UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Dynamic Query Modeling for Related Content Finding

Odijk, D.; Meij, E.; Sijaranamual, I.; de Rijke, M.

DOI
10.1145/2766462.2767715

Publication date
2015

Document Version
Author accepted manuscript

Published in
SIGIR 2015

Link to publication

Citation for published version (APA):
Odijk, D., Meij, E., Sijaranamual, I., & de Rijke, M. (2015). Dynamic Query Modeling for Related Content Finding. In *SIGIR 2015: proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval: August 9-13, 2015, Santiago, Chile* (pp. 43-52). Association for Computing Machinery. https://doi.org/10.1145/2766462.2767715

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:07 Oct 2026

Dynamic Query Modeling for Related Content Finding

Daan Odijk[†]
d.odijk@uva.nl

Edgar Meij[‡]
emeij@yahoo-inc.com

Isaac Sijaranamual[†]
i.b.sijaranamual@uva.nl

Maarten de Rijke[†]
derijke@uva.nl

[†] University of Amsterdam, Amsterdam, The Netherlands
[‡] Yahoo Labs, London, United Kingdom

ABSTRACT

While watching television, people increasingly consume additional content related to what they are watching. We consider the task of finding video content related to a live television broadcast for which we leverage the textual stream of subtitles associated with the broadcast. We model this task as a Markov decision process and propose a method that uses reinforcement learning to directly optimize the retrieval effectiveness of queries generated from the stream of subtitles. Our dynamic query modeling approach significantly outperforms state-of-the-art baselines for stationary query modeling and for text-based retrieval in a television setting. In particular we find that carefully weighting terms and decaying these weights based on recency significantly improves effectiveness. Moreover, our method is highly efficient and can be used in a live television setting, i.e., in near real time.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

Keywords

Information retrieval; Query representation; Text streams

1. INTRODUCTION

The way we watch television is changing: a growing number of people do so either on an interactive device or with such a device nearby [31]. Razorfish [36] found that 38% of "mobile multitaskers" access content that is related to the TV program they are watching. The consumption of TV programs can trigger searches for related information or content, e.g., for a broader perspective or to dive deeper into a topic. Thus, we require effective methods to support these emerging information needs. Since people increasingly expect to have related content directly available instead of having to search for it—especially in a TV watching setting—we need to minimize the disruption of having to manually search for related content. An ideal system would therefore present related content in an automatic and timely fashion.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
SIGIR'15, August 09–13, 2015, Santiago, Chile.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3621-5/15/08 . . . $15.00.
DOI: http://dx.doi.org/10.1145/2766462.2767715.

This new and emerging television landscape is beginning to generate new and interesting information retrieval problems. For instance, Blanco et al. [8] perform text-based retrieval in a TV setting and Odijk et al. [32] consider the problem of linking encyclopedic content to a live television stream. Given the highly diverse set of information sources to be retrieved, including web pages, social media, and other video content, we focus in this paper on information needs stemming from live television news broadcasts and provide an approach for retrieving archived video content.

Consider watching a news bulletin as motivation for our retrieval task. As a news item is being covered, users may find themselves interested in watching background video material on one or more of the main entities or themes of the item, either to view at that very moment or to bookmark for later consumption. In order to present users with related video items that provide relevant background information, we require an algorithm that is able to automatically select video items for each news item as it is being broadcast. The output of a system implementing such an algorithm is depicted in Figure 1. Up to four related video items are suggested in an overlay on top of the news broadcast, each consisting of a keyframe and title; in the screenshot the left-most suggestion is highlighted (indicated by the red rectangle). The scenario that we cover in this paper is one in which a user interrupts or pauses the newscast to find videos relevant to the current news item to learn more about the topic. The content retrieved to complement the ongoing broadcast is not personalized but should be related to the broadcast and interesting for a wide audience.

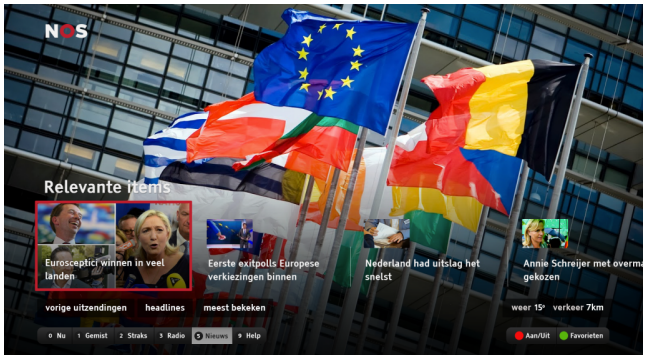


Figure 1: Screenshot of a smart TV application showing related items as an overlay on top of a live TV news broadcast.

In the setting of live television news, a constant stream of subtitles is typically available, generated with only a very slight delay as an aid for the hearing-impaired. To be able to retrieve related content, we analyze these subtitles and generate search queries from the stream that we subsequently use to search a video news archive.

As a news item is being broadcast, additional textual information capturing the content of the news item becomes available. Thus, the challenge that we face is to iteratively incorporate this new information as it appears in the stream to generate and maintain an effective query model, and to do so in near real time.

In this paper we propose our solution: a *dynamic* query modeling approach that explicitly considers the dynamics of streaming sources and is suited for the near real-time setting of live TV. We cast the task of finding video content related to a live television broadcast as a Markov decision process (MDP) and use reinforcement learning to optimize the retrieval effectiveness of queries based on a stream of live television subtitles [4, 38, 39]. We significantly outperform state-of-the-art baselines for stationary query modeling [26] and for text-based retrieval in a television setting [8]. Thus, in this paper, we make the following contributions:

(1) We formalize the task of related content finding to a live television broadcast as a Markov decision process.

(2) We propose a dynamic query modeling approach, using reinforcement learning to optimize a retrieval model that is sufficiently efficient to be run in near real time in a live television setting and that significantly improves retrieval effectiveness over state-of-the-art baselines for stationary query modeling and for text-based retrieval in a television setting.

(3) We provide a thorough evaluation and an analysis of when and why our approach works. We find that adding more weighted query terms and decaying term weights based on their recency significantly improve the effectiveness. Static term features derived from the target collection and background corpora are more important for selecting effective query terms than dynamic features that are dependent on the stream of subtitles.

The remainder of the paper is organized as follows. Related work is discussed in Section 2. Section 3 describes our approach to dynamic query modeling. Our experimental setup is detailed in Section 4. Results are presented and analyzed in Section 5 and discussed further in Section 6. We conclude in Section 7.

2. RELATED WORK

Dynamic query modeling is related to a number of tasks in information retrieval. We discuss related work on search in a streaming setting, on query modeling, on temporal relevance feedback, and on reinforcement learning and MDPs in information retrieval.

Search in a streaming setting. We use dynamic query modeling to find related content based on a textual stream. Similarly, keyword modeling has been studied in a streaming setting, e.g., analyzing transcripts of automatic speech recognition (ASR) [11] and meetings [37]. In this setting, the subsequent task of searching based on the stream is not considered.

VideoCLEF [24] features a linking task aimed at linking video content to related resources across languages. Noisy ASR for Dutch is used to produce links to target English Wikipedia articles. Odijk et al. [32] study the task of finding relevant background information for a second screen in a live TV setting. They approach this as an entity linking problem and propose a graph-based context model. Although both deal with streaming textual content, neither studies the task of search. The MediaEval Search and Hyperlinking task considers two related tasks: finding a known item in broadcast video and providing video hyperlinks for specific anchors. The first is a video retrieval task and for the second task participants focused on the multimedia aspect, e.g., by using image-based retrieval approaches. Our work differs because we model the entire textual stream and base our approach on modeling effective queries.

