‘It’s on the tip of my tongue’: A new Dataset for Known-Item Retrieval

Samarth Bhargav  
IRLab, University of Amsterdam  
The Netherlands

Georgios Sidiropoulos  
IRLab, University of Amsterdam  
The Netherlands

Evangelos Kanoulas  
IRLab, University of Amsterdam  
The Netherlands

ABSTRACT
The tip of the tongue known-item retrieval (TOT-KIR) task involves the ‘one-off’ retrieval of an item for which a user cannot recall a precise identifier. The emergence of several online communities where users pose known-item queries to other users indicates the inability of existing search systems to answer such queries. Research in this domain is hampered by the lack of large, open or realistic datasets. Prior datasets relied on either annotation by crowd workers, which can be expensive and time-consuming, or generating synthetic queries, which can be unrealistic. Additionally, small datasets make the application of modern (neural) retrieval methods unviable, since they require a large number of data-points. In this paper, we collect the largest dataset yet with 15K query-item pairs in two domains, namely, Movies and Books, from an online community using heuristics, rendering expensive annotation unnecessary while ensuring that queries are realistic. We show that our data collection method is accurate by conducting a data study. We further demonstrate that methods like BM25 fall short of answering such queries, corroborating prior research. The size of the dataset makes neural methods feasible, which we show outperforms lexical baselines, indicating that neural/dense retrieval is superior for the TOT-KIR task.

1 INTRODUCTION
The tip of the tongue known-item retrieval (TOT-KIR) task involves a user searching for an item which they had previously encountered, for which they are unable to recall the precise identifier [2]; for instance, they are unable to recall the name of a movie viewed several years ago, characterized by the phrase ‘it lies on the tip of my tongue’. More precisely, the task involves the retrieval of a single item from a potentially large collection of items, given a descriptive query with possibly imprecise information about the item, along with other information such as the circumstances of encountering the item, the sentiment/emotion associated with it, the medium it was encountered in, etc [2, 7–9]. While this is similar to Known-Item Retrieval (KIR) or item re-finding, we note that TOT-KIR is distinct from KIR: in TOT-KIR the identifier is unknown, the item is searched after a much longer period of time after encountering it, the requests contain imprecise/incorrect information i.e false memories [7–9] and the queries are verbose [2].

This task is particularly hard for existing search systems, which is evidenced by the emergence and subsequent popularity of several online communities1 that allow users to pose TOT-KIR queries. The challenging nature of this task is further evidenced by references to previous search attempts in these posts [2], highlighting the need for datasets and methods to tackle this problem.

This difficulty may stem from the nature of the query itself. For instance, most queries are verbose compared to typical searches, and may contain imprecise or incorrect descriptions from a user’s (typically) faint recollection of the item. Indeed, the incidence of false memories has been previously studied for KIR [2, 7–9]. Furthermore, the query may contain terms that might not be useful or even harmful for retrieval performance, such as previous search attempts, the context or medium it was encountered in, or hedging words like ‘probably’, ‘maybe’, etc [2]. In some cases, users mention other items to inform other users that the known-item are not among them. In addition, the text may contain indirect references (‘actor that looks like Tom Cruise’) which are not resolvable using methods like BM25, requiring more advanced methods [2]. Research in TOT-KIR is impeded by the lack of large scale or open collections [8, 9].

We make two contributions in this paper. First, we describe a process to extract known-item queries and their corresponding ‘gold’ items using a heuristic, from the Tip of my Tongue sub-reddit (https://www.reddit.com/r/tipofmytongue/), a sub-community in the social media website Reddit. We collect this dataset with a focus on precision i.e gather only queries for which we can get the ‘gold’ known-item with a high confidence, instead of gathering as

1Other sub-reddits like https://www.reddit.com/r/whatsthatbook/, or online communities such as https://irememberthismovie.com/ [2]
I couldn’t have been older than 4, so this was around 2002. I watched a movie with my parents (so I thought) and despite never watching it again, it became my favorite. It centered around a middle-aged man who went on some kind of adventure and turned into a fish. I also think I recall him visiting a school of some sort. It seemed like a slightly old movie, but it was in color and began with real actors and changed to animation. For weeks after I saw this movie I told my parents about it, but they insisted it was a dream so I let it go. Does anyone know what this movie is?

