Semantic Search of Mobile Applications Using Word Embeddings

João Coelho
Caixa Mágica Software, Lisbon, Portugal
Instituto Superior Técnico, Lisbon, Portugal

António Neto
Caixa Mágica Software, Lisbon, Portugal
University Institute of Lisbon, Portugal

Miguel Tavares
Caixa Mágica Software, Lisbon, Portugal
Lusophone University of Humanities and Technologies, Lisbon, Portugal

Carlos Coutinho
Caixa Mágica Software, Lisbon, Portugal
ISTAR-IUL, University Institute of Lisbon, Portugal

Ricardo Ribeiro
University Institute of Lisbon, Portugal
INESC-ID Lisbon, Portugal

Fernando Batista
University Institute of Lisbon, Portugal
INESC-ID Lisbon, Portugal

Abstract

This paper proposes a set of approaches for the semantic search of mobile applications, based on their name and on the unstructured textual information contained in their description. The proposed approaches make use of word-level, character-level, and contextual word-embeddings that have been trained or fine-tuned using a dataset of about 500 thousand mobile apps, collected in the scope of this work. The proposed approaches have been evaluated using a public dataset that includes information about 43 thousand applications, and 56 manually annotated non-exact queries. Our results show that both character-level embeddings trained on our data, and fine-tuned RoBERTa models surpass the performance of the other existing retrieval strategies reported in the literature.

2012 ACM Subject Classification Information systems → Retrieval models and ranking; Information systems → Document representation; Information systems → Language models; Information systems → Search engine indexing; Information systems → Similarity measures; Computing methodologies → Machine learning

Keywords and phrases Semantic Search, Word Embeddings, Elasticsearch, Mobile Applications

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.12

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

1 Introduction

The penetration of mobile devices in society has led most companies to see them as indispensable means for being in close contact with their customers. According to [8], Google Play Store has 2.6 million mobile apps available, Apple's iOS App Store has 1.85 million, and

Aptoide has about 1 million[1], which creates an extremely tough competition between apps. In terms of app downloads, in 2019 there were 204 billion app downloads, a number that has been increasing over the years. However, many of these downloads consist of multiple attempts to find the right app. Many downloaded apps are never used, and in 77% of cases, apps are not used again within 72 hours of installation. This shows a big misalignment between the supply of apps by the app stores (distribution services) and the demand for them by the consumers (discovery). Furthermore, around 2019, 52% of apps were discovered by word-of-mouth between acquaintances, friends or family, and only 40% were discovered by searching app stores. These inefficiencies make app discovery and distribution a considerable and extremely relevant challenge, since they take place in a market of massive penetration in societies and seriously affect the relationship between companies and consumers.

Based on this problem, the strategic objective of the AppRecommender project is to investigate and develop technologies capable of offering the right app, to the right customer, at the right time, by proposing a semantic search engine. The goal is to optimize current app distribution and discovery services and, inherently, to promote closer ties between companies and their target customers. The impact will be to increase user simplicity, efficiency and satisfaction in the discovery of apps, by optimizing the alignment between their needs, characteristics and context with the apps offered by the app store. As for developers or companies promoting mobile apps, the impact is on the level of increased proximity to target consumers, and optimizing their commercial success. The work here described was developed in the scope of the AppRecommender project.

This paper describes a dataset containing information about 500 thousand mobile apps, collected in the scope of this work, and proposes a set of approaches for semantic search, using the unstructured textual information contained in their name and description. The proposed approaches make use of word-level, character-level, and contextual word-embeddings that we have trained or fine-tuned with our available data. In order to evaluate the proposed approaches, we have used the public dataset described in [14], which includes information about 43,041 mobile applications and 56 non-exact queries previously annotated. Our results show that both character-level embeddings trained on our data, and fine-tuned RoBERTa models surpass the performance of the other existing retrieval strategies reported in the literature for this database.