Henzinger et al. [18] propose an approach to find relevant news articles during broadcast news. Every 15 seconds, they produce a two term query, using a "history feature," consisting of the last three blocks of text. Recent work by Blanco et al. [8] on Yahoo! IntoNews builds and improves on the work of Henzinger et al. [18]. Their focus is on the problem of detecting a change of topic, for which they propose several segmentation approaches. After segmentation, queries are generated using a bag-of-word approach with TF.IDF scores for each term. We include the query modeling approach of Blanco et al. [8] as our first baseline.

Generic methods to automatically segment (e.g., TextTiling [17]) have been extensively studied and shown to perform well in the streaming settings of new event detection [1] and story segmentation, a subtask of topic detection and tracking (TDT). Specific segmentation approaches have been proposed for streaming settings similar to ours (such as [8, 18]). Our dynamic query modeling approach can be applied after using these segmentation methods, instead of the relatively simple TF.IDF approaches used in [8, 18]. This would improve retrieval effectiveness, as we will see below in the comparison with Blanco et al. [8]'s baseline. Furthermore, our approach of decaying term weights based on their recency might provide a combined solution for the segmentation and query modeling problems. In future work, we plan to further investigate the use of decaying term weights in a setting where we need to address segmentation. However, the subtitles that we work with are generated from an auto-cue and thus contain markings indicating the start and finish of an item. Segmenting items or switching from one news item to the next will therefore not be covered in this paper.

Query modeling. A natural way of looking at search in streaming settings is to view it as a query modeling task, where the aim is to model an effective query based on a larger volume of text. This task has two distinct aspects: (i) query reduction and (ii) content-based query suggestion. The former deals with reformulating long or descriptive queries to shorter and more effective queries. Driving this research is that shorter queries are not only more likely to retrieve more focused results, they are also more efficient to process [5]. The latter task is to generate queries and the methods used here are similar to those used for query reduction. Content-based query suggestion has been used for linking news archives across modalities [10] and for generating phrasal-concept queries in literature search with pseudo-relevant feedback [22]. It differs from content-based recommendation, where a user profile or user interactions are also available. Bendersky et al. [7] propose a hybrid approach using content-based heuristics and signals from user interactions to retrieve video suggestions for related YouTube videos. Such a hybrid approach is not feasible in our scenario, as we have no user interactions that relate to the live TV content.

The state of the art in query modeling is formed by a family of related methods, e.g., [23, 26]. The typical approach is to produce query terms, score each, and generate one or more queries. Term scoring is often based on term features taken from large background corpora. Queries are selected based on query difficulty prediction features; these require an initial retrieval round, and as such are not feasible in real time. Lee and Croft [27] generate queries from textual passages by extracting noun phrases or named entities; a conditional random field is used to find selections that optimize retrieval effectiveness. In a web search setting, Kumaran and Carvalho [23] cast query reduction as a sub-query ranking problem. Given a long query, they generate all possible sub-queries for which they subsequently predict query quality and use learning to rank to select

the best. Balasubramanian et al. [2] improve web search results by dropping query terms and estimating reductions in query difficulty.

Methods to improve the retrieval effectiveness of descriptive queries are proposed in [6, 25, 26]. Lease et al. [25, 26] generate queries of up to six terms by sampling term weights and then computing a metric to learn query term weights. Bendersky and Croft [5] extract key concepts from verbose queries to improve retrieval. An extended approach with parameterized query expansion is particularly effective for verbose queries [6]. Similar features are used in [6, 26, 27]; the first two obtain comparable scores on descriptive queries. We propose a dynamic query modeling approach that is tailored for use in a streaming setting and outperforms the stationary baseline based on Lease et al. [26]. Our work differs in that we explicitly model the dynamic nature of streaming sources and in that we generate more complex queries (e.g., using field weights).

The dynamic nature of a document collection is studied in temporal information retrieval (IR), that specifically deals with modeling temporal patterns to improve information retrieval. Efron [14] uses linear time series models for weighting terms, where the time series are computed on the target document collections. Similarly, Peetz et al. [34] use temporal bursts in a microblog collection to reweigh query terms for improved retrieval. Our work differs in that we model temporal dynamics in the stream of subtitles from which we generate queries. We discuss experiments in which we also model temporal dynamics on the collection side in Section 6.

Reinforcement learning and MDPs. We model the task of finding video content related to a live television broadcast as a Markov decision process (MDP) [15] and base our approach on methods from reinforcement learning (RL) [38]. RL intertwines the concepts of optimal control and learning by trial and error. Central is the concept of an "agent" optimizing its actions by interacting with the environment. This is achieved by learning a *policy* that maps *states* to *actions*. An MDP is a specific type of reinforcement learning problem that was proposed before the field was known as reinforcement learning. In an MDP, we decide on the optimal action in a Markov process [4]. The Markov property holds when the policy for a state is independent on the previous states. A Markov state thus has to represent the entire history of previous states as far as this is relevant for the value of the policy.

MDPs have been used to model diverse IR problems, e.g., to explicitly model user behavior in session search [16, 28, 29]. Jin et al. [21] utilize reinforcement learning and MDPs to improve ranking over multiple search result pages. Guan et al. [16] decrease weights of new terms based on past rewards, whereas Luo et al. [28] model session search as a dual-agent stochastic game. Investigating the design choices for MDPs, they find that technology selection and explicit feedback are most effective in session search [29].

In our setting, new subtitles keep coming in, hence we are dealing with a non-stationary MDP. More specifically, because the decision of what action to choose does not influence the states that emerge from the environment, this is considered an associative search task [3]. Perhaps the best studied application of reinforcement learning in IR is such an associative search task, online learning to rank [19, 35, 40]. Here, a retrieval system is optimized based on noisy feedback from user interactions. Our work differs from the work listed above in that we are not just optimizing ranking, but we focus primarily on query generation.

3. DYNAMIC QUERY MODELING

The search task we address is to find relevant content for a dynamic textual stream. We analyze the stream of subtitles that comes with television broadcasts and generate search queries from this stream. We subsequently use these to search a news video archive. Our dynamic query modeling (DQM) approach is designed to take streaming textual data as input and combines a retrieval model that defines a set of hyperparameters w with a learning approach that optimizes these hyperparameters w based on feedback. For learning, we regard the retrieval model as a black box with hyperparameters that affect the retrieval process and we obtain feedback based on the retrieval results. Our learning approach is therefore able to directly optimize for retrieval effectiveness. Figure 2 depicts the retrieval model and Figure 3 summarizes the learning approach. We discuss the learning approach in more detail in Section 3.3. The retrieval model and its hyperparameters will be further detailed below in Section 3.2, after we describe the terminology we will use.

3.1 Terminology

In the context of using subtitles for content linking, Odijk et al. [32] define dynamic textual streams as sources that continually produce "chunks" of text. A chunk is the amount of subtitle text that can be displayed on a single screen. Hence, chunks do not necessarily form a grammatical sentence. However, as these chunks are produced to be read in sequence, syntactic phrases generally do not cross chunk boundaries. Chunks are relatively short, containing about seven terms on average. Chunks form a growing sequence $S = \langle s_1, s_2, \ldots \rangle$, where each s_i is a chunk. The task we address in this paper is to generate, in real time, a query q_i for chunk s_i (having observed chunks $s_1, \ldots, s_{i-1}$) that is able to retrieve relevant video content at that point in the broadcast.

