to find something, as in Figure 1b. Third, it seems that in some cases synonym words are considered during the search, as in Figure 1c, but in others it simply does not work, as in Figure 1d.

Besides the above observations, probably there might be some other particularities that could not be pointed out. However, the important thing is that the current search tool in the Android Settings uses a deterministic search approach. The advantage of such is its fast response due to simplicity, but the disadvantage is that it might make the search tool not intuitive and less inclusive. For example, blind users may want to use voice search to configure their devices. But, if they do not know exactly the right keywords to be used in the search tool, they might not get the desired results.

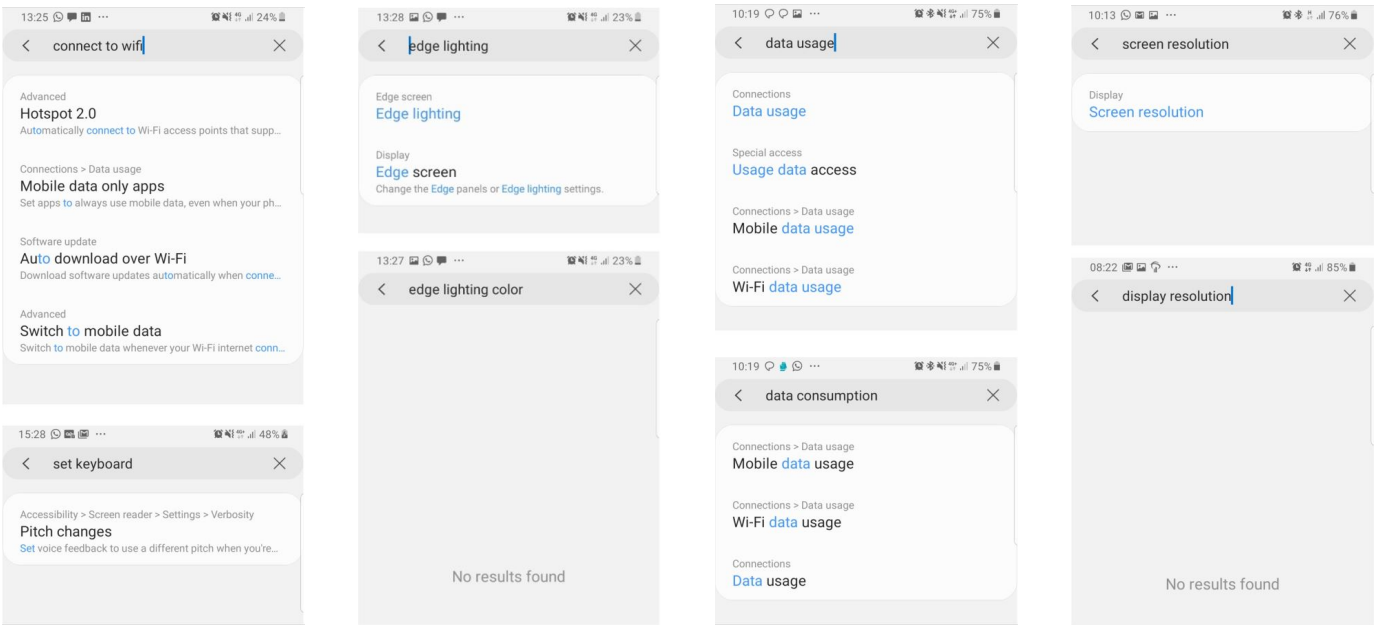
IV. The Proposed Method

To overcome the limitations present in the search tool of the Android Settings, we propose a method based on *word2vec* [4] that leverages the probabilistic distribution of words to encode the context of the screen they are inserted in. The following sections explains some background concepts and how they are useful to our method.

A. Continuous Bag of Words (CBOW)

One of the most simple language model is the bag-of-words (BOW), which was firstly mentioned as a hypothesis back in 1954 by [13]. This hypothesis stated that two different words in similar context have similar meanings. Since then, a paradigm for acquiring representations that could capture semantic and syntactic similarities between words has emerged. This method is very straightforward, consisting in creating a histogram of word occurrences to represent the context defined by a set of finite words. Therefore, two words that are inserted in similar contexts may have approximately the same words occurrences, and will probably have the same meaning.