Figure 1: A popular post in the subreddit, containing a title (top, bold text), a flair (in green) and a description (bottom). A subset of accompanying comments are shown in Figure 2. The full post can be viewed here.
We first filtered submissions based on the flair text. By default, a...

Figure 2: Two comment trees, one containing the gold answer, and one containing a suggestion rejected by the user - which consider as a ‘negative’

3.1 Filtering

We first filtered submissions based on the flair text. By default, a...

3.2 Extracting Gold Answers

Since a typical ‘Solved’ post has a canonical comment by the original poster, we design a heuristic to extract gold answers from solved requests, outlined in Algorithm 1. Our approach can be summarized as: (1) traverse the comment trees to find the reply which indicates that the author has accepted the answer (‘Solved!’) (2) construct a path from this node to the root of the tree (3) extract query-item pairs by picking requests with a single candidate. The URLs are examined only if they are from (1) Wikipedia (2) IMDb (for Movies) (3) GoodReads (for Books) with each such URL considered.

---

3. Along with other items, is described in Section 3.2. Since this process is automatic, we perform a quality check to verify the efficacy of our heuristic, which is described in Section 3.3. The following paragraph introduces the community and describes the structure of a typical request, called a submission.

The /r/TipOfMyTongue community is a sub-community where users post TOT requests (referred to as a submission or post) which are then answered by other users in the community, sometimes in a collaborative manner. The user who makes the submission, called the ‘Original Poster’, provides a title and a description. An example post is shown in Figure 1, which has a flair assigned to it by the moderators. Suggestions by other users form a comment forest, where the roots of the trees correspond to top-level replies. Two such trees are shown in Figure 2. We use all of these fields to filter and then extract query and known-item pairs.

3.2 Extracting Gold Answers

Since a typical ‘Solved’ post has a canonical comment by the original poster, we design a heuristic to extract gold answers from solved requests, outlined in Algorithm 1. Our approach can be summarized as: (1) traverse the comment trees to find the reply which indicates that the author has accepted the answer (‘Solved!’) (2) construct a path from this node to the root of the tree (3) extract query-item pairs by picking requests with a single candidate. The URLs are examined only if they are from (1) Wikipedia (2) IMDb (for Movies) (3) GoodReads (for Books) with each such URL considered.

---

3. Along with other items, is described in Section 3.2. Since this process is automatic, we perform a quality check to verify the efficacy of our heuristic, which is described in Section 3.3. The following paragraph introduces the community and describes the structure of a typical request, called a submission.

The /r/TipOfMyTongue community is a sub-community where users post TOT requests (referred to as a submission or post) which are then answered by other users in the community, sometimes in a collaborative manner. The user who makes the submission, called the ‘Original Poster’, provides a title and a description. An example post is shown in Figure 1, which has a flair assigned to it by the moderators. Suggestions by other users form a comment forest, where the roots of the trees correspond to top-level replies. Two such trees are shown in Figure 2. We use all of these fields to filter and then extract query and known-item pairs.

3.2 Extracting Gold Answers

Since a typical ‘Solved’ post has a canonical comment by the original poster, we design a heuristic to extract gold answers from solved requests, outlined in Algorithm 1. Our approach can be summarized as: (1) traverse the comment trees to find the reply which indicates that the author has accepted the answer (‘Solved!’) (2) construct a path from this node to the root of the tree (3) extract query-item pairs by picking requests with a single candidate. The URLs are examined only if they are from (1) Wikipedia (2) IMDb (for Movies) (3) GoodReads (for Books) with each such URL considered.