3.2 Retrieval model

The retrieval model we employ consists of four major parts, which are depicted in Figure 2 and described in Algorithm 1. Our retrieval model consists of four steps. First, we incrementally update a list of candidate query terms that we obtain from the textual stream of subtitles. We then compute a score for each term with a weighted sum of term features. Next, we generate a query from the scored query term candidates. The generated queries are complex, consisting of many weighted query terms, and they are selected from the highest scoring query term candidates. Finally, we retrieve the results. The retrieval model defines a set of hyperparameters $w = w_d \cup w_s \cup w_f \cup w_n \cup w_e$ that each alter the retrieval process. The hyperparameters w_d and w_s can be construed as feature weights and w_f as field weights, while hyperparameters w_n and w_e alter the number of selected query terms and the decay of query term candidate scores respectively. We provide more details with respect to these hyperparameters later in this section.

For the first step, *generating query term candidates*, we employ a bag-of-words approach which has proven to be effective in most retrieval settings, including settings similar to ours [26]. We generate a list of query term candidates by tokenizing the subtitles (line 2 in Algorithm 1). All terms from the subtitles in the video segment are considered as query term candidates. For each query term candidate we keep track of when we last saw this term (line 3).

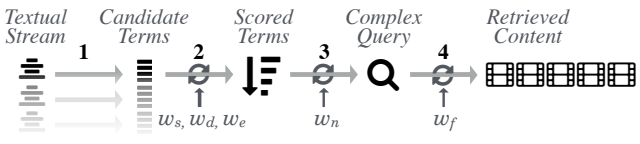


Figure 2: DQM retrieval model for textual streams, consisting of four steps: (1) query term candidate generation; (2) query term candidate scoring; (3) query generation; (4) retrieval.

Table 1: Dynamic term features for stream S.

$TF(t)$	Frequency of the term t in S
$TF.IDF(t)$	$TF(t)$, multiplied by $IDF(t)$ in the target collection
$AugmentedTF(t)$	$0.5 + \frac{0.5 * TF(t)}{max\{TF(t'):t' \in C\}}$
$Capitalized(t)$	Binary feature to indicate whether t has appeared capitalized in S

Table 2: Static term features and their per corpus availability.

		Target	Google	Wikipedia		
		collection	Web1T	title	body	anchors
$P(t,c)$	Probability of term t appearing in c		✓			
$TF(t,c)$	Frequency of term t in c	✓	✓			✓
$DF(t,c)$	Number of docs in c where t occurs	✓		✓	✓	✓
$IDF(t,c)$	$\log \frac{\lvert c \rvert}{DF(t,c)}$, where $\lvert c \rvert$ is number of docs	✓		✓	✓	✓
$RIDF(t,c)$	Residual IDF [13]	✓				✓

Algorithm 1: Dynamic query modeling

Input: Textual stream $S = \langle s_1, \ldots \rangle$.
Output: Stream of complex queries $Q = \langle q_1, \ldots \rangle$, where each query q_i is a set of (term, field, weight) tuples.
Data: Set of hyperparameters:
- w_d weights of dynamic features,
- w_s weights of static features,
- w_e decay rate for a query term candidate weight,
- w_n number of query terms for complex query,
- w_f weight for each document $field$.

1 **foreach** *new chunk in S* **do**

Step 1: Generate query term candidates

2 Tokenize the new chunk.
3 Update term candidates with new terms and last seen index.

Step 2: Assign score to each query term candidate

4 Compute static features for new candidates.
5 **foreach** *query term candidate* **do**
6 Compute dynamic features for each candidate term.
7 Compute query term candidate score as the weighted sum of feature values, using weights w_d and w_s.
8 Decay query term candidate score with $e^{-w_e \cdot i}$, where i is the relative age of term t in the stream thus far (0 for the current chunk and 1 for the first chunk).

Step 3: Generate complex query

9 Select top w_n query term candidates with highest score.
10 Generate complex query with individual field weights w_f.

The next step in the DQM retrieval model is to *assign a score* to each of the candidate query terms. A score for a query term candidate is initially computed as a weighted sum of the term's features. The feature weights are hyperparameters w_s and w_d for the retrieval model. We use a set of term features that are either static for the term or updated dynamically with the stream of text. The static features are computed once for each new query term candidate in the stream (line 4). The dynamic features are updated in each new state for all query term candidates (line 6). The dynamic and static features are listed in Tables 1 and 2 respectively.

The dynamic features are computed with information from the textual stream S. They include the term frequency $TF(t)$ and augmented term frequency, intended to prevent a bias towards longer documents. $TF.IDF(t)$ is computed using the target collection. The $Capitalized(t)$ feature indicates whether a term appeared capitalized in the stream, ignoring the beginning of a sentence. The intuition behind this is that a person or location is likely to appear capitalized and might be an effective query term.

We also compute five static features over three corpora (see Table 2). The first corpus we compute static features for is the target collection, where the features indicate whether the candidate query term will be effective in searching that collection. The two other corpora provide an indication of how common a term is across the web and in different document fields of an encyclopedic source. Both our subtitle source and the descriptions for the video items in our target collection are in Dutch. We therefore use the Dutch unigrams of the Google Web1T [9] corpus as counts of how common a term is on web pages. The counts were generated from approximately one hundred billion tokens. We use Wikipedia, with articles separated into title, body and anchor, as a source to indicate how common a term is in an encyclopedic text. Each should provide different clues about a term. A term appearing in the title might be more important than one appearing in the body. Likewise, if a term is used as anchor text, it might describe well what it refers to. Our five static features include the frequency of a term t, the number of documents it appears in and the probability of observing term t in the corpus c. From the document frequency we derive the inverse document frequency (IDF) and the residual IDF. This is the difference between the observed IDF and the value predicted by a Poisson model [13]: $RIDF(t,c) = IDF(t,c) - \log_2 \frac{1}{1-e^{-TF(t,c)/|c|}}$, where $|c|$ is the number of documents in c. Not all features can be computed for each corpus, e.g., the Google Web1T collection lacks document counts. The sets of dynamic and static features are extended with logarithmically transformed values. All 38 resulting features are min-max normalized.

For each query term candidate we compute a score as the weighted sum of all 38 features multiplied by an exponential term decay factor (lines 7 and 8 in Algorithm 1). This term decay factor is computed separately for each term candidate, based on how recently the term was observed in the stream S. The intuition behind this is that a more recently mentioned term might be more relevant than one that has not recently been mentioned. Hyperparameter w_e governs the decay rate. Concretely, we multiply the weighted sum of features with $e^{-w_e \cdot i}$, where i is the relative age of term t in the stream thus far. This relative age ranges from 0 to 1 and is 0 for the current chunk and 1 for the first chunk in the stream. The number of steps between 0 and 1 is equal to and increases with the number of chunks between the current and the first chunk in the stream.

The third step in the DQM retrieval model is to *generate a complex query* (lines 9 and 10 in Algorithm 1). From the ranked query term candidates we generate complex queries for the top n terms, where n is based on a hyperparameter w_n. The weights for each term in the resulting query are set to the score produced in the query term candidate scoring stage. We explicitly allow for negative weights for term features which may cause negative term candidate scores. In this case we omit the term, resulting in a query with fewer query terms. We extend the query with learned field weights, allowing the model to learn to attribute more importance to specific fields, such as the title of a video. The field weights w_f are also exposed as hyperparameters.

The final step in the DQM retrieval model is to *retrieve* the results. For this we use a state-of-the-art search engine that uses language modeling for retrieval and is described further in Section 4.

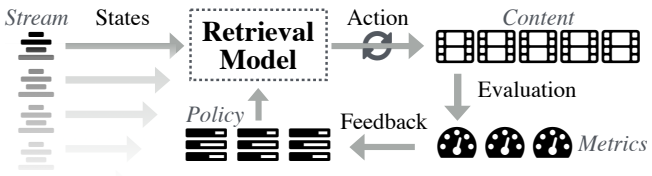


Figure 3: DQM learning approach. A textual stream is processed using a retrieval model that defines a set of hyperparameters. Feedback on the retrieval results is used to optimize the hyperparameters.