In this model each word is represented by a hot-encoded vector, which makes it costly for large vocabularies. Aiming at a simpler model to minimize computational complexity, [4] proposed CBOW, that has exactly the same principle as BOW. However, instead of representing words using sparse vectors, it uses continuous and dense word vectors. It might cause some loss of information, but the gain in data efficiency is higher, which makes it an advantageous trade-off. In CBOW,



(a) Literal search. (b) Keywords limitation. (c) Successful synonyms. (d) Unsuccessful synonyms.

Fig. 1: Examples of some results from the search tool in Settings.

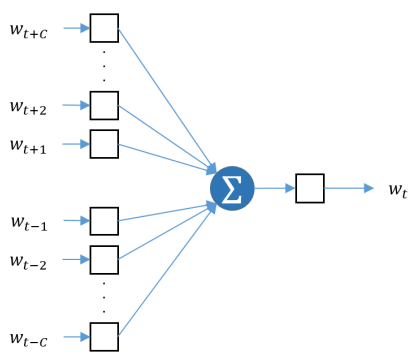


Fig. 2: CBOW model predicts w_t based on the context.

the context is given by a set of C words before and C words after the word we want to predict, as illustrated in Figure 2.

To make dense word vectors possible, [4] adopted the language model based on neural network. In this model, the network parameters themselves are used to define the word vectors, which were called *word2vec*. For this reason, the dense word vectors must be trained on a text dataset, big enough to generalize all relationships between words and contexts. Once it is done, two different words but with similar meaning may have dense word vectors close to each other.

This property is important to our method, since we need a way to make a screen search by its context. The next section describes how to encode the context of a screen by its textual content.

B. User Interface Dump Embedding

As it is described in [14], a screen in an Android application is formed by building blocks called *Views* that are responsible for drawing and event handling. The *Views* are used to create interactive UI components like buttons, check boxes and text views. The screen is organized as a hierarchy of blocks where *ViewGroups* represents invisible containers that hold together *Views* and other *ViewGroups*. Among all classes of *Views*, the one that interest us the most is the *TextView*, which contains strings that are shown on the UI. The hierarchy of a screen can be easily extracted from the UI dump, and can be represented as a tree. Figure 3 shows a simplification of the blocks hierarchy in the "Connections" screen, showing only the *TextView* blocks or the *ViewGroups* that contain at least one *TextView*.

Knowing this, we can introduce the UI dump embedding, which we denominate *dump2vec*. This embedding must encode the whole context of each screen as a feature vector. To do so, CBOW in form of *word2vec* is adopted to encode each word present on the screen. Then, each node of the UI hierarchy associated to a *TextView* is encoded by taking the average feature vector $\mathbf{v}_{node}$ as follows:

$$\mathbf{v}_{node} = \frac{\sum_{j=1}^{N} \mathbf{v}_j}{N} \tag{1}$$

where $\mathbf{v}_j$ is the *word2vec* of the jth word in the current *TextView* node, and N is its number of words.

Finally, a weighted average of all encoded nodes is done, such that nodes nearer to the hierarchy root get higher importance. Thus, the screen context is encoded by computing the following equation:

$$\mathbf{v}_{dump} = \frac{\sum_{i=1}^{D} \alpha^{d_i} \cdot \mathbf{v}_i}{\sum_{i=1}^{D} \alpha^{d_i}} \tag{2}$$