This paper is organized as follows: Section 2 presents an overview of the related literature, focusing on existing work on semantic retrieval. Section 3 presents the data used in our experiments, which consists of the data that we have collected to train or fine-tune our models, and also the dataset that we have used for evaluation. Section 4 describes our set of approaches. Section 5 describes the conducted experiments and the achieved results. Finally, Section 6 summarizes the most relevant contributions, and presents the future work.

2 Related Work

As previously mentioned, there is a big misalignment between the supply of apps by the app stores and the demand for them by the consumers. In fact, this happens even considering that mobile app stores have search engines that allow users to find apps according to a provided query.

In general, although difficult to assess as they are proprietary, commercial app stores use keyword-matching approaches and are based in search engines, such as Lucene [4], or in open-source solutions, such as Solr [20] and Elasticsearch [1], built on top of Lucene.

[1] `https://pt.aptoide.com/company/about-us`

In order to improve search results, several strategies have been explored. For example, Mobiwalla [5, 6], a search engine based on Lucene, uses natural language processing techniques such as stemming and lemmatization to create multiple versions of the original query and synonyms or hyponyms for generalization. Another strategy is to use topic information. Topic modeling captures the main themes of a collection of documents, thus improving search on that collection [2]. Zhuo et al. [24] enrich queries and apps representations with topics (based on the titles and descriptions of the applications) and tags (a filtered combination of human labels and tags obtained by crawling the web and usage data regarding each application) to improve semantic matching. Park et al. [14] also explore topic information by jointly modeling apps descriptions and reviews and generating apps representations based on their descriptions, using this topic model. In a subsequent work, Park et al. [13] explore social media data to model the implicit intention of a user query. They create a parallel corpus containing aligned text spans that associate explicit intentions to the corresponding implicit intentions (*[I want pizza]$_{explicit}$ because [I'm hungry]$_{implicit}$*). They use this data to infer the intention associated with a query and then use a relevance model to find the apps that match this intention. Ribeiro et al. [18] also use topic models to improve the semantic matching between the query and the apps. However, the focus is on how topic models (applied to the description of the apps) can be used to infer keywords to expand the queries sent to standard search engines.

Closer to our work, and given the successful use of word embeddings in several speech and language processing problems [7], recent work on retrieval tasks also focused in exploring these representations. For example, Yao et al. [22] train user specific word embeddings (using her/his personal data) and use them to compute representations of user queries and documents. The matching between queries and documents is done using a neural matching model. This was experimented in search logs. Samarawickrama et al. [19] also explore embeddings for searching in Twitter. They also train user specific word embeddings and use them for query expansion. Yates et al. [23] survey methods for text ranking (i.e., score a collection of textual documents with respect to a query) leveraging neural language models. They distinguish between dual-encoders and cross-encoders. The former encode queries and documents independently, performing better temporally, while the latter encode concatenations of queries and documents, generally obtaining better results, but not suitable to search over large collections, given its computational cost.

3 Data

The dataset used in the scope of this work was built from scratch, by scrapping Aptoide's API, and is publicly available[2] for reproducible purposes. A first endpoint was used to extract general data about applications, including the title, Aptoide identifier, added date, update date, and a set of statistics. The Aptoide identifier was then used to query two other endpoints for application-specific information. The second endpoint contains information regarding the developer, required permissions and the description. The third endpoint contains information about the categories associated with the application.

The first endpoint returned information about 499 thousand applications. For those, the second and third endpoints were queried, returning information for 436,969 of them. The observed discrepancy in values occurred, not only due to missing information on the API, but also due to privatization and/or discontinuation of applications.

[2] `https://apprecommender.caixamagica.pt/wordpress/resources/`

The relevancy statistics in our dataset include the number of downloads, the average rating, and the total number of ratings of each of application. An initial analysis revealed that the vast majority of the applications were not of much value. This is supported by the high number of applications with very few downloads (Figure 1), and by the high number of applications that have never been rated (414,053). For the applications that were rated at least once, the average rating distribution is depicted in Figure 2. Nonetheless, we considered the average rating not to be a very descriptive measure, due to an average number of total ratings of approximately 4.