Algorithm 2: Our implementation of the Dueling Bandits Gradient Descent algorithm [40].

Input: step size δ, learning rate α, initial weights w_1, retrieval function *retrieve*, a *metric*, *maxIterations*

$w = w_1$
for $i \in \{1, .., maxIterations\}$ **do**
  Sample small step ε of maximally length δ.
  $w' = \frac{w+\varepsilon}{|w+\varepsilon|}$
  $a = retrieve(f * w)$
  $b = retrieve(f * w')$
  $d = metric(b) - metric(a)$
  if $d > 0$ **then** $w = \frac{w+\alpha*\varepsilon}{|w+\alpha*\varepsilon|}$

3.3 Learning

The learning approach (depicted in Figure 3) considers the retrieval model as a black box that defines a set of hyperparameters, altering the retrieval process in a complex manner (cf. Section 3.2). We regard this learning problem as an MDP. A new *state* occurs when a new chunk of subtitles presents itself. In our setting we optimize the *action* of generating a query based on *feedback* in terms of retrieval effectiveness. We obtain this feedback by generating search results through our retrieval model that is governed by a set of hyperparameters. In reinforcement learning (RL) terms, we regard these hyperparameters as the *policy* that we are optimizing.

RL differs from supervised machine learning in that it explicitly deals with optimizing actions for the whole problem space based on goals within an uncertain environment. RL's trial and error type of learning fits well with the nature of our problem. We do not just optimize a single problem dimension, i.e., a combination of ranking features, but we optimize the entire process, from selecting and scoring candidate terms to generating a complex query.

We learn an optimal policy using the Dueling Bandits Gradient Descent (DBGD) algorithm [40], detailed in Algorithm 2. This algorithm iteratively tries a new policy w' that is a small change from the current best policy w. Based on feedback on the effects of both policies w' and w, it finds a winning policy. From this comparison it decides to keep the current policy w or to move the current best policy in the direction of w'. By iteratively updating the current best policy w, DBGD performs hill climbing to obtain an optimal policy based on the feedback. It thus obtains hyperparameter settings that optimize retrieval effectiveness for our DQM retrieval model.

DBGD has two parameters: step size δ for generating a new w' and learning rate α. Here, δ controls how big a change in weights we compare the current best to. The learning rate α controls how large a step is taken towards a better weight vector once it is found. We initially set the weights w_1 randomly for the feature weights w_d and w_s and the number of terms w_n to 10. We initialize the retrieval model, with equal field weights w_f and a decay term w_d of 0.

To compare the effects of two policies w and w', we retrieve results produced by the DQM retrieval model for both policies. We are not in a position to experiment with actual users from whom we could obtain online feedback, e.g., using interleaved comparison methods [35]. We therefore use offline relevance assessments. For both retrieval results (obtained through w and w'), we compute a retrieval performance metric. This can be any metric; see Sections 4 and 5.3. Based on the outcomes, we decide whether the new policy w' is better than the current best policy w. In case of a draw, the current best is kept. If we were to use feedback from live user interactions, the feedback would be more noisy; we explore how noisy feedback affects learning in the analysis in Section 5.3.

Since the items of a news broadcast are segmented, our task can be performed naturally on subsequences, which we call "video segments" (i.e., *episodes* in RL terminology). The last chunk of a video segment produces a special state, the terminal state. It is followed by a reset to the start state. DQM is *episodic* with an *indefinite horizon*, i.e., episodes are finite, but without a predefined length. The policy we optimize is generic across episodes, i.e., it does not consider any episode-specific information and will not be reset after a terminal state. Our view of this learning problem as an episodic MDP fits well with the segmented nature of TV news.

A typical issue for any RL problem is deciding when to learn: should we explore new policy options or exploit the current policy. We assume that users are interested in related content near the end of an episode, as this is where they will have learned most about a topic. We therefore chose to only do an explorative learning action in the terminal state of an episode. In the other states, we exploit the current best policy. This way, we obtain less feedback but optimize the policy at the states where it is most important to perform optimal.

4. EXPERIMENTAL SETUP

We seek to answer the following research questions regarding our proposed model:

RQ1 Does dynamic query modeling improve retrieval effectiveness over state-of-the-art stationary query modeling baselines?

RQ2 What do the components of the DQM retrieval model contribute to effectiveness?

We ask the following research question regarding our reinforcement learning approach:

RQ3 How do the reinforcement learning parameters, choice of optimization metric and noisy feedback influence the speed of the learning process and the resulting retrieval effectiveness?

To answer these research questions, we create an annotated dataset and set up experiments. Below, we describe the dataset, experimental set-up and detail our evaluation.

Subtitles. We obtain a dataset of subtitles for the hearing impaired from the Dutch eight o'clock evening news broadcast of the Nederlandse Omroep Stichting (NOS), the Dutch public news broadcaster. We selected this as our source because its content is diverse and volatile; it may cover items broadly and for minutes, or just very briefly. A news broadcast typically lasts about 25 minutes and contains around ten items, which we refer to as *video segments* or *episodes* in RL terms. A typical video segment consists of 44 chunks of on average seven terms per chunk and 306 terms per video segment. For evaluation purposes, seven news broadcasts are randomly selected from broadcasts dated May 2013, containing 50 video segments in total. The video segments do not overlap

in main topic. The subtitles are prepared based on the text for the teleprompter or auto-cue. It therefore contains markings to indicate when a new video segment starts.

Collection. As our target collection we use the video archive of the same news broadcaster, NOS, which contains individual news item videos. The video items are often taken from news broadcasts, but can also be longer versions of interviews or aimed to provide further background. This collection is publicly available and can be crawled via their website.[1] Our index covers the years 2011–2013 and contains 37,884 video items, with an average of 40 video items per day. For our experiments, we limit the queries to only the video items that were published before the news broadcast. We consider the title, description and tags as different textual fields. Note that we do not use any video specific information and thus regard the video items simply as textual metadata records.

Ground truth. To establish ground truth for our evaluation,[2] we ask assessors to read the subtitles of a video segment and then rate videos for relevance. These items are obtained by pooling the top rankings for each baseline to on average 79 videos per segment. The video items in the pool are presented in random order. We train two assessors and each one annotated half of the video segments. Our instructions to the assessors were: *Imagine that you interrupted the news broadcast after the segment because you're interested in watching related video content. How would you rate each video?* We use a five-point scale to capture the level of relevance of each video [12] (with the usual labels *perfect*, *excellent*, *good*, *fair* and *bad*). The distribution of the ratings over the respective labels from perfect to bad is 5%, 5%, 8%, 16% and 66%. This suggests that the task is not an easy task, but there are plenty of good videos to rank. For each video item, the assessors are provided with all metadata (title, date, description, keywords) and can watch and explore the original video to make a better judgement.[3]

Evaluation. In our setting, a viewer is searching for related video content for a news broadcast item, most likely at the end of an item or just after it has finished. We therefore evaluate the retrieval effectiveness at the end of a video segment or in RL terms, the terminal state of an episode. In the DQM setting, it is important to provide the most relevant video and to provide them as high in the ranking as possible. In a live TV setting, a user would not examine a full result list, but only a limited number of video items. As our main evaluation metric we choose normalized discounted cumulative gain (nDCG) [20], as it can handle graded relevance assessments and takes positional importance into account. We compute nDCG for the entire result list and for the first five positions as nDCG@5, skipping video items that were not annotated. We perform leave-one-out cross validation to evaluate our approach. For each video segment we train an individual model where all other video segments serve as training material. We then evaluate the effectiveness on the video segment that was left out of the training set. We consider nDCG@5 more relevant for our setting but optimize for nDCG as it is smoother than nDCG@5 and already gives high importance to the retrieved documents at the top of the ranking. We assume that taking the improvements in ranking below the fifth position into account will benefit learning in the long run. We revisit this decision in the analysis of our approach (Section 5).

