Using Topic Information to Improve Non-Exact Keyword-Based Search for Mobile Applications*

Eugénio Ribeiro[1,2,3] [0000−0001−7147−8675],
Ricardo Ribeiro[1,3] [0000−0002−2058−693X],
Fernando Batista[1,3] [0000−0002−1075−0177], and
João Oliveira[3] [0000−0003−4654−0881]

[1] INESC-ID Lisboa, Portugal
[2] Instituto Superior Técnico, Universidade de Lisboa, Portugal
[3] Instituto Universitário de Lisboa (ISCTE-IUL), Portugal
eugenio.ribeiro@inesc-id.pt

Abstract. Considering the wide offer of mobile applications available nowadays, effective search engines are imperative for an user to find applications that provide a specific desired functionality. Retrieval approaches that leverage topic similarity between queries and applications have shown promising results in previous studies. However, the search engines used by most app stores are based on keyword-matching and boosting. In this paper, we explore means to include topic information in such approaches, in order to improve their ability to retrieve relevant applications for non-exact queries, without impairing their computational performance. More specifically, we create topic models specialized on application descriptions and explore how the most relevant terms for each topic covered by an application can be used to complement the information provided by its description. Our experiments show that, although these topic keywords are not able to provide all the information of the topic model, they provide a sufficiently informative summary of the topics covered by the descriptions, leading to improved performance.

Keywords: Application search · Topic information · Non-exact queries.

1 Introduction

Nowadays, the offer of mobile applications with different functionality in app stores is constantly increasing. Thus, although users spend most of their time inside the applications, they also spend a significant amount of time searching for and installing new applications. This reveals the need for effective search and recommendation systems. However, most queries in app store search engines contain just the name of the application that the user is looking for. This means that users target specific applications, either because they were suggested to

* This work was supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia (FCT), with reference UIDB/50021/2020, and PT2020, project number 39703 (AppRecommender).

them by acquaintances or they found them using other approaches, such as web search. Word of mouth has always been an important form of marketing. Thus, searching for applications suggested by acquaintances is normal. On the other hand, searching for applications on the web is somewhat of a countersense, since app stores have specialized search engines. However, those engines are typically unable to semantically interpret the queries, considering their characteristics and context. Thus, they lose to web search engines, which are able to process more complex queries by crawling large amounts of data. Overall, data is the defining factor, since queries in app store search engines are typically short and the amount of data available to search on is reduced, especially in comparison to the whole web. Thus, in order to deliver better search results, app store search engines must overcome the data problem, either by semantically interpreting the queries or by inferring additional information from the existing data to improve the match ratio between the queries and relevant applications.

Topic information has been proved important in the context of information retrieval [23], including in search for applications [14, 24], since it enables matching when similar contexts are referred to using different words. However, while the existing approaches to topic-based retrieval are based on similarity between topic distributions, the highly distributed search approaches used in most app stores are based on keyword-matching and boosting according to popularity factors. In this paper, we explore means to include topic information in such approaches, in order to improve their ability to retrieve relevant applications for non-exact queries, without impairing their computational performance. More specifically, we start by creating topic models specialized on application descriptions. Then, we identify the most relevant and distinctive terms to represent each topic. Finally, we explore how the relevant terms for each topic covered by an application can be used to complement the information provided by the words of its description in the context of non-exact keyword-based search.

In the remainder of the paper, we start by providing an overview on related work on search for mobile applications, in Section 2. Then, in Section 3, we present our approach for including topic information in keyword-based search. Section 4 describes our experimental setup, including the dataset, evaluation approach, and implementation details that allow future reproduction of our experiments. The results of those experiments are presented and discussed in Section 5. Finally, Section 6 summarizes the contributions of this paper and provides pointers for future work.

2 Related Work

The algorithms behind the search engines of the two major mobile app stores, Google Play [9] and Apple's App Store [1], are constantly evolving and, since they are proprietary, not all the details are disclosed. However, it is known that they are mostly based on keyword-matching with multiple fields regarding the applications and boosting based on popularity factors or for business purposes. Most alternative app stores are also proprietary and use similar search approaches.