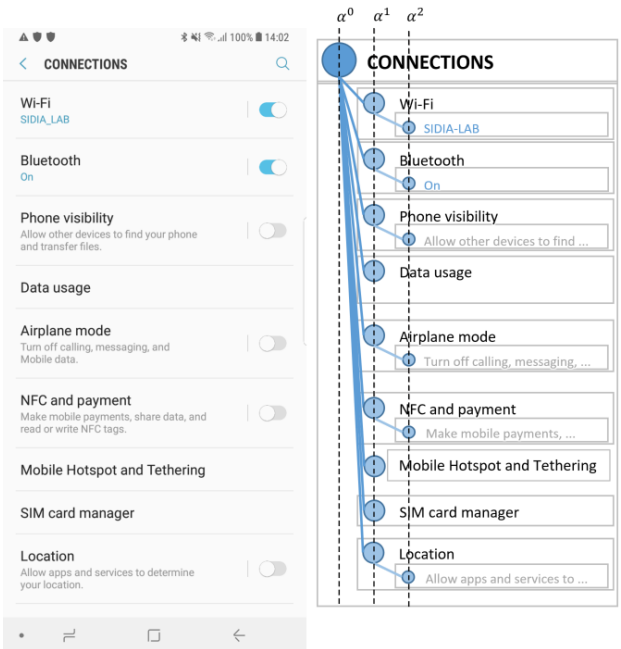


Fig. 3: Example of a simplified dump hierarchy and the *View* blocks.

where $0 < \alpha < 1$ is the weight that defines the importance of each node, d_i is the depth of the node $\mathbf{v}_i$ in the UI hierarchy and D is the total number of textual nodes.

The intuition of the *dump2vec* computation is simple. The title of the screen has already been chosen to represent the whole context, so it gets more importance on the weighted average. However, it is still important to consider the context defined by each component of the screen. Usually, a clickable component on UI has a *TextView* with a small text informing its click event and a *TextView* in lower hierarchical level with some description, which is specific to each component [14].

C. Keywords Embedding

Since we are now representing each screen as a *dump2vec*, we also need to convert the keywords used in search to a feature vector. Different from UI dump, which is structured as a hierarchy, keywords have nothing to indicate the importance of each word. Therefore, the computation of its embedding is done by simply taking the common average over the *word2vec* of all words, as in Equation 1.

Once we have computed *dump2vec* of all screens and we have the keywords embedding, we must have a way to find the screens that most approach to the context defined by the keywords. For this, we employ the K-nearest neighbors, which is explained in the next section.

D. K-Nearest Neighbors

K-Nearest Neighbors (KNN) [9] is a supervised machine learning algorithm applied mostly in classification problems, although it can also be used in regression problems. KNN requires prior annotated data (i.e. the training data) where each observation or data point in the dataset has a set of predictors (features represented by independent variables) and a label (the dependent variable) indicating the class to which the observations belong. The algorithm intuition is based on the assumption that similar observation exists within a proximity, i.e., they are close to each other. Therefore to determine the similarity between two observations, KNN often relies on distance measures like Euclidean or Manhattan, however in this paper the cosine similarity is used instead (refer to Section IV-E). KNN can be described as a query algorithm where the K factor indicates that the algorithm is expected to answer to a given query $\hat{q}$ by returning the K most similar observations in its dataset.

Consider Q being a set of labeled observations, i.e., the training data, each observation $q \in Q$ is a $n-$dimensional vector containing n variables where $n-1$ of them are reserved for the features and 1 discrete value represents the label. The following steps provide a general view of how KNN works:

1) receive an unclassified observation $\hat{q}$ with $n-1$ features;
2) calculate the distance between $\hat{q}$ and every observation $q \in Q$;
3) organize Q in a ranking R such that observations q are ranked by their distance to $\hat{q}$ starting form the smallest distance to biggest;
4) take the top K observations in the ranking R.

E. Cosine Similarity

As mentioned in the last section, cosine similarity was used as the distance measure on KNN. Cosine similarity [9] is a metric that was initially used to estimate the similarity between two documents independently of their sizes. However, this method can be extended to any other problem in which the data can be mapped into a vector space. For the sake of simplicity and convenience, we will explain the method for our problem.

So, formally, after the search keywords are embedded into a feature vector $\mathbf{v}_{keywords}$ (see section IV-C) and the dump to be compared with is mapped into *dump2vec* space as $\mathbf{v}_{dump}$ (see section IV-B), the cosine of the angle between these vectors is measured. We compute this cosine as follows:

$$cos(\theta) = \frac{\mathbf{v}_{keywords} \cdot \mathbf{v}_{dump}}{\|\mathbf{v}_{keywords}\| \cdot \|\mathbf{v}_{dump}\|} \tag{3}$$

Since we take into account the cosine of the angle between two vectors, this measurement ranges from -1 to 1, being -1 totally dissimilar and 1 totally similar.