Retrieval engine. For all experiments we use the language modeling and inference network-based Indri retrieval model [30], with stopword removal and without stemming. We use Dirichlet smoothing with smoothing parameter μ set to the default 2500.

[1] http://nos.nl [2] http://ilps.science.uva.nl/resources/sigir2015-dqm

[3] We evaluate assessor-agreement over a set of 25 videos from five doubly annotated segments; Spearman's rank correlation measures 0.8636, signaling good agreement.

Baselines. Since we evaluate at the end of a video segment, we can compare to stationary *baselines*. For these baselines, we concatenate all subtitles of a video segment to form a single pseudo document. Based on this pseudo document we search for related video content. In this way, the task becomes similar to the more-like-this task, that is supported by many search engines. We include two stationary approaches, that we label "Baseline" and "Modified Lease." The first, "Baseline", uses the top-10 terms from a bag-of-words model of the pseudo document, ranked by TF.IDF score (where the document frequency is computed on the target collection). This baseline is how Blanco et al. [8] perform query modeling in their text-based retrieval approach for TV. The second stationary approach ("Modified Lease") is comparable to the state-of-the-art model of Lease et al. [26]. The retrieval model is based on regression to learn queries that consist of no more than six terms. The terms are selected and weighted based on supervised machine learning (regularized linear regression) using a bag-of-words representation of the pseudo document. The features we use for the Modified Lease baseline are the same as for our DQM approach, excluding the features based on Wikipedia (not used in [26]). Furthermore, we choose not to include the simple part-of-speech features and the lexical context features (the word before and word after) from their model. These features get relatively low weights in [26] and are less applicable in our setting than in their descriptive queries setting (hence, we dub this approach "Modified Lease").

5. RESULTS & ANALYSIS

We describe the results of our experiments and investigate the effectiveness of our DQM approach, following the three research questions listed in the previous section.

5.1 Retrieval effectiveness

To answer RQ1, we compare the effectiveness of DQM to that of the stationary baselines. Table 3 shows the performance in terms of retrieval effectiveness. Both baselines show a decent performance with an nDCG score of around 0.7. We cannot directly compare the performance of the baselines to that of Lease et al. [26], as they report on different collections and use shorter descriptive queries. Surprisingly, the modified Lease approach is not able to significantly improve on the less complex baseline. A plausible explanation for this is the limited number of query terms in the modified Lease approach; see the analysis in Section 5.2 below.

Rows 3–8 of Table 3 show the retrieval effectiveness of our DQM approach, building up from a basic approach (labeled DQM$^-$) to

Table 3: Retrieval effectiveness of the dynamic query modeling (DQM) approach vs the two baselines. Significant differences, tested using a two-tailed paired t-test, are indicated with $^-$ (none), $^\triangle$ ($p < 0.05$) and $^\blacktriangle$ ($p < 0.01$); the position of the symbol indicates whether the comparison is against row 1 (left most), row 2 (center) or row 3 (right most).

	Method	nDCG@5	nDCG
1.	Baseline [8]	$0.6486^{-\blacktriangledown}$	$0.6113^{-\blacktriangledown}$
2.	Modified Lease [26]	0.6994^{--}	$0.6484^{-\blacktriangledown}$
3.	DQM$^-$	$0.7570^{\blacktriangle--}$	$0.7393^{\blacktriangle\blacktriangle}$
4.	DQM$^-$ + field weights	$0.7651^{\blacktriangle\triangle-}$	$0.7566^{\blacktriangle\blacktriangle\triangle}$
5.	DQM$^-$ + term weighting	$0.7814^{\blacktriangle\blacktriangle-}$	$0.7845^{\blacktriangle\blacktriangle\blacktriangle}$
6.	DQM$^-$ + term and field weighting	$0.7781^{\blacktriangle\blacktriangle-}$	$0.7945^{\blacktriangle\blacktriangle\blacktriangle}$
7.	DQM$^-$ + decayed term weighting	$0.7940^{\blacktriangle\blacktriangle\blacktriangle}$	$0.7897^{\blacktriangle\blacktriangle\blacktriangle}$
8.	DQM	$0.8005^{\blacktriangle\blacktriangle\blacktriangle}$	$0.8072^{\blacktriangle\blacktriangle\blacktriangle}$

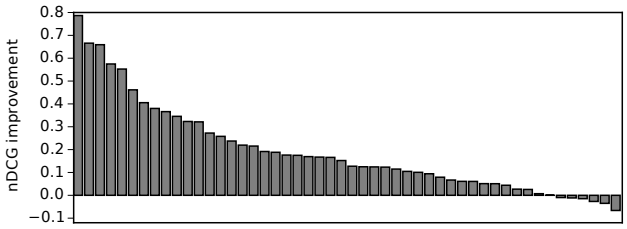


Figure 4: Improvement in terms of nDCG for DQM over the baseline, where each bar represents a video segment.

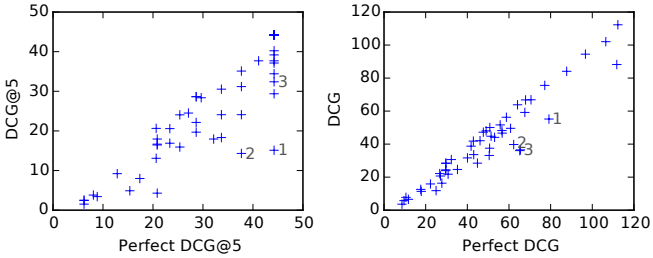


Figure 5: DCG for DQM versus perfect DCG for each video segment for the top five (left) and for the full result list (right).

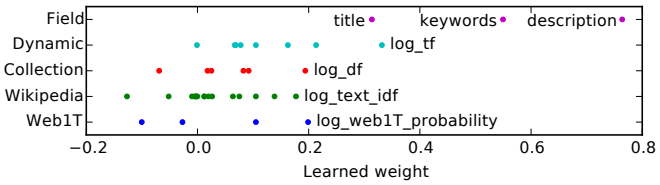


Figure 6: Feature and field weights for the DQM approach. The field weights and top weights for each group are annotated.

the full DQM approach. DQM$^-$ uses only the dynamic and static term features to select query terms. It is, however, already able to significantly improve nDCG over both baselines (row 3). Next, we look at the weighting in DQM. Enabling field weighting gives a small boost in retrieval effectiveness (row 4). If we enable term weighting based on the machine learned scores, the model performs significantly better on the full result list, although the effect is not significantly present on nDCG@5 (row 5). Interestingly, adding field weights to the term weighting approach results in a drop of retrieval effectiveness for the top five, but improved effectiveness for the full rank list (row 6). Enabling term weight decay without field weights gives an additional boost towards a significantly better approach than the base approach, both on the top five and the full result list (row 7). Finally, our full DQM approach in row 8 significantly improves over the stationary baselines and the base DQM$^-$ approach. The effects are similar in nDCG and nDCG@5, although they are more clear for nDCG, i.e., the full ranked list.

To investigate where our approach works, we look at the effectiveness per video segment compared to the baseline. We plot the difference in nDCG in Figure 4. Our approach is able to substantially improve retrieval effectiveness for the bulk of the video segments. There are seven segments where our approach hurts performance. A closer look at these reveals that they mostly already have a high nDCG score for the baseline. There does not seem to be an influence of the length of the segments.