---

3. Along with other items, is described in Section 3.2. Since this process is automatic, we perform a quality check to verify the efficacy of our heuristic, which is described in Section 3.3. The following paragraph introduces the community and describes the structure of a typical request, called a submission.

The /r/TipOfMyTongue community is a sub-community where users post TOT requests (referred to as a submission or post) which are then answered by other users in the community, sometimes in a collaborative manner. The user who makes the submission, called the ‘Original Poster’, provides a title and a description. An example post is shown in Figure 1, which has a flair assigned to it by the moderators. Suggestions by other users form a comment forest, where the roots of the trees correspond to top-level replies. Two such trees are shown in Figure 2. We use all of these fields to filter and then extract query and known-item pairs.

3.2 Extracting Gold Answers

Since a typical ‘Solved’ post has a canonical comment by the original poster, we design a heuristic to extract gold answers from solved requests, outlined in Algorithm 1. Our approach can be summarized as: (1) traverse the comment trees to find the reply which indicates that the author has accepted the answer (‘Solved!’) (2) construct a path from this node to the root of the tree (3) extract query-item pairs by picking requests with a single candidate. The URLs are examined only if they are from (1) Wikipedia (2) IMDb (for Movies) (3) GoodReads (for Books) with each such URL considered.
as a candidate for the gold answer. As this heuristic may extract multiple candidates, only submissions with a single candidate were considered, which is justified since our goal is to gather accurate query-item pairs in an automated manner. We leave the utilization of submissions with multiple candidates to future work.

Each candidate consists of the following: (1) an identifier uniquely identifying the known-item (2) a title associated with the item (3) a description of the item (4) additional meta data (if applicable). The following two paragraphs detail the data extracted for each subset, followed by a discussion of other domains and the extraction of other items to make the task more realistic.

### 3.2.1 Movies

The IMDb ID was used as the identifier for Movies, since this is referred to most frequently in the comments. If, instead, a Wikipedia URL was found, the IMDb ID (property P345) was obtained using the corresponding WikiData ID associated with the URL. Similarly, if the URL was an IMDb ID, we linked this movie back to a Wikidata entry, while ensuring that the IMDb ids from both sources matched. The title and description of the movie were extracted from Wikipedia (using Wikiplots, similar to [2]), if available, otherwise from IMDb.

### 3.2.2 Books

We used the GoodReads ‘Work ID’ as the identifier for Books, instead of ISBN-10/13 since books can have multiple editions. If a Wikipedia URL was available instead, we extracted the ISBN-10/13 using the corresponding Wikidata entity, and linked it back to GoodReads (and vice versa). We utilized the Book Graph dataset [23, 24] (extracted in late 2017) for matching ISBNs to work IDs and for extracting titles and descriptions. Additional information such as reviews could also be used towards improving retrieval performance. While we focus on Books and Movies, it is straightforward to extract similar data for other domains.

### 3.2.3 Other domains

The subreddit has several other categories not considered in this work. For instance, there were 50,163 solved submissions corresponding with the (free-text) categories ‘song’ and ‘music’. Creating a dataset for other domains is straightforward and involves (1) a method to map a URL to an identifier (and optionally link this to other sources like Wikipedia) (2) methods to extract data for a particular item, or to link it to existing data (i.e get_data in Algorithm 1).

### 3.2.4 Other candidates and negatives

In addition to the query-item pairs described above, we gathered other items that were not selected as ‘gold’ for any query. These were sourced from candidates which were not selected as gold answers, including those that were discarded in the filtering step (i.e submissions with > 1 candidates found in get_answers). These items increase the number of candidates in the retrieval pool, making the task more realistic. In addition to these, we extracted negatives sourced from submissions where a query-item pair was extracted. An item is considered a negative for a given query, if it was proposed as an answer but ended up being rejected by the user. These items are termed negatives since other users confused them with the ‘gold’ item. These items were not extracted from the comment tree containing the accepted answer. As there were too few negatives for Books, we did not include them in the data or experiments. The resulting dataset statistics are reported in Table 2. Note that some queries have the same item, which is why the documents are fewer than the queries. In Reddit-TOMT, the most frequently requested item is ‘Mindhunters’ (21 times!) in Movies, ‘The Transall Saga’ (8 times) in Books.