Among these, many are based on the Lucene search engine [5] or one of the highly distributed search engines built on top of it, such as Solr [19] or Elasticsearch [2], which focus on speed and availability.

For instance, Aptoide's search engine [21] is based on Elasticsearch and performs keyword-matching between the terms present in the query and fields containing application information regarding its name, its package, and its description. Furthermore, in order to improve the match ratio, it includes alternatives of the name, such as abbreviations, lemmatized words, and split and merged versions of multi-word names. Matches with each of these fields contribute to the relevance score with different weights. Furthermore, information regarding the number of downloads of the application, its rating, the number of users that rated the application, and whether it should be promoted for business purposes is used to boost the score.

Mobilewalla [6, 7] uses an application search engine based on Lucene. The keyword-matching fields include the application name, description, and its categories, while boosting fields include the rating and rank of the application, its age and the frequency of releases, the number of users that commented and rated the application, the number of applications in the same categories, and information about the developer. The main difference from Aptoide's approach is that the computation of alternatives is not on the application side, but rather on the query side. That is, the knowledge base does not include alternative application names, but multiple versions of the query are generated by stemming and lemmatizing its words. Furthermore, if using all the terms in the query does not lead to the retrieval of enough results, alternative queries are generated by dropping part of the terms. Alternatively, the query can be expanded by replacing terms with corresponding synonyms or hyponyms.

To reduce the number of mismatches in keyword search caused by the use of different terms by the users and developers, Tencent's MyApp [24] extends the queries performed in its search engine with topic and tag information. The set of more than a thousand topics was obtained by applying Latent Dirichlet Allocation (LDA) [3] to the title and descriptions of a million applications. Using this model, each application can then be represented as a topic distribution. Since the queries are typically too short for performing an accurate inference of their topic distribution, they are extended with information from the applications which have been clicked on after similar queries. By computing the similarity between the topic distribution of an extended query and those of the applications, the search engine is able to identify the most relevant applications for the query in terms of topic. Tag information is used to add fine-grained semantics to the query. The set of tags of an application is a filtered combination of human labels and tags obtained by crawling web and usage data regarding that application. A query is extended with tags using a template-based method which uses information from clicked applications to select the templates. Finally, the LambdaMART algorithm [4] is applied to aggregate the applications obtained through term, topic, and tag matching and order them for presentation to the user. Although considering topic and tag information leads to a higher match ra-

tio, the query extensions and the computation of its topic distribution introduce a high computational overhead during search.

Park et al. [14] explored the use of user reviews to improve the match ratio by bridging the gap between the vocabulary used by users and developers. Furthermore, in their study, they compared the performance of multiple retrieval approaches – BM25(F) [18], Query Likelihood (QL) [16], and LDA-Based Document Model (LBDM) [22]. The first is based on keyword-matching, the second on language modeling, and the last on the combination of keyword- and topic-matching. While relying solely on application descriptions, the highest performance on a set of more than 50 non-exact queries was achieved using LBDM with a topic model with 300 topics trained on the descriptions of 40,000 applications. This confirms that topic information is able to complement the information explicitly present in descriptions by providing associations with words that refer to similar topics. Furthermore, using the information provided by user reviews significantly improved the performance of every retrieval approach. The best results were achieved using an approach that combines language modeling with topic-based retrieval. Separate topic models are trained on descriptions and reviews, but the review-level model is conditioned by the description-level one. This allows the identification of review topics that do not match any description topic and, thus, are not relevant for application retrieval. In a later study, Park et al. [13] also relied on language and topic modeling to induce queries from users' social media text and recommend relevant applications.