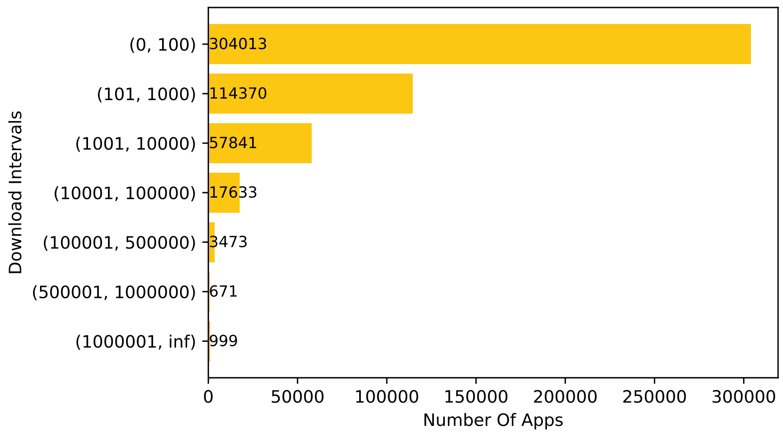

Figure 1 Number of applications within a interval of download values.

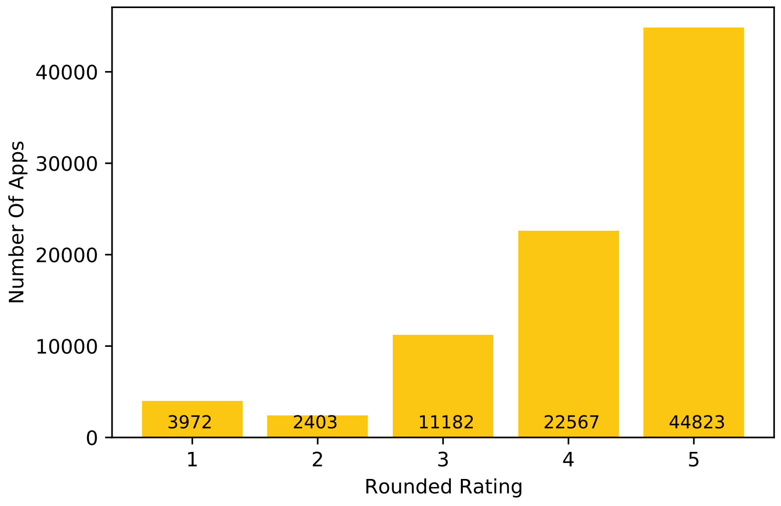

Figure 2 Rounded average rating distribution.

This way, we hypothesised that searching the whole dataset may not be ideal due to the irrelevant nature of the majority of the applications. As such, we derived a subset of relevant-only applications to search upon, based on four heuristics:

1. The top-5000 downloaded applications which were updated in the last 2 years;
2. The top-5000 downloaded applications which were updated in the last 6 months;
3. The top-1000 rated applications, with at least 200 rates and 1000 downloads;
4. The top-750 with more rates, added in the last 3 months, with at least 1000 downloads.

The objective was to consider applications that are widely used, not leaving out recent ones that are being updated constantly. The applications were also filtered to consider only those with descriptions in English. Since the information about the language is not included in our dataset, we used PyCLD3 (Python bindings for Google's CLD3 language detection model) to automatically detect it.

Overall, after removing duplicates, we ended up with 5,819 relevant applications. For each one of those applications, their name, description, downloads, average rating, total rating, added date and last update date were indexed in Elasticsearch.

The remaining non-relevant applications were stored in raw text files, since their textual data (name and description) constitute relevant resources for training our language models.