### 3.3 Data Quality Checks

Three aspects of the data were examined in this study: (1) Heuristic Accuracy: whether the answer picked by the heuristic matched the answer picked by the author of the post (2) Data Leakage: whether the answer itself was in the text (3) Malformed Query: Whether

---

*These websites were selected after a preliminary analysis showed that these were the most frequent.*
the query was usable e.g if it contained only a URL, it was considered unusable. This happens if the query pointed to an image, or a screenshot of a movie. Two of the authors first independently labeled the data using Label Studio, and then resolved disagreements, on 3 subsets: QABooks, 100 randomly selected data points from the Books; QAMovies>1, 100 randomly selected data points from Movies, where the depth of the reply containing the answer is greater than one; QABooks, 100 randomly selected data points from the Books subset. As there were too few data points where the answer positions were greater than one in the Books subset, we didn’t conduct a study similar to QAMovies>1 for Books. The results are reported in Table 1 and discussed below.

3.3.1 Heuristic Accuracy. The accuracy for QAMovies>1 (96%) is slightly lower than that for QAMovies (98%), which indicates that the heuristic works slightly worse for answers located deeper in a comment tree. This is likely to happen only in cases where the original poster rejects an answer containing a URL but accepts a plain-text answer (no URL) in a deeper reply, which is not picked up by the heuristic. However, we noted there are relatively few posts where the answer is located deeper in the comment tree. We note that the overall accuracy of 97.6% might be under-estimated, since most of the answers were found at the root of the comment tree (99.9% of Books and 98.9% of Movies).

3.3.2 Data Leakage. The data seem to have very few answers in the query itself. To prevent any data leakage (i.e. the name of the answer is available in the test), we picked only posts with edited descriptions (see footnote in previous page), and examined these to check if they had tokens from the title of the known-item in the last sentence. If such a token occurred, the last sentence is removed and the description is updated. We note that there were only 5 such descriptions (0.22%) in Books, and 66 (0.49%) in Movies.

3.3.3 Malformed Query. We removed queries which had either an empty description, or have a short description with only a URL. This resulted in 628 (4.45%) and 61 (2.56%) queries being removed from Movies and Books respectively.

Despite the efficacy of our approach, it is important to note its limitations, which we outline here. First, users only post requests to these communities only if their TOT queries fail using existing search engines, which creates a sampling bias in the types of posts. However, this is an unavoidable bias inherent in approaches that source data from communities like Reddit [11], Yahoo Answers [7] or other community websites [2]. Second, since we filter out unsolved queries, it contains only queries which can be resolved i.e Reddit-TOMT cannot reveal why TOT queries fail, which we believe can be important for a deeper understanding of the task. In addition, the filtering process considerably reduces the number of queries e.g using submissions with links may lead to several queries (without URLs) being missed out. This can happen if the user uses plain text without linking to these websites, for instance. While we attempted to capture the latter using entity linking, the large number of false positives (even after filtering out entities without IMDb IDs for Movies) made this approach unviable without annotation, which is contrary to our goal. In addition, since we didn’t consider replies made by users in subsequent interactions with other users, some queries themselves might be ‘unsolvable’ without taking these clarifications into account. However, as these submissions are in the minority, this issue is minimized. Finally, we note that the sub-reddit contains only English posts.

Table 1: Results of the data study outlined in Section 3.3. Overall accuracy is 97.6%, highlighting the efficacy of the heuristic outlined in Algorithm 1.