3 Topic Information for Keyword-Based Search

When topic information is used for retrieval, documents are typically ranked according to the similarity between their topic distribution and that of the query. The approach we describe below enables the representation of topic information as keywords that can be used by keyword-based retrieval approaches. These keywords correspond to a set of terms that are sufficiently relevant and distinctive to identify a topic and, thus, can function as its summary. In addition to how the the keywords are generated from the topic models and included in the retrieval approaches, we also describe preprocessing and topic model training approaches that allow the generated models and the corresponding keywords to focus on relevant aspects for mobile application search. Since our intent is to show that topic information can be represented as keywords, we focus on obtaining that information from application descriptions. However, the approach can be generalized to other textual information sources, such as user reviews.

3.1 Preprocessing

In the preprocessing phase, each description is split into sentences and dependency parsed and its tokens are Part-of-Speech (POS) tagged and lemmatized. By splitting into sentences, we are able to train both generic models based on whole descriptions and more specific ones based on the sentences. POS tagging

allows filtering by the word classes that are more relevant for application search, such as nouns, adjectives, and verbs. While the first reveal the concepts focused by the description, the second reveal their characteristics. Furthermore, in this context, non-auxiliary verbs typically reveal functionality. By combining the POS tags and the dependency parse of each sentence, we can identify adjectives and verbs that are negated. This is important to avoid grouping descriptions or sentences that have opposite meanings. Finally, lemmatization simplifies matching and leads to the generation of more constrained models.

Additionally, while terms that occur in a small set of descriptions or sentences are unrelated to the most relevant topics covered by the whole collection, terms that occur in a large portion of the collection are typically not discriminative. Thus, we discard tokens that are commonly classified as stopwords, as well as those which have a document frequency below a threshold df_{min} or above a threshold df_{max}. The most appropriate values for these thresholds vary according to the model. Those used in our experiments are detailed in Section 4.3.

Since the descriptions are lemmatized, we also lemmatize queries, in order to enable matching. No additional preprocessing is performed on the queries.

3.2 Topic Models

We obtain our topic models using a classical LDA approach [3]. In an LDA model, topics are seen as term distributions while documents are seen as mixtures of topics. Thus, the definitions of term and document have a wide impact on the aspects that are actually modeled. In typical applications of LDA, documents are relatively large pieces of text, such as news articles or reports, and the terms are the words in the documents, excluding stopwords. However, as referred in the previous section, we can split the descriptions in different ways and filter the tokens by specific word classes, in order to identify topics that are more informative for application retrieval.

Regarding terms, after the preprocessing described in the previous section, when training the topic models, we discard tokens that are not nouns, adjectives, or non-auxiliary verbs. Furthermore, negated verbs and adjectives are distinguished from their positive counterparts.

In terms of documents, the most straightforward approach is to consider each description a document. However, since the LDA model uses a Bag of Words (BoW) approach, it is not aware of the dependency relations between nouns and adjectives nor between verbs and their arguments. Thus, it assumes that all the terms that occur in the document are related in the same manner. This leads to the identification of more generic topics that may group terms that are not directly related in the descriptions. On the other hand, each individual sentence in a description typically contains terms that are directly related. Thus, training a sentence-level model leads to the identification of more constrained topics. Since both kinds of topic may provide relevant information for application retrieval, we train both a description-level model and a sentence-level model.

Finally, similarly to any application of LDA, the number of topics, N_t, must be defined a priori. The selection of an appropriate value for this parameter

reduces the probability of identifying topics that are either too generic to be useful or so specific that capture irrelevant aspects. However, this categorization depends on the intended use for the topics. Furthermore, the best value typically depends on the dimensionality of the collection and the number of terms in the vocabulary. Thus, the most appropriate number of topics is expected to differ between the description- and sentence-level models.

3.3 Topic Keywords

Having trained the topic models, in order to use the information that they capture in the context of keyword-based retrieval approach, it must be transformed into keywords. A straightforward approach is to represent each topic by the top n terms in its distribution. However, using a fixed number of terms may lead either to the inclusion of non-relevant terms or the discarding of terms that are relevant for a topic. Thus, we use the approach described in Algorithm 1 to identify the set of relevant terms for each topic. The idea behind it is to approximate the term distribution of a topic by a negative exponential function and select the terms that appear before the inflection point as relevant. Thus, given a term distribution, T, the algorithm starts by sorting it in decreasing weight order. Then, only the n terms with highest weight in the distribution are considered, as long as their weight is above a residual threshold, r. To account for noisy distributions, the weights of the terms are then smoothed using a weighted running average that further approximates the distribution to a negative exponential one. The remainder of the algorithm identifies the inflection point by analyzing the weight differences between consecutive terms.