We further study the effectiveness across video segments by separating the nDCG scores into their components: the DCG score and the perfect DCG score. *Perfect DCG* is the DCG score obtained when the documents are ideally ranked according to the ground truth. If a ranking is ideal, the DCG score is equal to the perfect DCG score and nDCG is equal to 1. We plot the DCG scores versus the perfect DCG scores in Figure 5, for the top five and the full result list. A perfect nDCG score would be on the diagonal. The closer a result is to the bottom, the lower the nDCG score. We see a similar pattern for the full ranked list as for the top five, although more distinctly for the top five. DQM is able to obtain perfect scores for video segments with both high and low perfect DCG scores. However, we also perform below perfect; this does not seem to be related to the perfect score that can be obtained.

We look in more detail at three video segments that are far from the diagonal (and thus have a low nDCG score), marked in Figure 5 as 1, 2 and 3. Interestingly, segments 1 and 3 are in the top five most improved by our approach and segment 2 is one of the few that is hurt by DQM (respectively the 4th and the 5th bar from the left and the 4th from the right in Figure 4). Segment 1 is broad, linking French protests against a new law on gay marriage to a movie at the Cannes film festival. Segment 3 is a short item about the increase in ATM robberies and segment 2 is a broad item about the relationship between the Netherlands and Germany, on the occasion of a trade summit. From this we can see that, although DQM gets substantial improvements for broad items, there is still something to be gained.

5.2 Analysis of components

To answer RQ2, we investigate the contribution of the components of our approach. The learned values of the DQM retrieval model hyperparameters are plotted in Figure 6. Looking at field weights w_f, we observe that different values are learned, the highest weight is twice the value of the lowest weight. The highest weight is given to the longer description field. The keywords get a higher weight than the title field.

Number of weighted query terms. Next, we turn to the number of weighted query terms that DQM generates. In our experiments, this parameter was set through RL to a value close to 100 terms. This can include terms with very small weights or even negative weights (and thus not included in the final query). This is a substantially larger number than the fixed number of 6 query terms in the Lease approach. We investigate the impact on the retrieval effectiveness of DQM and the baselines; see Figure 7. From this figure we observe that both baselines clearly benefit from adding more terms to the query. The modified Lease approach consistently outperforms the baseline approach. Interestingly, for the baseline approach, adding more query terms than around 30 does not further improve the effectiveness on the top five results, but does steadily improve the effectiveness on the entire ranked list. The modified Lease approach actually shows a drop in effectiveness on the top five when using more than around 50 query terms. For any number of query terms, DQM consistently outperforms both the baseline and Modified Lease on both metrics. For DQM, the retrieval effectiveness in terms of nDCG for the top five results and the entire result list show a qualitatively similar pattern. The effectiveness of DQM increases with the number of query terms, until it reaches a plateau at around 40 query terms.

Term decay. In our experiments the term decay factor w_e was set through RL to a value of 0.5601. Our retrieval model thus has a mild preference for more recent terms. With this decay factor query term candidates from the first chunk are given a score that is 57% lower than the most recent terms. In Table 3, we observed a small boost in retrieval effectiveness when enabling decayed term weights for our full DQM approach. Interestingly, adding term decay to our baseline gives an nDCG improvement of 11.90% for the top 5 and 16.29% for the full ranked list. This suggests that term decay does indeed contribute to improving retrieval effectiveness.

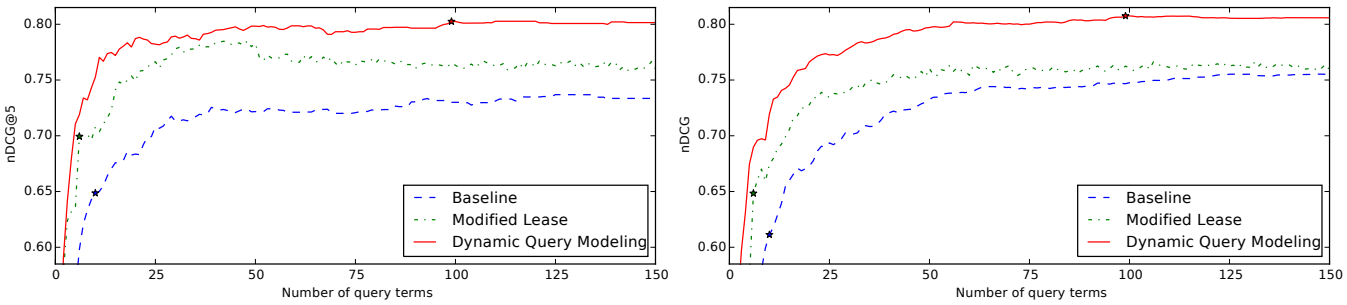


Figure 7: Baseline retrieval effectiveness with different number of query terms as measured on the top five (nDCG@5, left) and on the entire result list (nDCG, right). The settings used in our main experiments are marked with a star.

Term feature weights. Next, we turn to the term feature weights w_d and w_s, plotted in Figure 6. We observe that a diverse set of features receive high weights; there is no single most important feature. The highest weights are assigned to the dynamic features. Of the static features, the collection features receive the highest weights, specifically the logarithm of the document frequency. For Wikipedia-based features, the log IDF in text receives the highest weight. The weights for features derived from the title and anchor text are substantially smaller than for the all Wikipedia text. For Web1T, the log term probability receives the highest weight. Five features receive negative weights, most notably the document frequency in Wikipedia text and the frequency in Web1T. These weights indicate a natural negative bias for common terms.

Contributions of components. We perform an ablation study and disable parts of DQM to investigate the effects on the retrieval effectiveness; see Table 4. We evaluate the effectiveness of our DQM approach with specific sets of term features disabled. These results are presented in rows 2–6 of Table 4. On rows 2 and 3, we see the effects of disabling respectively the entire set of dynamic and of static term features. Both result in a significant drop of performance. In fact, if we disable all static term features, the performance on the top five drops below the modified Lease baseline. Clearly, the static term features are essential for the performance of DQM. This makes it surprising that the dynamic term features obtain the highest weights. To further investigate this, we disable specific subsets of static term features. We observe from rows 4–6 in Table 4 that disabling any of the subsets significantly degrades the performance of DQM. For Web1T, this effect is not significant in the top five and not as significant as for the Wikipedia and collection features. Lastly, we investigate whether our strong static term features are strong enough by themselves, without the other features. The results in rows 7–9 of Table 4 show a qualitatively similar pattern to rows 4–6. Using only one subset of the static term features and no dynamic features significantly degrades the performance, although all variants still outperform our baselines. Using only Wikipedia for term features comes closest to our full DQM model, but it still performs significantly worse.

5.3 Learning

To answer the reinforcement learning related research question RQ3, we investigate how DQM learns by looking at learning curves. These curves are generated by evaluating the DQM model at each iteration on the left-out video segments. The curves are averaged over five runs for each of the 50 video segments and are thus comparable to the numbers in Tables 3 and 4.

Reinforcement learning parameters. First, we turn to the two parameters of the DBGD algorithm: the step size δ and the learning rate α. δ determines the distance between the current best weights and the new weights. The learning rate α is the size of the step taken towards the best scoring weight vector. We explore different values for δ and α and plot the results in Figure 8a.

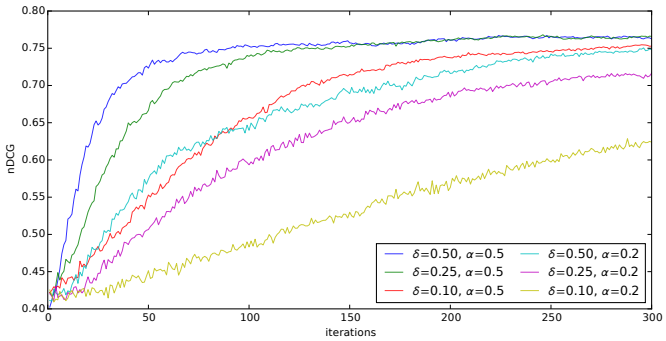
Table 4: Ablation analysis of the contributions of DQM term feature sets to retrieval effectiveness. Significant differences, tested using a two-tailed paired t-test against row 1, are indicated for rows 2–9 with $^-$ (none), $^\triangledown$ ($p < 0.05$) and $^\blacktriangledown$ ($p < 0.01$).