<table>
<thead>
<tr>
<th>QA Set</th>
<th>Question</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>QAMovies&gt;1</td>
<td>Correct/Incorrect Answer</td>
<td>96 / 4</td>
</tr>
<tr>
<td>QAMovies</td>
<td>Well-formed / Malformed Query</td>
<td>86 / 14</td>
</tr>
<tr>
<td>QABooks</td>
<td>Answer in submission / Not in submission</td>
<td>99 / 1</td>
</tr>
<tr>
<td>QABooks</td>
<td>Correct/Incorrect Answer</td>
<td>99 / 1</td>
</tr>
<tr>
<td>QAMovies</td>
<td>Well-formed / Malformed Query</td>
<td>97 / 3</td>
</tr>
<tr>
<td>Books</td>
<td>Answer in submission / Not in submission</td>
<td>100 / 0</td>
</tr>
</tbody>
</table>

Algorithm 1: Heuristic for finding the ‘gold’ known item from a comment forest, given the author of the request

```
function get_solved_node(comment_forest, original_poster)
    for comment_tree in comment_forest do
        reply_stack = [comment_tree];
        while reply_stack is not empty do
            reply = reply_stack.pop();
            if reply.author == original_poster then
                if reply contains 'Solved' then
                    return comment_tree, reply;
                end
            end
        end
    end

function get_answers(submission)
    comment_tree, solved_node = solvedPath(submission, comment_forest, submission.author);
    urls = extract_urls_to_path(comment_tree, solved_node);
    candidates = [];
    for url in urls do
        if url domain is wikipedia or imdb (or goodreads) then
            gold_answer = get_data(url);
            candidates.append(gold_answer);
        end
    end
    return candidates;
```
We first split the data into a train (80%), validation (10%) and test set (10%). We tested the performance of a state-of-the-art dense retriever as well as traditional lexical-based ones on Reddit-TOMT. We used BM25 [20] and PL2 [1], two lexical retrieval methods, and Dense Passage Retrieval [12], a dense retrieval method.

4.1 Benchmarks

We tested the performance of a state-of-the-art dense retriever as well as traditional lexical-based ones on Reddit-TOMT. We used BM25 [20] and PL2 [1], two lexical retrieval methods, and Dense Passage Retrieval [12], a dense retrieval method.

4.1.1 Sparse Retrieval Benchmarks. BM25 [20] is a standard baseline in IR tasks, and is considered difficult to outperform. PL2, a divergence from randomness (DFR) model, is the 2-Poisson model with Laplace after-effect, with normalization [1], which may be suited for early-precision tasks [10, 19]. For both baselines, we used the py-Terrier framework’s implementations. The data was processed using Terrier’s default processor, which removes stop words and stems the tokens using the Porter stemmer.

4.1.2 Dense Retrieval Benchmarks. In contrast to traditional methods which treat each document as a bag of tokens, dense retrieval conducts retrieval in a dense representation space, by encoding each document/query into a latent space. By projecting documents into a latent space, models can account for ‘noise’, for example, synonymy, allowing for retrieval based on semantics rather than lexical overlap. We used Dense Retrieval (DR) [12, 25] as a benchmark for this purpose. DR is a passage retrieval model where dense representations are learned from pairs of questions and passages (or answers) by a dual-encoder model, without any additional pre-training. Given a question \( q \), alongside a relevant passage \( p^r \) and a set of irrelevant passages \( \{p_1, p_2, \ldots, p_m\} \), the model learns to rank relevant passages higher by optimizing the negative log likelihood (NLL) of the positive passage. In our problem, a relevant passage is the description of the ‘gold’ known-item, and negative passages are other items. The following section describes the experimental setup.