Algorithm 1 Relevant Terms

Input: T // The term weight distribution
Input: r // The residual weight threshold
Input: n // The maximum number of terms
Output: R // The relevant terms

$T \leftarrow \textsc{Sort}(\{(t,w) \in T\}, (t_i,w_i) < (t_j,w_j) := w_i > w_j)$
$W \leftarrow \{w_i : (t_i,w_i) \in T, w_i > r, 0 < i \leq n\}$
$W \leftarrow \textsc{WeightedRunningAverage}(W)$
$d \leftarrow$ **false**
for $i = 1 : |W|$ **do**
  $m \leftarrow (W_i - W_{i-1}) \times |W|$
  if d **and** $m > -1$ **then**
    break
  else if $m < -1$ **and not** d **then**
    $d \leftarrow$ **true**
  end if
end for
$R \leftarrow T_{1:i}$
return R

3.4 Application Retrieval

The approach described in the previous section identifies the set of relevant terms for a topic. To identify the set of description-level topic keywords for an application, a, we use the corresponding topic model to compute the topic mixture of its description. Then, we discard topics with weight below a residual threshold, r. The keywords are then given by the aggregate of the relevant terms for the remaining topics. The set of sentence-level topic keywords is computed in a similar fashion. However, the topic mixture is computed for each sentence in the application's description and the keywords of the application are given by the aggregate of the topic keywords of its sentences.

For retrieval purposes, each application is represented by a set of three textual fields for keyword matching, $\{a_d, a_{st}, a_{dt}\}$, corresponding to its textual description, the set of sentence-level topic keywords, and the set of document-level topic keywords, respectively. In our experiments, we explore two keyword-based retrieval approaches – BM25F [18] and Elasticsearch [2]. While the first is a widely used information retrieval approach for semi-structured textual data, the latter is a highly distributed search engine focused on speed and availability. Given a query, q, both return a list of applications ordered by relevance score. However, the scoring function differs. The adaptation of the two scoring functions to our problem is presented below.

BM25F We use the same formulation of the base BM25F scoring function found in several previous studies (e.g. [8, 15, 14]):

$$\text{score}(q, a) = \sum_{t \in q \cap a} \left(\text{idf}(t) \times \frac{(k_3 + 1)c(t, q)}{k_3 + c(t, q)} \times \frac{(k_1 + 1)c'(t, a)}{k_1 + c'(t, a)} \right) \tag{1}$$

where $\text{idf}(t)$ is the inverse document frequency of term t in the set of descriptions, k_1 and k_3 are parameters that can be tuned according to the problem, $c(t, q)$ is t's count in q and $c'(t, a)$ is t's normalized count in a, weighted by field:

$$c'(t, a) = \frac{w_d \cdot c(t, a_d)}{1 - b + b\frac{|a_d|}{\bar{n}}} + w_{st} \cdot c(t, a_{st}) + w_{dt} \cdot c(t, a_{dt}) \tag{2}$$

where w_d, w_{st}, and w_{dt} are the weights given to textual descriptions, sentence-level topic information, and description-level topic information, respectively, and b is a parameter that controls the strength of the normalization according to the mean description length, $\bar{n}$. We do not include normalization factors for topic information, since the number of topic keywords is not relevant for the problem.