Method	nDCG@5	nDCG
1. Full Dynamic Query Modeling (DQM)	0.8005	0.8072
2. without static features	0.6884$^\blacktriangledown$	0.6721$^\blacktriangledown$
3. without dynamic features	0.7823$^-$	0.7698$^\blacktriangledown$
4. without Web1T features	0.8012$^-$	0.7967$^\triangledown$
5. without Wikipedia features	0.7459$^\blacktriangledown$	0.7485$^\blacktriangledown$
6. without Collection features	0.7421$^\blacktriangledown$	0.7267$^\blacktriangledown$
7. with only Web1T features	0.7387$^\blacktriangledown$	0.7377$^\blacktriangledown$
8. with only Wikipedia features	0.7633$^\triangledown$	0.7617$^\blacktriangledown$
9. with only Collection features	0.7571$^\triangledown$	0.7583$^\blacktriangledown$

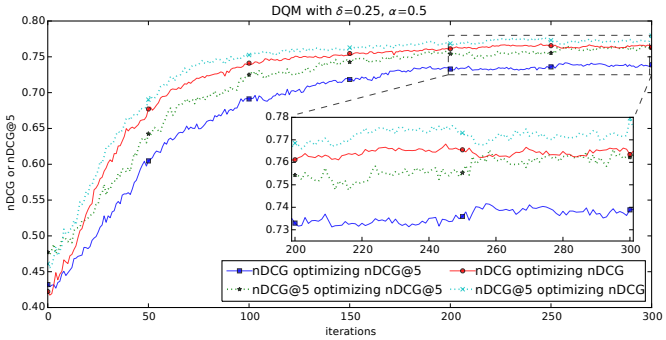
We can observe from Figure 8a that for the same step size δ, a larger learning rate means that we reach higher effectiveness in fewer iterations. A larger step size δ will also result in faster learning, although there appears to be a risk in setting the δ to high. Despite early gains when using $\delta = 0.5$ and $\alpha = 0.2$, at around 75 iterations, it gets taken over by the run with the much smaller step size of $\delta = 0.1$ and higher learning rate of $\alpha = 0.5$. At $\alpha = 0.5$ the medium step size $\delta = 0.25$ appears to be able to better learn the subtleties when the effectiveness reaches a plateau, in comparison with the higher learning rate $\alpha = 0.5$.

Next, we investigate how optimizing for nDCG influences the performance as measured by nDCG@5. In Figure 8b we plot nDCG and nDCG@5 for runs in which we optimize either the nDCG or the nDCG@5 metric. We observe a consistently higher nDCG value when optimizing nDCG versus optimizing nDCG@5. Interestingly, we also see a consistently higher nDCG@5 value, confirming our idea that optimizing nDCG will also optimize nDCG@5.

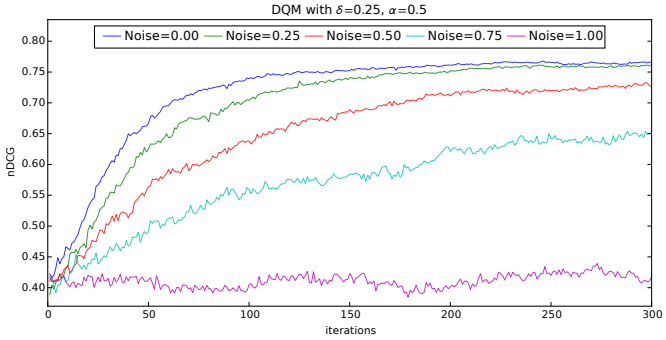
Noisy feedback. DQM is also suited for use in a setting where feedback comes from user interaction and thus is not always perfect. We study how well it can deal with noisy feedback. For this we run a variant of DQM where we replace the comparison in the DBGD algorithm with a noisy comparison. This noisy comparison randomly returns a random comparison outcome. The noise level controls how often this occurs, where a noise level of zero is equal to our regular setting and a noise level of one results in completely random feedback. The results of this analysis are presented in Figure 8c. We observe that given completely random feedback, DQM will not improve in terms of effectiveness. However, it will also not degrade in performance. Adding more noise to the feedback for DQM will increase the number of iterations it takes to learn an



(a) Learning curves for different values of DBGD parameters δ and α.



(b) Development nDCG(@5) when optimizing for nDCG or nDCG@5.



(c) Influence of noise on the convergence rate of effectiveness.

Figure 8: Learning curves showing the development in terms of effectiveness for the DQM approach across learning iterations. Metrics are computed using leave-one-out cross validation and averaged over five runs and 50 video segments at each iteration.

optimal value, but an optimal value will be found even with highly noisy feedback. This suggests that DQM is also suited for use in a setting where feedback comes from user interaction.

6. DISCUSSION

We see two obvious improvements to retrieval effectiveness for our DQM approach, which we will discuss here. First, we could use pseudo-relevance feedback (PRF) to do query expansion, an approach shown to be effective for generating phrasal-concept queries in literature search [22]. To increase recall, the original query is expanded with pseudo-relevant terms taken from the top retrieved documents. We experimented with enabling pseudo-relevant feedback. For the baseline, this increases nDCG with 3.5% for the top five and 7.3% for the full ranked list. We observed no improvement in retrieval effectiveness for DQM when enabling PRF. A possible explanation is that DQM's retrieval model uses many query terms, selected partly based on term statistics for the target collection. Instead of Indri's out-of-the-box PRF approach, a PRF approach that is tailored to our DQM retrieval model and the complex queries that it produces might be able to improve retrieval effectiveness.

Second, a plausible assumption is that more recent videos might be more relevant. We therefore also explored using temporal document priors, specifically ones inspired by human memory [33]. The DQM learning approach found no preference for recent documents and we observed no improvements in retrieval effectiveness with a temporal document prior compared to without one. An analysis of our ground truth confirmed that more recent documents were not found to be more relevant. This might be due to our retrospective annotation of the ground truth. In a live TV setting, with feedback from actual user interactions, more experimentation with a temporal document prior (e.g., a Weibull decay) would be advised.

So far, the discussion of our results has focused on effectiveness. However, our target setting is live TV, where efficiency is of great importance. To verify that we can efficiently perform DQM in near real time, we compute the average number of chunks we process per second on a single core machine, averaged over ten passes over all 50 video segments. DQM is able to process 23.9 chunks per second. Odijk et al. [32] report that they observe an average of 0.24 chunks per second in a live TV setting similar to ours, two orders of magnitude less than what we are able to achieve.

7. CONCLUSION

We formalized the task of finding video content related to a live television broadcast as a Markov decision process and proposed a reinforcement learning approach to directly optimize retrieval effectiveness. We showed that our approach significantly improves retrieval effectiveness over state-of-the-art stationary baselines, while remaining sufficiently efficient to be used in near real time in a live television setting. We have shown that each DQM retrieval model component contributes to the overall effectiveness. A larger number of weighted query terms significantly improve effectiveness. Static term features that are dependent on the target collection and background corpora are more important for selecting effective query terms than dynamic features derived from the stream of subtitles. Decaying term weights based on their recency further improves retrieval effectiveness.