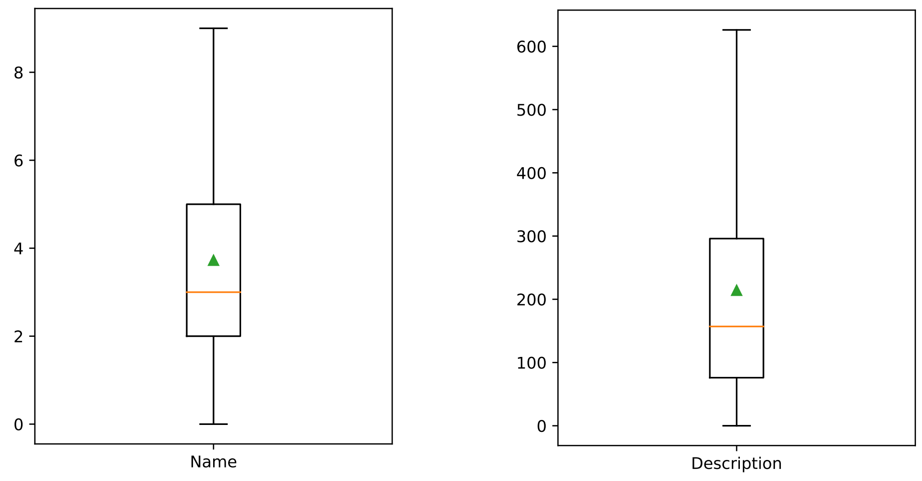

Figure 3 Distribution of the number of tokens for application's names (left) and application's descriptions (right). Green triangle represents the mean value, whereas the yellow line represents the median.

Figure 3 shows some basic statistics about the number of words that constitute both the application name and description, after removing some of the most salient outliers for visualization purposes. The figure reveals that the name of an application usually contains more than 2 tokens, and that it's description ranges from less than 100 tokens to more than 300 tokens. The following text is an example of an application description containing 111 words and multiple sentences.

> Don't Starve: Pocket Edition, brings the hit PC game enjoyed by over 6 million players to Android.
> Now you can experience the uncompromising wilderness survival game full of science and magic on the go! Play as Wilson, an intrepid Gentleman Scientist who has been trapped and transported to a mysterious wilderness world. Wilson must learn to exploit his environment and its inhabitants if he ever hopes to escape and find his way back home.
> Enter a strange and unexplored world full of strange creatures, dangers, and surprises. Gather resources to craft items and structures that match your survival style. Play your way as you unravel the mysteries of this strange land.

4 Approach

The main objective of our proposal was to compare semantic with lexical search in the context of mobile applications. Sections 4.2 to 4.3 introduce the models used to do so. For this, we consider both word-level and character-level word embeddings, to access which works better in this domain. Then, we compare both to contextual embeddings, generated through Transformer-based neural language models. Section 4.4 describes the indexing and searching mechanisms.

4.1 GloVe Word Embeddings

GloVe [15] is a model for unsupervised generation of static word embeddings. Alike Word2Vec [12], it uses local context window methods, but combines it with word-word correlation matrix factorization. We used a pre-trained model on textual data from Wikipedia and Gigatext, with 100-dimensional vectors.

To generate the embeddings for application's names, the strings are lower-cased and out-of-vocabulary words are removed. For descriptions, besides lower-casing and out-of-vocabulary word removal, stop-words words are removed. The models are used to generate word-level embeddings, and the average of the vectors is used as a final sentence embedding.

4.2 FastText Word Embeddings

FastText [3] follows an approach similar to Word2Vec [12], but each word is represented as a bag of character n-grams. This way, character-level embeddings are considered, instead of word-level representations.

In preliminary tests, we compared a CBOW [12] fastText model (FT1), pre-trained on English Common Crawl considering word 5-grams, to the aforementioned pretrained GloVe model. As the results favored the character-level embeddings, we trained a CBOW fastText model (FT2) from scratch using out textual data (see Section 3).

Since these models are quite big, we reduced the dimension of the vectors from 300 to 100, using the dimension reduction tool provided by fastText python library.