V. Experimental Framework

A. Implementation

The implementation was done in Python with the *gensim*[1] library due to its facility on dealing with natural language processing methods. Regarding the *word2vec* model, it was used the *word2vec-Android*[2], a model trained on a corpora

[1] *gensim* is a Python library for natural language processing. Available in: https://radimrehurek.com/gensim/

[2] Pre-trained word2vec-Android available in: https://github.com/fanglinchen/Word2Vec_Android

extracted from the Stack Exchange Data Dump[3] that contained all posts up to 2016 with Android tag in the Stackoverflow site[4].

B. Evaluation Methodology

For evaluation, a dataset was collected from the **Settings** app (version 9) of a Samsung Galaxy S10+ (SM-G975U), with Android P. Adopting a depth-first-like approach, a systematic exploration was done to collect as much UI dump as possible. As result, 122 screens have been visited and saved to the dataset. These screens are the ones to be found, given a query of words.

Regarding the search queries, part of them was generated by getting random textual components of every screen on the dataset, and other part was manually generated by getting queries that could not be handled correctly by the Settings search tool. As a result, 468 queries have been generated. For evaluation, each query randomly collected was associated to the screen from where they were obtained. On the other hand, for each manually generated query the correct screen was defined as the screen that we found to be the best match for it.

Finally, the evaluation metric adopted in this work was the *Top-K accuracy*. This metric is useful when the estimator returns several estimations, for example KNN. So, if one of the K top estimations correspond to the expected result, it is considered as a true positive case. Otherwise, it is considered as a false positive. Once all N queries have been tested and the amount of true positives TP have been counted, the *Top-K accuracy* can be computed by the following equation:

$$\text{Top-K} = \frac{TP}{N} \tag{4}$$

Besides this metric, a qualitative analysis is also considered during evaluation. The following section shows experimental results and a discussion over the possible reasons of false positives is provided.

VI. Results

The proposed method was tested for K ranging from 1 to 10 in KNN over the 468 search queries. However, during the test we have noted that among the randomly generated queries, there were cases where the selected words were names, e.g., name of app, wi-fi network, security certificates, that were not included in the vocabulary used for *word2vec* training. Furthermore, users probably would not search specific names on the search tool, they would instead search for the screen that contains those names, e.g., wi-fi, which contains the names of available networks. Therefore, another test was done filtering those cases, reducing to 388 queries. Figure 4 shows the accuracies obtained for both tests on the two first columns (blue and orange) in the chart, and it is possible to observe

[3]An anonymized dump of all user-contributed content on the Stack Exchange network available in: https://archive.org/details/stackexchange

[4]Stackoverflow is a question and answer site for programmers. Available in: https://stackoverflow.com/

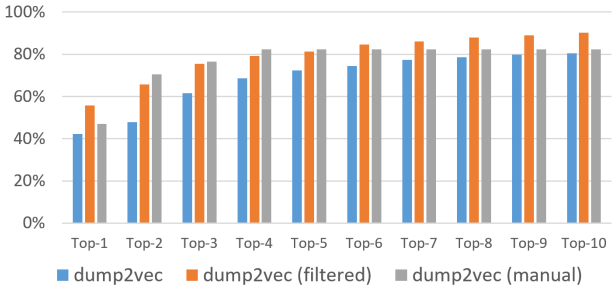


Fig. 4: Obtained results.