4.2 Experimental Setup

We first split the data into a train (80%), validation (10%) and test set (10%), by randomly sampling query-item pairs, both for Books and Movies separately. We index the descriptions of each candidate from each document/query into a latent space. By projecting documents and queries into a latent space, models can account for ‘noise’, for example, synonymy, allowing for retrieval based on semantics rather than lexical overlap. We used Dense Retrieval (DR) [12, 25] as a benchmark for this purpose. DR is a passage retrieval model where dense representations are learned from pairs of questions and passages (or answers) by a dual-encoder model, without any additional pre-training. Given a question \( q \), alongside a relevant passage \( p^r \) and a set of irrelevant passages \( \{p_1, p_2, \ldots, p_m\} \), the model learns to rank relevant passages higher by optimizing the negative log likelihood (NLL) of the positive passage. In our problem, a relevant passage is the description of the ‘gold’ known-item, and negative passages are other items. The following section describes the experimental setup.

### Table 2: Overall statistics of Reddit-TOMT

<table>
<thead>
<tr>
<th>Subset</th>
<th>No. queries</th>
<th>No. positive documents</th>
<th>No. other candidates</th>
<th>No. negatives</th>
<th>Total documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2319</td>
<td>1910</td>
<td>710</td>
<td>-</td>
<td>2620</td>
</tr>
<tr>
<td>Movies</td>
<td>13494</td>
<td>8845</td>
<td>4797</td>
<td>1221</td>
<td>14863</td>
</tr>
<tr>
<td>Total</td>
<td>15788</td>
<td>10755</td>
<td>5507</td>
<td>1221</td>
<td>17483</td>
</tr>
</tbody>
</table>

For BM25, we tuned the \( c, k_1 \) and \( k_3 \) parameters. We performed grid search using the following values: \( c \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\} \), \( k_1 \in \{0.3, 0.6, 0.9, 1.2, 1.4, 1.6, 2\} \), and \( k_3 \in \{0.5, 2, 4, 6, 8, 10, 12, 14, 20\} \). For these methods, which do not involve training a model, we tuned the hyper-parameters on the train-validation sets, and report the performance on the test set.

We trained DR on the training set for 25 epochs on 2 \( \times \) 12GB GPUs. Instead of using a bi-encoder architecture with separately parameterized query and passage encoders, we share the parameters for both. In preliminary experiments, we found that this performed better, perhaps due to the descriptive nature of the queries. We used a RoBERTa-based encoder, an Adam optimizer with a learning rate of \( 2e - 5 \), and linear scheduling with a warm-up ratio of 0.1. We trained using in-batch BM25 negatives, with a batch size of 4 and one additional negative per question. Finally, the maximum passage and query length was limited to 512; truncating anything beyond this length. Hard negatives for DR were obtained using BM25 (default parameters). In addition, since we extracted negatives from the data (as opposed to BM25), we experimented with using these in place of BM25 negatives in the training process. In this setting, which we term DR\( p_{BM25} \), if a negative from Reddit-TOMT wasn’t available, BM25 negatives were used instead. We note that only negatives were available only for 1444 queries in the training set.

We evaluate the benchmarks on the following metrics, after truncating the ranking list to 1000:

- \( \text{Recall}@1 \) (R@1) and \( \text{Recall}@10 \) (R@10) (referred to as Success@1/10 in [2]): Which measures the ability of a method to return the correct result in the top 1 or 10 list respectively. This follows Arguello et al. [2].
- \( \text{Mean Reciprocal Rank (MRR)} \): MRR is the average of the reciprocal ranks of results. Higher numbers indicate that known-items were positioned higher in the ranking.

\( R@K \) measures whether a user manages to find the item in the initial ranks, while MRR measures effort, since the user might have to peruse items further down in the list. We report both the mean and the standard deviation across all queries for each metric. The following section reports and discusses the results of these benchmarks.

5 BENCHMARK RESULTS

The overall results are reported in Table 3. From the results, we can conclude the following: (1) Lexical methods like BM25 may be inadequate for known-item search, corroborating prior findings [2] on a larger dataset dataset (2) DR outperforms all lexical methods, indicating dense methods may be better suited for TOT-KIR. We go over these results in the following paragraphs.