For both FT1 and FT2, the process to generate embeddings is the same, which is the default behaviour of fastText's embedding generation tool, with minor changes. For application's names, the strings are lower-cased. For descriptions, besides lower-casing, stop-words are also removed. The nature of these embeddings (character-level) allow for out-of-vocabulary words to be included. The models are used to generate word-level embeddings, which are then normalized by their L2 norm. The average of the vectors with positive L2 norm is taken as a final sentence embedding.

4.3 RoBERTa Contextual Embeddings

The main limitation of the previous models is that a word always has the same vector, despite the context of its usage. As such, we consider the usage of Contextual Embeddings, which are context-dependent representations that capture the use of words across multiple scenarios [10]. We explore the usage of a fine-tuned neural language model as a dual-encoder in the context of a mobile application search engine, since good results have been reported in other retrieval contexts [9, 16, 17], and dual-encoders make it possible to pre-compute and index representations for the application's textual data.

We start by fine-tuning RoBERTa$_{\text{base}}$ [11] on a masked language modelling task, using the Huggingface Transformers library [21]. We split our textual data (see Section 3) into train and test sets (90% and 10%, respectively). The text in the train set is processed into sets of 512 tokens by RoBERTa's original tokenizer, with a masking probability of 15%. The model was trained during 1 epoch with a batch size of 4, using a cross-entropy loss function. The test set was used to evaluate the fine-tuned model, which achieved a perplexity of 4.54 on a mask prediction task.

We further trained the model on a semantic similarity task, using the Sentence Transformers library [17]. For this, we synthetically build queries by concatenating an application's name with its categories. Then, each query is associated with the description of the application. This way, we build a collection of (query, relevant description) pairs. Within a training batch of size 8, triplets are built from the pairs, in the form (query, relevant description, fake description). The relevant description of a given query is used as the fake descriptions for the others, which allows for $8 \times 7 = 56$ training examples per batch. The model is used to generate representations for the query and the descriptions, and scores are obtained from the cosine similarity between the representations of the query and the descriptions. The objective is to minimize the negative log-likelihood of softmaxed scores. The whole textual dataset is used to train, except for 500 random applications which were used to evaluate. For those applications, queries were built as previously described, and a description corpus was built from their descriptions. The model was used as a dual-encoder to rank the 500 descriptions for each query. Given that each query only has one relevant description, we computed the Precision@1, the Recall@10, and the MRR@10. The results were 0.8795, 0.9732, and 0.9167, respectively.

Given a sentence as input, the final model (henceforth RoBERTapp) is used to generate representations through mean pooling of the word token embeddings of the last hidden-layer.

4.4 Indexing and Searching

The previous models were used to generate representations for the name and description of applications to be indexed (see Section 3). The representations were pre-computed, and stored along with the name and description of each application in an ElasticSearch index. The name and description are used to perform the classic lexical search. We built an interface to allow us to query this index, where we can choose which model to use, and which fields to consider (i.e., name, description, or both).

Searching with the lexical model uses the standard ElasticSearch analyser to process the query, which was also used to process the indexed textual data. Given a query q, the Lucene Scoring Function is used to compute the scores over the chosen fields, combining them if more than one:

$$\text{score}(q, a) = \sum_{f \in a} \text{LSF}(q, f) \;, \tag{1}$$

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

where f are application a's textual fields (in our case, name and/or description), t are the query's tokens, tf is the term-frequency, and idf is the inverse document frequency.

Searching with the vector-based models leverages ElasticSearch's built-in cosine similarity function. Representations for queries are generated at run-time, using the chosen model. Given a query q and an application a, the score is computed as follows:

$$\text{score}(q, a) = \alpha \,\text{sim}(\text{M}(q), \text{M}(a_n)) + \beta \,\text{sim}(\text{M}(q), \text{M}(a_d)) \,, \tag{3}$$

where M is the model encoding function, a_n is an application's name, a_d is an application's description, sim is the cosine similarity function, and α, β are combination weights. Note that $\text{M}(a_n)$ and $\text{M}(a_d)$ are already indexed, only $\text{M}(q)$ is generated at run-time.