Elasticsearch Scoring in Elasticsearch is based on Lucene's Practical Scoring Function, which computes individual scores for each field, f, as

$$\text{score}(q, f) = \frac{1}{\sqrt{\sum_{t \in q} \text{idf}(t)^2}} \times \frac{|q \cap f|}{|q|} \times \sum_{t \in q} \left(\frac{\text{tf}(t, f) \cdot \text{idf}(t)^2 \cdot w_f}{\sqrt{|f|}} \right) \tag{3}$$

where the first factor is a cross-query normalization factor, the second factor boosts according to the number of matching terms, $\text{tf}(t, f)$ is the term frequency of t in f, $\text{idf}(t)$ is the inverse document frequency of t in the field f of all applications, and w_f is the weight of the field.

The relevance score of an application for a query is then given by

$$\text{score}(q, a) = (1 - \text{tb}) \cdot \max_{f \in a} \text{score}(q, f) + \text{tb} \cdot \sum_{f \in a} \text{score}(q, f) \tag{4}$$

where tb is a parameter that controls the extent to which the non-top scoring fields contribute for the overall relevance score of the application.

4 Experimental Setup

In this section, we describe our experimental setup, including the dataset, the evaluation approach, and implementation details that enable the reproduction of our experiments in future studies.

4.1 Dataset

In our experiments, we use the dataset crawled by Park et al. [14], which features information regarding 43,041 mobile applications. Among other less relevant information, for each application, it includes the name, category, description, developer, date of publication, price, and number of downloads. Furthermore, it includes review information in the form of the number of reviews, the average rating, and textual data of up to 50 reviews per application, with a total of 1,385,607 reviews. Additionally, the dataset features 56 non-exact queries generated from forum posts that targeted an application with a specific functionality. Each of these queries is paired with relevance information of the top 20 applications retrieved using multiple retrieval approaches. On average, there are 81 judged applications per query. Each query-application pair was annotated by three users in a three-value scale: 0 for no satisfaction at all, 1 for partial satisfaction, and 2 for perfect satisfaction. The relevance score is then given by the average judgement of the annotators.

We decided to use this dataset since, to the best of our knowledge, it is the only publicly available one featuring relevance scores of query-application pairs. Furthermore, the results of previous studies on this dataset provide a baseline for comparison of our results.

4.2 Evaluation Approach

In order to compare our results with those reported in previous studies on the same dataset, we use the same evaluation metric as Park et al. [14], that is, the Normalized Discounted Cumulative Gain (NDCG) [11] at 3, 5, 10, and 20 top retrieved applications. NDCG is a widely used metric in the context of

information retrieval to measure the effectiveness of search engine algorithms, by assessing whether the results are ordered by relevance. In the context of search for mobile applications, looking beyond the fifth result typically involves scrolling and, thus, the NDCG at 3 and 5 are the most important to consider.

Given a graded relevance scale of applications in a result set for a given query, to compute the corresponding NDCG, we start by computing the Discounted Cumulative Gain (DCG) of the result set:

$$\mathrm{DCG}_k = \sum_{i=1}^{k} \frac{\mathrm{rel}_i}{\log_2(i+1)} \tag{5}$$

where rel_i is the relevance of the i-th application in the result list for query q. This metric measures the gain of an application based on its position in the result list. The gain is then accumulated from the top of the list, with the gain of each result being discounted as the distance from the top increases. The NDCG is then obtained through normalization using the Ideal Discounted Cumulative Gain (IDCG), that is, the DCG of a perfectly sorted result list:

$$\mathrm{NDCG}_k = \frac{\mathrm{DCG}_k}{\mathrm{IDCG}_k} \tag{6}$$

As baselines, we use the results achieved using both BM25F [18] and Elasticsearch [2] when relying solely on matching with description texts, without topic information. That is, $w_d = 1, w_{st} = 0, w_{dt} = 0$ in Equations 2 and 3. This transforms BM25F into its single-field version, BM25. Additionally, we compare our results with the LBDM [22] and Google Play [9] results reported by Park et al. [14]. Since LBDM is able to take advantage of all the information captured by the topic model, its results provide an upper bound for performance when pairing topic information with keyword-matching with application descriptions. On the other hand, Google Play results serve as an indicator of the performance of current app store search engines, which rely on additional fields for matching and on popularity information for boosting.