Regarding our reinforcement learning approach we have found that a medium explorative step size and a larger learning rate are the best choice in our setting. We have shown that optimizing nDCG also yields the best nDCG@5 scores. Lastly, we showed that our reinforcement learning based approach to DQM still learns effectively when feedback becomes noisy. This suggests that our DQM approach is also suited for use in a setting where feedback comes from user interaction.

To understand the broader applicability of our work, it helps to point out that DQM is a task that combines two traditional basic information retrieval tasks: search and filtering. In a typical search task the query changes and the collection remains static. In a typical document filtering task, a standing query is used to filter a stream of documents. Our task concerns both. Similar tasks exist, such as summarizing social media in real time and finding replications of news articles while they appear. We believe that our DQM approach is applicable to those tasks too.

As to future work, we plan to explore learning from noisy feedback from actual user interactions. DQM generates a single query in each state. It may be useful to generate multiple queries and merge either queries or results. Similarly, an extension that explicitly generates temporally or topically diverse results may enhance the user experience. We could model this as a slot filling problem, where we have four video slots for which we select the most interesting video to show a user.

Acknowledgements. We thank NOS, NPO, and TWC for cooperating on the smart TV application that formed the setting for this research. This research was supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the Netherlands Organisation for Scientific Research (NWO) under project nrs 727.011.005, 612.001.116, HOR-11-10, 640.006.013, 612.066.930, CI-14-25, SH-322-15, Amsterdam Data Science, the Dutch national program COMMIT, the ESF Research Network Program ELIAS, the Elite Network Shifts project funded by the Royal Dutch Academy of Sciences (KNAW), the Netherlands eScience Center under project number 027.012.105, the Yahoo! Faculty Research and Engagement Program, the Microsoft Research PhD program, and the HPC Fund.

REFERENCES

[1] J. Allan, J. Carbonell, G. Doddington, J. Yamron, and Y. Yang. Topic detection and tracking pilot study final report. Computer Science Department Paper 341, CMU, 1998.

[2] N. Balasubramanian, G. Kumaran, and V. R. Carvalho. Exploring reductions for long web queries. In *SIGIR '10*, pages 571–578. ACM, 2010.

[3] A. G. Barto, R. S. Sutton, and P. S. Brouwer. Associative search network: A reinforcement learning associative memory. *Biological cybernetics*, 40(3):201–211, 1981.

[4] R. E. Bellman. A markovian decision process. *J. Math. Mech.*, 6:679–684, 1957.

[5] M. Bendersky and W. B. Croft. Discovering key concepts in verbose queries. In *SIGIR '08*, pages 491–498. ACM, 2008.

[6] M. Bendersky, D. Metzler, and W. B. Croft. Parameterized concept weighting in verbose queries. In *SIGIR '11*, pages 605–614. ACM, 2011.

[7] M. Bendersky, L. Garcia-Pueyo, J. Harmsen, V. Josifovski, and D. Lepikhin. Up next. In *KDD '14*, pages 1769–1778, New York, New York, USA, Aug. 2014. ACM Press.

[8] R. Blanco, G. D. F. Morales, and F. Silvestri. Intonews: Online news retrieval using closed captions. *Inf. Proc. & Man.*, 51(1):148–162, 2015.

[9] T. Brants and A. Franz. Web 1T 5-gram 10 european languages version 1 LDC2009T25. Philadelphia: Linguistic Data Consortium, 2009.

[10] M. Bron, B. Huurnink, and M. de Rijke. Linking archives using document enrichment and term selection. In *TPDL '11*, pages 360–371. Springer, 2011.

[11] E. Brown, S. Srinivasan, A. Coden, D. Ponceleon, J. Cooper, A. Amir, and J. Pieper. Towards speech as a knowledge resource. In *CIKM '01*, pages 526–528. ACM, 2001.

[12] B. Carterette and R. Jones. Evaluating search engines by modeling the relationship between relevance and clicks. In *NIPS*, volume 7, pages 217–224, 2007.

[13] K. W. Church and W. A. Gale. Poisson mixtures. *Nat. Lang. Eng.*, 1:163–190, 1995.

[14] M. Efron. Linear time series models for term weighting in information retrieval. *J. American Society for Information Science and Technology*, 61(7):1299–1312, 2010.

[15] E. Feinberg and A. Shwartz. *Handbook of Markov Decision Processes*. Kluwer, 2002.

[16] D. Guan, S. Zhang, and H. Yang. Utilizing query change for session search. In *SIGIR '13*, pages 453–462. ACM, 2013.

[17] M. A. Hearst. TextTiling: Segmenting text into multi-paragraph subtopic passages. *Comput. Linguist.*, 23 (1):33–64, March 1997.

[18] M. Henzinger, B.-W. Chang, B. Milch, and S. Brin. Query-free news search. *WWW '05*, 8(2):101–126, 2005.

[19] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval*, 16(1):63–90, 2013.

[20] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4): 422–446, Oct. 2002.

[21] X. Jin, M. Sloan, and J. Wang. Interactive exploratory search for multi page search results. In *WWW '13*, pages 655–666. ACM, 2013.

[22] Y. Kim, J. Seo, W. B. Croft, and D. A. Smith. Automatic suggestion of phrasal-concept queries for literature search. *Inf. Proc. & Man.*, 50(4):568–583, 2014.

[23] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. In *SIGIR '09*, pages 564–571. ACM, 2009.

[24] M. Larson, E. Newman, and G. Jones. Overview of VideoCLEF 2009. In *CLEF '09*. Springer, 2010.

[25] M. Lease. An improved markov random field model for supporting verbose queries. In *SIGIR '09*, pages 476–483, New York, NY, USA, 2009. ACM.

[26] M. Lease, J. Allan, and W. B. Croft. Regression rank: Learning to meet the opportunity of descriptive queries. In *ECIR '09*, pages 90–101. Springer, 2009.

[27] C.-J. Lee and W. B. Croft. Generating queries from user-selected text. In *Proc. of the 4th Information Interaction in Context Symposium*, pages 100–109. ACM, 2012.

[28] J. Luo, S. Zhang, and H. Yang. Win-win search: Dual-agent stochastic game in session search. In *SIGIR '14*, pages 587–596, New York, NY, USA, 2014. ACM.

[29] J. Luo, S. Zhang, X. Dong, and H. Yang. Designing states, actions, and rewards for using pomdp in session search. *ECIR '15*, 2015.

[30] D. Metzler and W. B. Croft. Combining the language model and inference network approaches to retrieval. *Inf. Proc. & Man.*, 40(5):735–750, 2004.

[31] Nielsen. State of the media: The cross-platform report. http://bit.ly/1hNXs4m, 2013.

[32] D. Odijk, E. Meij, and M. de Rijke. Feeding the second screen: Semantic linking based on subtitles. In *OAIR '13*, 2013.

[33] M.-H. Peetz and M. de Rijke. Cognitive temporal document priors. In *ECIR '13*, pages 318–330. Springer, 2013.

[34] M.-H. Peetz, E. Meij, M. de Rijke, and W. Weerkamp. Adaptive temporal query modeling. In *ECIR '12*, pages 455–458. Springer, 2012.

[35] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM '08*, pages 43–52. ACM, 2008.

[36] Razorfish. Forget mobile, think multiscreen. http://razorfish-outlook.razorfish.com/articles/forgetmobile.aspx, 2011.

[37] H.-J. Song, J. Go, S.-B. Park, and S.-Y. Park. A just-in-time keyword extraction from meeting transcripts. In *NAACL-HLT '13*, pages 888–896, 2013.

[38] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[39] C. J. C. H. Watkins. *Learning from delayed rewards.* PhD thesis, U. Cambridge, 1989.

[40] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML '09*, pages 1201–1208. ACM, 2009.