Ultimately, given a query and a model, the scores are computed and the top-N scoring applications are retrieved.

5 Experiments and Results

Since our dataset does not include any queries and relevance judgments, we evaluate our models on the data provided by Park et al. [14], before manually analysing the results of the models on our indexed data.

This dataset contains information about 43,041 mobile applications including name and description. The dataset also features 56 non-exact queries (i.e., queries with a meaningful semantic context, instead of an application's name). For each one of the queries, 81 applications are labelled with a relevancy score of 0 (not relevant), 1 (somewhat relevant), or 2 (very relevant). These scores were manually annotated.

The authors used the Normalized Discounted Cumulative Gain as the evaluation metric, which takes the full order of the item list and graded relevances into account:

$$\text{DCG@}k = \text{R}(1) + \sum_{i=2}^{k} \frac{\text{R}(i)}{\log_2(i)} \,, \tag{4}$$

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k} \,, \tag{5}$$

where $\text{R}(i)$ is a function that returns the relevance value of the passage at rank i. The index of the passage up to which the ranking is considered is represented by k. The DCG is normalized with the ideal DCG (IDCG), i.e., the DCG of a perfectly sorted result.

Table 1 shows the results for the evaluation of our models, which was conducted under the same conditions as reported by Park et al. [14], i.e., ranking the 81 labeled applications for each one of the 56 queries.

For comparison, Table 2 shows results achieved by Google Play and LBDM reported by Park et al. [14], and results achieved by lexical models leveraging topic modelling techniques, reported by Ribeiro et al. [18], for the same scenario.

The results show that the pre-trained models (GloVe and FT1) performed worse than previous approaches. On the other hand, FT2 and the RoBERTapp models surpass previous approaches. We can also conclude that searching considering the name and description is, overall, the most advantageous combination. In this scenario, RoBERTapp achieved the best results for NDCG@{3,25}, and the best scores for NDCG@{5,10} were achieved by FT2.

RoBERTapp is the best model when dealing with descriptions only, perhaps due to the second fine-tuning task, which consisted in scoring descriptions based on a synthetic query. Conversely, FT2 is superior to RoBERTapp when searching with name only.

Since models FT2 and RoBERTapp performed well, we manually evaluated them on our relevant application index, considering names and descriptions. We query the index as described in Section 4.4. As expected, when the query corresponds or is close to an

Table 1 NDCG@{3,5,10,25} for the multiple models, considering application's name and description (N+D), only name (N), and only description (D).

	NDCG@3	**NDCG@5**	**NDCG@10**	**NDCG@25**
GloVe (N + D)	0.527	0.523	0.522	0.538
GloVe (N)	0.523	0.514	0.512	0.529
GloVe (D)	0.504	0.491	0.489	0.508
FT1 (N + D)	0.540	0.521	0.532	0.543
FT1 (N)	0.512	0.507	0.513	0.529
FT1 (D)	0.462	0.466	0.461	0.476
FT2 (N + D)	0.587	**0.589**	**0.582**	0.600
FT2 (N)	0.595	0.582	0.571	0.582
FT2 (D)	0.519	0.529	0.519	0.545
RoBERTapp (N + D)	**0.616**	0.587	0.581	**0.605**
RoBERTapp (N)	0.582	0.570	0.568	0.590
RoBERTapp (D)	0.585	0.581	0.577	0.585

Table 2 NDCG@{3,5,10,25} achieved by Google Play and LBDM [14], and achieved by lexical models considering applications' description and topics [18].

	NDCG@3	**NDCG@5**	**NDCG@10**	**NDCG@25**
LBDM	0.584	0.563	0.543	0.565
Google Play	0.589	0.575	0.568	0.566
BM25F	0.574	0.542	0.527	0.544
ElasticSearch	0.552	0.532	0.504	0.519

application name, none of the models had problems retrieving the correct applications. Since our objective is to enrich a search engine with semantic capabilities, we further tested with non-exact queries. Table 3 shows the results for three example queries. Note that this search was conducted in the setup described in Section 4.4, considering only the embeddings generated by the models. This is, no other information regarding relevance (e.g., downloads, ratings, etc...) was considered, so as to better access the usefulness of semantic models.