4.3 Implementation Details

The application descriptions provided in the dataset contain HTML tags and escape characters. We used the html2text package [20] to convert them to plain text. Then, we used the spaCy parser [10] for sentence splitting, dependency parsing, POS tagging, and lemmatization. Since the set of English stopwords used by spaCy is too aggressive, we relied on the set defined in NLTK [12] while filtering the tokens. Additionally, for consistency with the experiments by Park et al. [14], we defined $\mathrm{df}_{min} = 5$ and $\mathrm{df}_{max} = 0.3$ for keyword-matching with the description. That is, we discarded tokens that appeared in less than 5 descriptions or in more than 30%.

To train the topic models, we used the parallelized LDA implementation provided by the gensim library [17]. Additionally, we performed a more aggressive low-frequency token filtering. While training the sentence-level topic model, we

used $\mathrm{df}_{min} = 10$, since, in this case, we considered the sentence frequency and lower values still included many terms that only occurred in a single description. While training the description-level topic model, we used $\mathrm{df}_{min} = 0.01$, that is, we discarded tokens that appeared in less than 1% of the descriptions, in order to identify more generic topics. In terms of the number of topics, we defined $N_t = 300$ for the description-level model, for consistency with the experiments by Park et al. [14], and $N_t = 100$ for the sentence-level model, since for higher values there were topics that were not attributed to any application. While identifying the relevant terms for each topic, we defined a maximum number of terms $n = 20$ and a residual weight threshold $r = 0.01$. The same residual weight threshold was used to attribute topics to applications.

For keyword-matching with the description in BM25(F), we used the same parameters as Park et al. [14]. That is, $k1 = 4$, $k3 = 1000$, and $b = 0.4$. The remaining parameters – w_d , w_{st}, and w_{dt} for both BM25F and Elasticsearch, and tb for Elasticsearch – were tuned using grid search to maximize the mean of the four NDCG results, that is,

$$\bar{\mathrm{NDCG}} = \frac{\sum_{k \in K} \mathrm{NDCG}_k}{|K|}, K = \{3, 5, 10, 20\} \tag{7}$$

For that reason, the concrete values and their meaning are discussed in Section 5.

5 Results

Table 1 shows the NDCG results of our experiments, as well as the reference results achieved using LBDM and Google Play. In the context of search for mobile applications, the NDCG results lose relevance as the number of considered applications increases, since only a reduced set can be shown on screen at each time. Thus, we will focus this discussion on NDCG@3 results. However, since the parameters were tuned to maximize the mean results at the multiple values of k, we will also make some remarks regarding the results achieved when a higher number of applications is considered.

First of all, it is important to note that the baseline BM25 results are one percentage point lower than those reported by Park et al. [14] for $k \in \{3, 5\}$, in spite of using the same values for all the parameters. This is due to differences in preprocessing, especially regarding the filtering of tokens, which also considered frequency in reviews. This means that the results achieved using LBDM are also expected to be lower if using our preprocessing approach.

Overall, due to its focus on temporal performance, Elasticsearch performs worse than BM25(F). When considering textual descriptions, the decrease in performance is between two and three percentage points. However, when considering topic information only, the decrease is around 20 percentage points. This is due to the reduced vocabulary and the normalization factors applied by the Elasticsearch score function. On the other hand, since we consider the whole vocabulary and do not include normalization factors for topic information while computing the BM25F scores, its results are not penalized.

Table 1. NDCG results of BM25(F) and Elasticsearch applied to textual descriptions (D), topic information (T) and their combination (D + T). The last block provides reference results reported by Park et al. [14].