Queried words	Expected Screen	Best Estimation
I. App names		
Google Play services	Settings	Mobile data usage
Duo Preview	Add account	Screen saver
Briefing	Mobile data usage	Region
Messages	Wi-Fi data usage	Status
Email	Wi-Fi data usage	Set secure screen lock
Call Filter	Wi-Fi data usage	Creating Mobile Hotspot
My Files	Full screen apps	Encrypt SD card
Call Filter	Full screen apps	Creating Mobile Hotspot
Messages	Full screen apps	Status
Calculator	Full screen apps	Roaming clock
II. Generic terms and screens		
App details in store	App info	Application data usage
Mobile data	App info	Data usage
Service provider SW ver.	Software information	SIM card status
Device admin apps	Device admin apps	Wi-Fi control history
III. Could be considered		
Face recognition	Biometrics and security	Face recognition
Do not disturb	Notifications	Do not disturb
Factory data reset	Status	Factory data reset
All languages	Add a language	Language
Turn on as scheduled	Turn on as scheduled	Night mode
IV. Context issue		
Send diagnostic data	Biometrics and security	Data usage
Send SOS messages	Advanced features	Vibration intensity
Samsung suggested	Add a language	Legal information
Easy mute	Motions and gestures	Sound mode
Media	Allow exceptions	Font style
Smart Switch	Accounts and backup	Advanced features
V. Misleading *dump2vec*		
Show layout bounds	Developer options	App icon badges
Bug report shortcut	Developer options	Legal information
Force allow apps on external	Developer options	Allow app while Data saver on
Mobile data always active	Developer options	Data usage
Always show crash dialog	Developer options	App icon badges

TABLE I: Examples of false positives.

that in more realistic situation, the proposed method achieves Top-10 accuracy of 90.29%. The gray columns in the chart of Figure 4 presents the accuracies over those queries manually generated, showing that 82.35% of them can be handled successfully with the screen context search.

Regarding the found false positive cases, we can make some observations about probable causes and possible evaluation failures. One of the main source of false positives is querying names of installed applications. Because application names can appear in many different screens that usually contain a list of applications, the estimation end up being not so precise, as shown in the first section of Table I. For example, one could find the term *Email* on screens such as *Apps*, *Wi-Fi data usage*, *Full screen apps*, *Memory usage*, among others. Therefore, saying that a single screen is the correct one for an application name may not be a good approach for evaluation.

Another issue that may be considered in our results is about generic terms and screens, i.e. terms that have no specific context for appearing in many screens, or different screens that follow the same template, e.g., App Info screen of an installed application. On these cases, if the queried words are

generic, the estimation will not return a satisfactory result, as shown in the second section of Table I.

The third section of Table I shows some examples of cases that were evaluated as false positive but conceptually could be considered as true positive. When evaluating the correctness of the estimation, we considered only if one of the estimated screens was the same as the one from which the queried words were taken. However that screen is not necessarily the only best recommendation for that queried words. For example, when looking for *Face recognition*, the expected result was the *Biometrics and security* screen, but a result that leads directly to the *Face recognition* screen also seems to be correct, since it is the destiny of the clickable component *Face recognition* on the *Biometrics and security* screen.

Besides, there are those related to components that do not seem to fit well in the screen context. In our method, we assume each screen has a context that can be interpreted as a distribution of word vectors over the feature space. However, there are some cases that a specific *TextView* node contains terms that are not very well fitted on the screen context distribution on the feature space, representing an outlier. In this sense, when this particular component is searched, certainly its context will not match with the *dump2vec* of its actual corresponding screen. Some examples are presented on the fourth section of Table I.

Lastly, *dump2vec* presented a disadvantage on situations where the searched screen is too broad in terms of context, and there are other screens with more restrict context that match with the queried words. An example that shows this situation is when we search for *Show layout bounds*, which leads to the *App icon badges* screen instead of *Developer options* that was the expected result. By screen title, it is difficult to see the case, but checking inside the *App icon badges* screen, one can note that almost every *TextView* contains the word "*Show*", while in the *Developer options* screen there are few nodes that characterize similar context to the queried words. Moreover, since all words in the query receive the same importance, if there is a screen with high frequency of a particular word, this will affect the estimation. Therefore, in such cases, the way how *dump2vec* encodes the screen context end up misleading the screen search. More examples are shown in the last section of Table I

VII. Conclusion

In this work, we proposed a method for searching screens on Android Settings by their context instead of the literal keywords. To get the context-based search, the CBOW model of *word2vec* was used to encode the context of every screen by computing the proposed *dump2vec*. This approach allowed us to get more than 90% of Top-10 accuracy, and solve around 82% of cases that could not be handled by the current search tool of the Android Settings.