The semantic models were able to retrieve relevant applications which fit the scope of the non-exact queries. For example, searching for "social network" with RoBERTapp returned the most widely-used social networks. One can argue that FT2 worked better than RoBERTapp for the query "airline tickets", since the latter returned a game as the top-result. Still, overall, both models provide more appropriate results than the lexical model.

6 Conclusions and Future Work

This paper proposes a set of approaches for semantic search of mobile applications, which use clues contained in the name and textual descriptions of these applications for selecting the most relevant ones for a given query. Our approaches are based on word-level (GloVe), character-level (fastText), and on contextual (RoBERTa) word-embeddings. We have described the process of collecting a dataset, containing information about mobile apps, that was further described and used for training and fine-tuning our models. The proposed approaches have been evaluated using a publicly available dataset of mobile applications, and the results achieved show that both character-level embeddings, trained on our data, and fine-tuned RoBERTa models, fine-tuned also using our data, when applied in an unsupervised way,

Table 3 Comparison between the lexical model and the semantic models (FT2, RoBERTapp) for non-exact queries, considering name and description. The ordered top-5 results are shown.

Query: "social network"

Lexical	**RoBERTapp**	**FT2**
Hornet - Social Network	Facebook	Peeks Social
BandLab – Music Studio & Social Network	Instagram	Network Browser
Pi Network	Twitter	Hornet - Social Network
Network Browser	Internet	Cartoon Network App
Peeks Social	Facebook Viewer	Air-Share

Query: "food at home"

Lexical	**RoBERTapp**	**FT2**
Domino's Pizza - Online Food Delivery App	EatSure - Food Delivery | Order Food Now!	foodpanda - Local Food & Grocery Delivery
Mixer – Interactive Streaming	foodpanda: Fastest food delivery, amazing offers	EatSure - Food Delivery | Order Food Now!
Trendyol - Online Shopping	foodpanda - Local Food & Grocery Delivery	Swiggy Food Order & Delivery
DoorDash - Food Delivery	DoorDash - Food Delivery	Zomato - Online Food Delivery & Restaurant Reviews
foodpanda - Local Food & Grocery Delivery	Toca Kitchen Sushi Restaurant	Glovo: Order Anything. Food Delivery and Much More

Query: "airline tickets"

Lexical	**RoBERTapp**	**FT2**
Privat24	Airline Commander - A real flight experience	MakeMyTrip-Flights Hotels Cabs
Trip.com: Flights, Hotels, Train	Southwest Airlines	Cleartrip - Flights, Hotels, Train Booking
OpenSooq	American Airlines	ebookers - Hotel, Flight, Car Hires
Flüge.de	Flightradar24 Flight Tracker	Goibibo - Hotel Car Flight
KAYAK flights, hotels & car hire	MakeMyTrip-Flights Hotels Cabs	Trip.com: Flights, Hotels, Train

surpass the performance of the other existing retrieval strategies reported in the literature. We further confirmed that the proposed semantic-related models capture the scope of non-exact queries, which lexical models struggle to do, by manually searching over our relevant applications index.

In the near future, we are planning to improve our RoBERTa model by considering additional unsupervised/semi-supervised training tasks. The reported experiments use only the similarity between the query and each one of the candidate applications. We are also planning to create a multi-criteria retrieval system that takes into account other relevant information, such as the number of downloads, rating and the fact that a given application was updated recently.

References

1 S. Banon. Elasticsearch, 2010. URL: `https://www.elastic.co/`.

2 David M. Blei. Probabilistic Topic Models. *Commun. ACM*, 55(4):77–84, 2012. `doi:10.1145/2133806.2133826`.

3 Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics*, 5:135–146, 2017. URL: `https://transacl.org/ojs/index.php/tacl/article/view/999`.