We first note that BM25 outperforms PL2 in all settings. We further note that the performance on \( R@10 \) using BM25, 0.3433, is higher compared to the performance noted in Arguello et al. [2] - 0.1327 another TOT-KIR dataset.
while DR outperforms BM25 by a large margin, indicating we can’t make a broader conclusion due to the limited number of

This indicates lexical methods fail to retrieve the correct known items, instead ranking negatives higher for some queries. While we can’t make a broader conclusion due to the limited number of queries with negatives, it might be that users are driven to these communities precisely because the wrong movies are retrieved by existing search engines. This is further evidenced by some users responding to the current query. Both use the default parameters for BM25 and PL2, highlighting that about a third of the queries can be resolved by a user perusing the top-10 results. Despite this, if we consider the typical scenario where a user views only the first few results, BM25 falls short: the majority (i.e. approximately 70%) of queries seemingly cannot be answered using BM25/PL2. We note however, that the capability of existing search engines to resolve TOT-KIR queries is unknown, so a broader conclusion about its efficacy on queries cannot be made (see limitations in Section 3.3).

Initial experiments showed that DR (using a pretrained RoBERTa [17]) without training on Reddit-TOMT, achieves very poor performance, with a R@1 of 0.0067 (0.0815), R@10 of 0.0223(0.1476) and a MRR of 0.0131(0.0903). This further motivates Reddit-TOMT, since deep models typically require a large training set to perform effectively.

Unsurprisingly, DR outperforms BM25 by a large margin for Books and a smaller but still large margin for Movies. For Books, DR retrieves the correct known-item in the top-10 for almost half of the queries, and for Movies this is reduced to a third. On average, the correct known-item is also placed higher in the result list, as evidenced by higher MRR scores.

DRHN, which uses negatives from the dataset instead of the default BM25 negatives, performs slightly worse than DR, but is still better than lexical methods. DRHN has a lower R@1 of 0.1248 (0.3305) and a lower MRR of 0.1888 (0.3295), but has a marginally better recall of 0.3210 (0.4668) for Movies. While the performance differences are insignificant, we hypothesize that this might be due to multiple factors: First, only a fraction of the queries in the training set have negatives and access to additional negatives might change the results; second, since negatives are extracted from user comments, they might be noisy, making distinguishing between negatives and the correct item trivial, leading to little or no learning; finally other factors introduced in training deep models (like choice of model, etc) might also be a factor.

To further investigate the effort required to find items in a list of results, we compute and plot recall at multiple cut-offs in Figure 4. It is evident that DR outperforms BM25 and PL2, highlighting the inability of lexical methods to retrieve the correct result. We conclude that Dense retrieval is more effective at the known-item retrieval task. The following paragraph describes examples where either BM25/DR excel or fail.

Qualitative Analysis. Table 4 contains examples from the dataset. We note that BM25 excels at retrieving items with very discriminative words like ‘geyser’ (1) or ‘atticus’ (2). However, since false memories are common in KIR [7–9, 11], discriminative terms, e.g. names, may lead to failure. Furthermore, the presence of hedging terms or otherwise imprecise descriptions aren’t handled directly

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>R@1</th>
<th>R@10</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>BM25</td>
<td>0.1416 (0.3487)</td>
<td>0.3133 (0.4638)</td>
<td>0.1971 (0.3444)</td>
</tr>
<tr>
<td></td>
<td>PL2</td>
<td>0.1350 (0.3396)</td>
<td>0.2981 (0.4546)</td>
<td>0.1873 (0.3363)</td>
</tr>
<tr>
<td></td>
<td>DR</td>
<td>0.1974 (0.3981)</td>
<td>0.4206 (0.4937)</td>
<td>0.2783 (0.3835)</td>
</tr>
<tr>
<td>Movies</td>
<td>BM25</td>
<td>0.1152 (0.3192)</td>
<td>0.2697 (0.4438)</td>
<td>0.1679 (0.3226)</td>
</tr>
<tr>
<td></td>
<td>PL2</td>
<td>0.0973 (0.2964)</td>
<td>0.2221 (0.4157)</td>
<td>0.1423 (0.3009)</td>
</tr>
<tr>
<td></td>
<td>DR</td>
<td>0.1285 (0.3347)</td>
<td>0.3180 (0.4657)</td>
<td>0.1938 (0.3343)</td>
</tr>
</tbody>
</table>