Approach	**NDCG@3**	**NDCG@5**	**NDCG@10**	**NDCG@20**
BM25 (D)	0.569	0.540	0.523	0.537
Elasticsearch (D)	0.540	0.523	0.502	0.512
BM25F (T)	0.554	0.553	0.535	0.530
Elasticsearch (T)	0.341	0.342	0.356	0.370
BM25F (D + T)	0.574	0.542	0.527	0.544
Elasticsearch (D + T)	0.552	0.532	0.504	0.519
LBDM	0.584	0.563	0.543	0.565
Google Play	0.589	0.575	0.568	0.566

Comparing the results achieved using textual descriptions with those achieved using topic information, we can see that the performance of BM25(F) decreases 1.5 percentage points in terms of NDCG@3, but actually increases in terms of NDCG@5 and NDCG@10. This means that the set of topic keywords is an appropriate summary of the information provided by the description. However, these results were achieved when relying solely on the sentence-level topic keywords, that is, $w_{st} = 1$ and $w_{dt} = 0$. Including description-level topic keywords does not lead to improvement. On the other hand, the results using Elasticsearch were achieved using $w_{st} = 2w_{dt}$ and tb = 0.1, which means that the sentence-level topic keywords are still the most informative, but that the document-level topic keywords can provide complementary information.

As expected, the best results are achieved when combining the information provided by textual descriptions and topic information. However, in the case of BM25F, there is only improvement when topic information is given a reduced weight. More specifically, the results reported in Table 1 were achieved with $w_d = 0.96$, $w_{st} = 0$, and $w_{dt} = 0.04$. Several other configurations, including ones that also give weight to sentence-level topic information, lead to similar results. Still, the weights are always severely biased towards the textual descriptions. For instance, the parameters that maximized NDCG@3 in our experiments were $w_d = 0.98$, $w_{st} = 0.01$, and $w_{dt} = 0.01$. This means that the topic keywords are only used as complementary information that enable the retrieval of more relevant applications in specific cases. On the other hand, in the case of Elasticsearch, the mean NDCG was maximized with $w_{st} = 2w_{dt}$, $w_d = w_{dt}$, and tb = 0.5. This means that the relation between the weights of sentence- and description-level topic information is kept in relation to when the textual descriptions are not considered. Furthermore, although higher weight is given to topic information, the value of the tie breaker parameter shows that all fields have an important contribution to the score.

Overall, including topic information improves the performance of Elasticsearch by one percentage point in terms of both NDCG@3 and NDCG@5. However, by comparing the BM25F results with those of LBDM, even assuming that the performance of LBDM is expected to decrease with our preprocessing approach, we can see that the topic keywords are not able to capture all the information provided by the topic models. This happens because the representation of the topics in the form of their most relevant terms does not allow matching with similar keywords that are not as common. Finally, the performance of Google Play shows that additional fields, such as the application titles, and popularity information are relevant for delivering the best results for non-exact queries.

6 Conclusions

In this paper, we have explored how topic information can be represented in the form of keywords to be considered by mobile application retrieval approaches based on keyword-matching. This is important, since app store search engines have strict requirements in terms of temporal performance and availability, which, currently, are only fulfilled by highly distributed retrieval approaches based on multi-field keyword-matching and boosting.

We focused on application descriptions and trained two LDA models, one on whole descriptions and another on their sentences. While the first generates more generic topics, the second captures more fine-grained subjects. Then, we computed the topic mixtures of the application descriptions and represented the topic information of an application as the aggregate of the relevant terms for each topic in its mixture. The set of relevant terms for a topic is identified by approximating its term distribution by a negative exponential function and selecting the terms which appear before the inflection point.

The results of our experiments have shown that both sentence- and description-level topic information provides cues for application retrieval from non-exact queries, leading to improved performance. Furthermore, the topic keywords make a sufficiently informative summary of the information provided by the descriptions. However, they do not allow matching with similar keywords that are not as common. Thus, the performance is still lower than when performing retrieval based on topic similarity, which relies on all the information provided by the topic models. Thus, as future work, it would be interesting to assess whether including synonyms of the relevant terms that occur in the same context can enable matching with those less common keywords without introducing ambiguity.

Furthermore, it is important to assess whether the performance improvement observed by Park et al. [14] when leveraging review data can also be observed when the information captured by the topic models that merge description and review information is provided in the form of keywords.

Finally, it is important to assess how this approach behaves in combination with boosting factors based on popularity.