4 D. Cutting. Apache Lucene, 1999. URL: `https://lucene.apache.org/`.

5 Anindya Datta, Kaushik Dutta, Sangar Kajanan, and Nargin Pervin. Mobilewalla: A Mobile Application Search Engine. In Joy Ying Zhang, Jarek Wilkiewicz, and Ani Nahapetian, editors, *Mobile Computing, Applications, and Services*, pages 172–187, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

6 Anindya Datta, Sangaralingam Kajanan, and Nargis Pervin. A Mobile App Search Engine. *Mobile Networks and Applications*, 18, 2013.

7 Sahar Ghannay, Benoit Favre, Yannick Estève, and Nathalie Camelin. Word Embedding Evaluation and Combination. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 300–305, Portorož, Slovenia, 2016. European Language Resources Association (ELRA). URL: `https://www.aclweb.org/anthology/L16-1046`.

8 Mansoor Iqbal. App download and usage statistics (2020). web page, October 2020. URL: `https://www.businessofapps.com/data/app-statistics/`.

9 Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 39–48. ACM, 2020. `doi:10.1145/3397271.3401075`.

10 Qi Liu, Matt J. Kusner, and P. Blunsom. A Survey on Contextual Embeddings. *ArXiv*, abs/2003.07278, 2020. `arXiv:2003.07278`.

11 Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. `arXiv:1907.11692`.

12 Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the International Conference on Learning Representations*, 2013.

13 Dae Hoon Park, Yi Fang, Mengwen Liu, and ChengXiang Zhai. Mobile App Retrieval for Social Media Users via Inference of Implicit Intent in Social Media Text. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, page 959–968, New York, NY, USA, 2016. Association for Computing Machinery. `doi:10.1145/2983323.2983843`.

14 Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. Leveraging User Reviews to Improve Accuracy for Mobile App Retrieval. In Ricardo Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto, editors, *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 533–542. ACM, 2015. `doi:10.1145/2766462.2767759`.

15 Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global Vectors for Word Representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014. `doi:10.3115/v1/d14-1162`.

16 Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering. *CoRR*, abs/2010.08191, 2020. `arXiv:2010.08191`.

17 Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, November 2019. `arXiv:1908.10084`.

18 Eugénio Ribeiro, Ricardo Ribeiro, Fernando Batista, and João Oliveira. Using Topic Information to Improve Non-exact Keyword-Based Search for Mobile Applications. In Marie-Jeanne Lesot, Susana Vieira, Marek Z. Reformat, João Paulo Carvalho, Anna Wilbik, Bernadette Bouchon-Meunier, and Ronald R. Yager, editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 373–386, Cham, 2020. Springer International Publishing.

19 Sameendra Samarawickrama, Shanika Karunasekera, Aaron Harwood, and Ramamohanarao Kotagiri. Search Result Personalization in Twitter Using Neural Word Embeddings. In Ladjel Bellatreche and Sharma Chakravarthy, editors, *Big Data Analytics and Knowledge Discovery*, pages 244–258, Cham, 2017. Springer International Publishing.

20 Y. Seeley. Apache Solr, 2004. URL: `https://lucene.apache.org/solr/`.

21 Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, 2020. Association for Computational Linguistics. URL: `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

22 Jing Yao, Zhicheng Dou, and Ji-Rong Wen. Employing Personal Word Embeddings for Personalized Search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1359–1368, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3397271.3401153`.

23 Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. Pretrained transformers for text ranking: BERT and beyond. In Liane Lewin-Eytan, David Carmel, Elad Yom-Tov, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*, pages 1154–1156. ACM, 2021. `doi:10.1145/3437963.3441667`.

24 Juchao Zhuo, Zeqian Huang, Yunfeng Liu, Zhanhui Kang, Xun Cao, Mingzhi Li, and Long Jin. Semantic Matching in APP Search. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, page 209–210, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2684822.2697046`.