Table 3: Results from benchmarks: BM25 outperforms PL2, while DR outperforms BM25 by a large margin, indicating that dense methods are more suitable for TOT-KIR.
by BM25. DR, however, appears to handle this better i.e (3), or when there is a low lexical overlap. Surprisingly DR ranks the gold item high for (4), which describes only the cover of the book. The last query (5) contains incorrect information (‘Tuesday’ instead of ‘Monday’) and both methods fail to retrieve the correct item. We present concluding remarks and future research directions in the next section.

6 CONCLUSION

In this paper, we outlined a novel, automated process to gather a dataset for known-item retrieval in two domains, Books and Movies, using heuristics for finding ‘gold’ known-items, along with negatives that other users may confuse the correct item with. Through data quality checks, we showed that this heuristic is accurate, rendering automated collection of TOT-KIR datasets feasible. In contrast to prior work, this process does not require human labeling, allowing for large scale datasets in diverse domains. Furthermore, this process renders synthetic known-item queries unnecessary. Using this algorithm, we collected 15,788 query/known-item pairs, the largest dataset to date for the TOT-KIR task. This large scale dataset allows for training neural models. We have open sourced the code to enable other researchers to collect additional data, perhaps in other domains.

Finally, we evaluated multiple benchmarks on this dataset, using both methods that rely on lexical overlap and methods that create dense representations for retrieval. We showed that lexical methods cannot retrieve the correct known item for a majority of the data, highlighting the difficulty of the task. Furthermore, we showed that a dense retrieval benchmark outperforms the lexical baselines by a large margin. We see many directions of potential future work. The first thread of work involves the data collection process, while the second is methodological.

While we focused on two domains, gathering data for other domains is straightforward, since only part of the algorithm (e.g. the get_data method) needs to be adapted for the new domain. Apart from this, the data can be periodically expanded by considering recent submissions. In addition, while this work focused on only solved submissions, further research is required to understand why TOT requests fail. For instance, discriminative information might be missing in such queries, and identifying this could be useful e.g in the formulation of clarifying questions. The number of data points can be increased by using entity linking instead of using URLs, or by finding the correct known item from the pool of extracted candidates. Finally, data for items can be augmented from additional sources. For instance, we found some users who referred to reviews in their answers. Review information can be extracted and indexed as well, augmenting the information available for items - especially since TOT requests can contain plot information otherwise not contained in a synopsis (‘spoilers’). It has been hypothesized that TOT queries can benefit from multi-hop reasoning to answer indirect queries [2]. Such methods can use entity information, which can be gathered from Wikidata using the Wikidata IDs we extract (for Movies). In addition, since we also collect all replies made by users, they can potentially be used to either build conversational agents or use clarifications in these replies to improve retrieval performance.

ACKNOWLEDGMENTS

The authors would like to thank Wilker Aziz, Antonis Krasakis, Svitlana Vakulenko, Vera Provatorova and Mohammad Alannejadi for their suggestions. The authors also thank the reviewers for their valuable feedback. This research was supported by the NWO Innovational Research Incentives Scheme Vidi (016.Vidi.189.039), the NWO Smart Culture - Big Data / Digital Humanities (314-99-301), and the H2020-EU.3.4 - SOCIETAL CHALLENGES - Smart, Green And Integrated Transport (814961). All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.
REFERENCES