References

1. Apple: App Store. https://www.apple.com/ios/app-store/ (2008)
2. Banon, S.: Elasticsearch. https://www.elastic.co/ (2010)
3. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet Allocation. Journal of Machine Learning Research **3**, 993–1022 (2003). https://doi.org/10.5555/944919.944937
4. Burges, C.J.C.: From RankNet to LambdaRank to LambdaMART: An Overview. Learning **11**(23–581), 81 (2010)
5. Cutting, D.: Apache Lucene. https://lucene.apache.org/ (1999)
6. Datta, A., Dutta, K., Kajanan, S., Pervin, N.: Mobilewalla: A Mobile Application Search Engine. In: MobiCASE. pp. 172–187 (2011). https://doi.org/10.1007/978-3-642-32320-1_12
7. Datta, A., Kajanan, S., Pervin, N.: A Mobile App Search Engine. Mobile Networks and Applications **18**(1), 42–59 (2013). https://doi.org/10.1007/s11036-012-0413-z
8. Fang, H., Tao, T., Zhai, C.: A Formal Study of Information Retrieval Heuristics. In: SIGIR. pp. 49–56 (2004). https://doi.org/10.1145/1008992.1009004
9. Google: Google Play. https://play.google.com/ (2008)
10. Honnibal, M., Montani, I.: spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing. https://spacy.io/ (2017)
11. Järvelin, K., Kekäläinen, J.: Cumulated Gain-Based Evaluation of IR Techniques. ACM Transactions on Information Systems **20**(4), 422–446 (2002). https://doi.org/10.1145/582415.582418
12. Loper, E., Bird, S.: NLTK: The Natural Language Toolkit. In: ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. vol. 1, pp. 63–70 (2002). https://doi.org/10.3115/1118108.1118117
13. Park, D.H., Fang, Y., Liu, M., Zhai, C.: Mobile App Retrieval for Social Media Users via Inference of Implicit Intent in Social Media Text. In: CIKM. pp. 959–968 (2016). https://doi.org/10.1145/2983323.2983843
14. Park, D.H., Liu, M., Zhai, C., Wang, H.: Leveraging User Reviews to Improve Accuracy for Mobile App Retrieval. In: SIGIR. pp. 533–542 (2015). https://doi.org/10.1145/2766462.2767759
15. Pérez-Iglesias, J., Pérez-Agüera, J.R., Fresno, V., Feinstein, Y.Z.: Integrating the Probabilistic Models BM25/BM25F into Lucene. Computing Research Repository **arXiv:0911.5046** (2009)
16. Ponte, J.M., Croft, W.B.: A Language Modeling Approach to Information Retrieval. In: SIGIR. pp. 275–281 (1998). https://doi.org/10.1145/290941.291008
17. Řehůřek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In: LREC Workshop on New Challenges for NLP Frameworks. pp. 45–50 (2010). https://doi.org/10.13140/2.1.2393.1847
18. Robertson, S., Zaragoza, H.: The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends® in Information Retrieval **3**(4), 333–389 (2009). https://doi.org/10.1561/1500000019
19. Seeley, Y.: Apache Solr. https://lucene.apache.org/solr/ (2004)
20. Swartz, A.: html2text. https://github.com/Alir3z4/html2text/ (2003)
21. Trezentos, P.: Aptoide. https://www.aptoide.com/ (2009)
22. Wei, X., Croft, W.B.: LDA-Based Document Models for Ad-Hoc Retrieval. In: SIGIR. pp. 178–185 (2006). https://doi.org/10.1145/1148170.1148204

23. Yi, X., Allan, J.: A Comparative Study of Utilizing Topic Models for Information Retrieval. In: ECIR. pp. 29–41 (2009). https://doi.org/10.1007/978-3-642-00958-7_6
24. Zhuo, J., Huang, Z., Liu, Y., Kang, Z., Cao, X., Li, M., Jin, L.: Semantic Matching in APP Search. In: WSDM. pp. 209–210 (2015). https://doi.org/10.1145/2684822.2697046