It is important to note that the Android Search framework is used on the Settings search tool, but it is also available to Android developers so they can use it on their application [3]. The search algorithm however, can be implemented in a specific Android activity, which receives the search queries and returns the search results. Knowing this, one could apply the search method proposed in this paper to make the search tool more intelligent, and consequently more inclusive. Beyond app development, this approach could also be employed for further applications, such as testing automation for searching a specific feature to be tested.

For future work, we will be concentrating efforts to get better trained *word2vec* model with more recent corpora. Also, a study of how the context of an app is organized may be considered to improve the search approach. And, not less important, to reformulate a more efficient evaluation method.

References

[1] P. Zheng and L. M. Ni, Smart phone and next generation mobile computing. San Francisco, CA: Elsevier, 2006.

[2] A. Ali, M. Alrasheedi, A. Ouda, and L. Fernando Capretz, "A STUDY OF THE INTERFACE USABILITY ISSUES OF MOBILE LEARNING APPLICATIONS FOR SMART PHONES FROM THE USER'S PERSPECTIVE," International Journal on Integrating Technology in Education (IJITE), vol. 3, no. 4, 2014. [Online]. Available: http://www.airccse.org/journal/ijite/papers/3414ijite01.pdf

[3] Google Developers, "Search Overview — Android Developers," 2019. [Online]. Available: https://developer.android.com/guide/topics/search

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," jan 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[5] statcounter GlobalStats, "Mobile Operating System Market Share Worldwide. [Online]," p. 1, 2019. [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/worldwide

[6] P. Gupta, R. Negi, and S. Shekhar, "Searching made easy: A multithreading based desktop search engine," in 2017 7th International Conference on Communication Systems and Network Technologies (CSNT). IEEE, nov 2017, pp. 376–379. [Online]. Available: https://ieeexplore.ieee.org/document/8418570/

[7] R. Kumar, S. K. Singh, and V. Kumar, "A heuristic approach for search engine selection in meta-search engine," in International Conference on Computing, Communication & Automation. IEEE, may 2015, pp. 865–869. [Online]. Available: http://ieeexplore.ieee.org/document/7148496/

[8] D. Liu, X. Xu, and Y. Long, "On member search engine selection using artificial neural network in meta search engine," in 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS). IEEE, may 2017, pp. 865–868. [Online]. Available: http://ieeexplore.ieee.org/document/7960113/

[9] K. Shi, L. Li, H. Liu, J. He, N. Zhang, and W. Song, "An improved KNN text classification algorithm based on density," in CCIS2011 - Proceedings: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems. IEEE, 2011, pp. 113–117.

[10] Y. Jiang, S.-J. Lee, Y.-S. Lin, and J.-Y. Jiang, "A Similarity Measure for Text Classification and Clustering," IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, vol. 26, no. 7, p. 1575, 2014. [Online]. Available: http://www.ieee.org/publications_standards/publications/rights/index.html

[11] T. V. Nguyen, A. T. Nguyen, H. D. Phan, T. D. Nguyen, and T. N. Nguyen, "Combining Word2Vec with Revised Vector Space Model for Better Code Retrieval," in 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). IEEE, may 2017, pp. 183–185. [Online]. Available: http://ieeexplore.ieee.org/document/7965297/

[12] Y. Nan, L. Yaping, and L. Qing, "Improving Search Result Clustering by Enriching Snippets with Word2Vec Model," in 2017 14th Web Information Systems and Applications Conference (WISA). IEEE, nov 2017, pp. 33–37. [Online]. Available: http://ieeexplore.ieee.org/document/8332582/

[13] Z. S. Harris, "Distributional Structure," WORD, vol. 10, no. 3, pp. 146–162, 1954. [Online]. Available: https://www.tandfonline.com/action/journalInformation?journalCode=rwrd20

[14] Google Developers, "Layouts — Android Developers," 2019. [Online]. Available: https://developer.android.com/guide/topics/ui/declaring-layout